EE263 homework 4 solutions

5.2 *Complex linear algebra and least-squares.* Most of the linear algebra you have seen is unchanged when the scalars, matrices, and vectors are complex, *i.e.*, have complex entries. For example, we say a set of complex vectors $\{v_1, \ldots, v_n\}$ is dependent if there exist complex scalars $\alpha_1, \ldots, \alpha_n$, not all zero, such that $\alpha_1 v_1 + \cdots + \alpha_n v_n = 0$. There are some slight differences when it comes to the inner product and other expressions that, in the real case, involve the transpose operator. For complex matrices (or vectors) we define the *Hermitian conjugate* as the complex conjugate of the transpose. We denote this as A^*, which is equal to $(\overline{A})^T$. Thus, the ij entry of the matrix A^* is given by $\overline{(A_{ji})}$. The Hermitian conjugate of a matrix is sometimes called its *conjugate transpose* (which is a nice, explanatory name). Note that for a real matrix or vector, the Hermitian conjugate is the same as the transpose. We define the inner product of two complex vectors $u,\ v \in \mathbf{C}^n$ as

$$\langle u, v\rangle = u^* v,$$

which, in general, is a complex number. The norm of a complex vector is defined as

$$\|u\| = \sqrt{\langle u, u\rangle} = \left(|u_1|^2 + \cdots + |u_n|^2\right)^{1/2}.$$

Note that these two expressions agree with the definitions you already know when the vectors are real. The complex least-squares problem is to find the $x \in \mathbf{C}^n$ that minimizes $\|Ax - y\|^2$, where $A \in \mathbf{C}^{m\times n}$ and $y \in \mathbf{C}^m$ are given. Assuming A is full rank and skinny, the solution is $x_{\rm ls} = A^\dagger y$, where $A^\dagger$ is the (complex) pseudo-inverse of A, given by

$$A^\dagger = (A^* A)^{-1} A^*.$$

(Which also reduces to the pseudo-inverse you've already seen when A is real). There are two general approaches to dealing with complex linear algebra problems. In the first, you simply generalize all the results to work for complex matrices, vectors, and scalars. Another approach is to represent complex matrices and vectors using real matrices and vectors of twice the dimensions, and then you apply what you already know about real linear algebra. We'll explore that idea in this problem. We associate with a complex vector $u \in \mathbf{C}^n$ a *real* vector $\tilde{u} \in \mathbf{R}^{2n}$, given by

$$\tilde{u} = \begin{bmatrix} \Re u \\ \Im u \end{bmatrix}.$$

We associate with a complex matrix $A \in \mathbf{C}^{m\times n}$ the *real* matrix $\tilde{A} \in \mathbf{R}^{2m\times 2n}$ given by

$$\tilde{A} = \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix}.$$

(a) What is the relation between $\langle u, v\rangle$ and $\langle \tilde{u}, \tilde{v}\rangle$? Note that the first inner product involves complex vectors and the second involves real vectors.

(b) What is the relation between $\|u\|$ and $\|\tilde{u}\|$?

(c) What is the relation between Au (complex matrix-vector multiplication) and $\tilde{A}\tilde{u}$ (real matrix-vector multiplication)?

(d) What is the relation between $\tilde{A}^T$ and A^*?

(e) Using the results above, verify that $A^\dagger y$ solves the complex least-squares problem of minimizing $\|Ax - y\|$ (where A, x, y are complex). Express $A^\dagger y$ in terms of the real and imaginary parts of A and y. (You don't need to simplify your expression; you can leave block matrices in it.)

Solution:

(a) We have

$$\begin{aligned}\langle u, v\rangle &= u^* v\\ &= (\Re u - j\Im u)^T(\Re v + j\Im v)\\ &= \Re u^T \Re v + \Im u^T \Im v + j\left(-\Im u^T \Re v + \Re u^T \Im v\right)\end{aligned}$$

$$\begin{aligned}\langle \tilde{u}, \tilde{v}\rangle &= \begin{bmatrix} \Re u^T & \Im u^T \end{bmatrix}\begin{bmatrix} \Re v \\ \Im v \end{bmatrix}\\ &= \Re u^T \Re v + \Im u^T \Im v\end{aligned}$$

We thus have $\Re\langle u, v\rangle = \langle \tilde{u}, \tilde{v}\rangle$.

(b) We have

$$\|u\|^2 = u^* u = (\Re u - j\Im u)^T(\Re u - j\Im u) = \Re u^T \Re u + \Im u^T \Im u = \|\tilde{u}\|^2.$$

(c) We have

$$\begin{aligned}Au &= (\Re A + k\Im A)(\Re u + j\Im u)\\ &= (\Re A \Re u - \Im A \Im u) + j(\Re A \Im u + \Im u \Re u).\end{aligned}$$

Now let's see what $\tilde{A}\tilde{u}$ works out to be:

$$\begin{aligned}\tilde{A}\tilde{u} &= \begin{bmatrix} \Re A & -\Im A \\ \Im A & \Re A \end{bmatrix}\begin{bmatrix} \Re u \\ \Im u \end{bmatrix}\\ &= \begin{bmatrix} \Re A \Re u - \Im A \Im u \\ \Re A \Im u + \Im u \Re u \end{bmatrix}.\end{aligned}$$

Thus, the real $2n$ vector $\tilde{A}\tilde{u}$ corresponds to the complex n vector Au.

(d) The real $2n \times 2n$ matrix $\tilde{A}^T$ corresponds exactly to the complex $n \times n$ matrix A^*.

(e) We can express the complex least squares problem

$$\text{minimize } \|Ax - y\|$$

in terms of the double size, real problem

$$\text{minimize } \|\tilde{A}\tilde{x} - \tilde{y}\|,$$

since norms and matrix multiplication are preserved in this correspondence. Therefore the solution is

$$\tilde{x} = (\tilde{A}^T\tilde{A})^{-1}\tilde{A}^T\tilde{y}.$$

This gives the solution in terms of the real and imaginary parts; we just have to work out what it means using complex vectors. First of all, $\tilde{A}^T\tilde{y}$ corresponds to A^*y. The $2n \times 2n$ real matrix $\tilde{A}^T\tilde{A}$ corresponds to the $n \times n$ complex matrix A^*A. Although we didn't verify it above (but we should have . . . I'll add it to the problem for future years) inverses correspond as well. To see this, we first show that matrix products correspond. In other words, if A and B are complex matrices, then the complex matrix AB corresponds to the real matrix $\tilde{A}\tilde{B}$. To show that inverses correspond, we apply this result to the matrix equation $AB = I$, where A and B are complex square matrices (so, of course, B is the inverse of A). By the above argument, we have $\tilde{A}\tilde{B} = I$, so $\tilde{B}$ is the inverse of $\tilde{A}$. Now we can put it all together. The correspondence between complex matrices, and the (double-sized) real matrices preserves matrix product, sums, norm, Hermitian conjugate becomes transpose, inverses are preserved, etc. To minimize $\|Ax - y\|$ we can minimize $\|\tilde{A}\tilde{x} - \tilde{y}\|$. That's a real least-squares problem, with solution

$$\tilde{x} = (\tilde{A}^T\tilde{A})^{-1}\tilde{A}^T y.$$

This correponds, according to our discussion above, to the complex equation

$$x = (A^*A)^{-1}A^*y,$$

which is exactly $x = A^\dagger y$, as claimed.

Note: In order to solve a complex linear algebra problem, it is usually not necessary to convert it into a real one in the way shown in this problem. Many numerical linear algebra codes and software directly handle complex matrices. For example, Matlab handles complex matrices and vectors automatically; you don't need to do the conversion described in the problem. Indeed, in Matlab `pinv(A)` gives the pseudo-inverse ($A^\dagger$) of a (skinny, full rank) complex matrix. In Matlab you can solve the complex least-squares problem of minimizing $\|Ax - y\|$ over complex x using the four ASCII characters `x=A\y`.

6.2 *Optimal control of unit mass.* In this problem you will use Matlab to solve an optimal control problem for a force acting on a unit mass. Consider a unit mass at position $p(t)$ with velocity $\dot{p}(t)$, subjected to force $f(t)$, where $f(t) = x_i$ for $i - 1 < t \leq i$, for $i = 1, \ldots, 10$.

(a) Assume the mass has zero initial position and velocity: $p(0) = \dot{p}(0) = 0$. Find x that minimizes

$$\int_{t=0}^{10} f(t)^2 \, dt$$

subject to the following specifications: $p(10) = 1$, $\dot{p}(10) = 0$, and $p(5) = 0$. Plot the optimal force f, and the resulting p and $\dot{p}$. Make sure the specifications are satisfied. Give a short intuitive explanation for what you see.

(b) Assume the mass has initial position $p(0) = 0$ and velocity $\dot{p}(0) = 1$. Our goal is to bring the mass near or to the origin at $t = 10$, at or near rest, *i.e.*, we want

$$J_1 = p(10)^2 + \dot{p}(10)^2,$$

small, while keeping

$$J_2 = \int_{t=0}^{10} f(t)^2 \, dt$$

small, or at least not too large. Plot the optimal trade-off curve between J_2 and J_1. Check that the end points make sense to you. *Hint:* the parameter μ has to cover a very large range, so it usually works better in practice to give it a logarithmic spacing, using, *e.g.*, `logspace` in Matlab. You don't need more than 50 or so points on the trade-off curve.

Your solution to this problem should consist of a clear written narrative that explains what you are doing, and gives formulas symbolically; the Matlab source code you devise to find the numerical answers, along with comments explaining it all; and the final plots produced by Matlab. *Solution:*

(a) First note that $\int_0^{10} f(t)^2 dt$ is nothing but $\|x\|^2$ because

$$\begin{aligned}
\int_0^{10} f(t)^2 dt &= \int_0^1 x_1^2 \, dt + \int_1^2 x_2^2 \, dt + \cdots + \int_9^{10} x_{10}^2 \, dt \\
&= x_1^2 + x_2^2 + \cdots + x_{10}^2 \\
&= \|x\|^2.
\end{aligned}$$

The constraints $p(10) = 1$, $\dot{p}(10) = 0$, and $p(5) = 0$ can be expressed as linear constraints in the force program x. From homework 1 we know that

$$\begin{bmatrix} p(10) \\ \dot{p}(10) \\ p(5) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 9.5 & 8.5 & 7.5 & 6.5 & 5.5 & 4.5 & 3.5 & 2.5 & 1.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 4.5 & 3.5 & 2.5 & 1.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x. \quad (1)$$

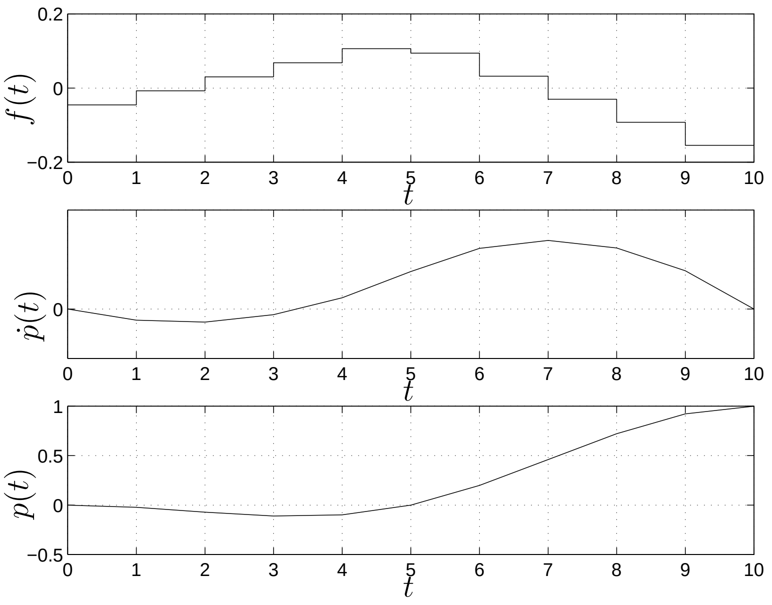
Therefore the optimal x is given by the minimizer of $\|x\|$ subject to (1). In other words, the optimal x is the minimum norm solution $x = x_{\text{ln}}$ of (1). x can be calculated using Matlab as follows:

```
>> a1=linspace(9.5,0.5,10);
>> a2=ones(1,10);
>> a3=[linspace(4.5,0.5,5), 0 0 0 0 0];
>> A=[a1;a2;a3]
A =
Columns 1 through 7
9.5000    8.5000    7.5000    6.5000    5.5000    4.5000    3.5000
1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
4.5000    3.5000    2.5000    1.5000    0.5000         0         0
Columns 8 through 10
2.5000    1.5000    0.5000
1.0000    1.0000    1.0000
0         0         0
>> y=[1;0;0]
y =
1
0
0
>> x=pinv(A)*y
x =
-0.0455
-0.0076
0.0303
0.0682
0.1061
0.0939
0.0318
-0.0303
-0.0924
-0.1545
>> A*x
ans =
1.0000
0.0000
-0.0000
>> norm(x)
ans =
0.2492
>>
```

Note that between times $t = i$ and $t = i + 1$ for $i = 0, \ldots, 9$ the force f is piecewise constant, the velocity $\dot{p}$ is piecewise linear, and the position p is piecewise quadratic. The following Matlab commands plot f, p and $\dot{p}$.

```
>> figure
>> subplot(3,1,1)
>> stairs([x;0])
>> grid on
>> xlabel('t')
>> ylabel('f(t)')
>> T1=toeplitz([ones(10,1)],[1,zeros(1,9)])
T1 =
1     0     0     0     0     0     0     0     0     0
1     1     0     0     0     0     0     0     0     0
1     1     1     0     0     0     0     0     0     0
1     1     1     1     0     0     0     0     0     0
1     1     1     1     1     0     0     0     0     0
1     1     1     1     1     1     0     0     0     0
1     1     1     1     1     1     1     0     0     0
1     1     1     1     1     1     1     1     0     0
1     1     1     1     1     1     1     1     1     0
1     1     1     1     1     1     1     1     1     1
>> p_dot=T1*x
p_dot =
-0.0455
-0.0530
-0.0227
0.0455
0.1515
0.2455
0.2773
0.2470
0.1545
0.0000
>> subplot(3,1,2)
>> plot(linspace(0,10,11),[0;p_dot])
>> grid on
>> xlabel('t')
>> ylabel('p_dot(t)')
>> T2=toeplitz(linspace(0.5,9.5,10)',[0.5,zeros(1,9)])
T2 =
Columns 1 through 7
0.5000         0         0         0         0         0         0
1.5000    0.5000         0         0         0         0         0
2.5000    1.5000    0.5000         0         0         0         0
3.5000    2.5000    1.5000    0.5000         0         0         0
```

```
4.5000    3.5000    2.5000    1.5000    0.5000         0         0
5.5000    4.5000    3.5000    2.5000    1.5000    0.5000         0
6.5000    5.5000    4.5000    3.5000    2.5000    1.5000    0.5000
7.5000    6.5000    5.5000    4.5000    3.5000    2.5000    1.5000
8.5000    7.5000    6.5000    5.5000    4.5000    3.5000    2.5000
9.5000    8.5000    7.5000    6.5000    5.5000    4.5000    3.5000
Columns 8 through 10
0         0         0
0         0         0
0         0         0
0         0         0
0         0         0
0         0         0
0         0         0
0.5000         0         0
1.5000    0.5000         0
2.5000    1.5000    0.5000
>> p=T2*x
p =
-0.0227
-0.0720
-0.1098
-0.0985
-0.0000
0.1985
0.4598
0.7220
0.9227
1.0000
>> subplot(3,1,3)
>> plot(linspace(0,10,11),[0;p])
>> grid on
>> xlabel('t')
>> ylabel('p(t)')
```



Note that the plot for $p(t)$ is not quite right because the `plot` command in Matlab plots $p(t)$ as being piecewise linear and not piecewise quadratic. It is interesting that the optimal force is such that $p(t) < 0$ for $0 < t < 5$ which means that the mass doesn't stay at position zero for $0 < t < 5$. It backups a little bit so the second time it reaches position zero it has positive velocity.

(b) In this case there are two competing objectives that we want to keep small. $J_2 = \|x\|^2$ as it was shown above. To express J_1 we rewrite the motion equations, this time for $p(0) = 0$ and $\dot{p}(0) = 1$. It is easy to verify that:

$$\begin{bmatrix} p(10) \\ \dot{p}(10) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 9.5 & 8.5 & 7.5 & 6.5 & 5.5 & 4.5 & 3.5 & 2.5 & 1.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} x + \begin{bmatrix} 10 \\ 1 \end{bmatrix}. \quad (2)$$

Therefore, to minimize J_1 we have to minimize $\|Ax - y\|^2$, where $y = \begin{bmatrix} -10 \\ -1 \end{bmatrix}$. So, all we need is to calculate the minimizer x_μ of the expression $J_1 + \mu J_2 = \|Ax - y\|^2 + \mu\|x\|^2$ for different values of μ. As it was proven in the lecture notes,

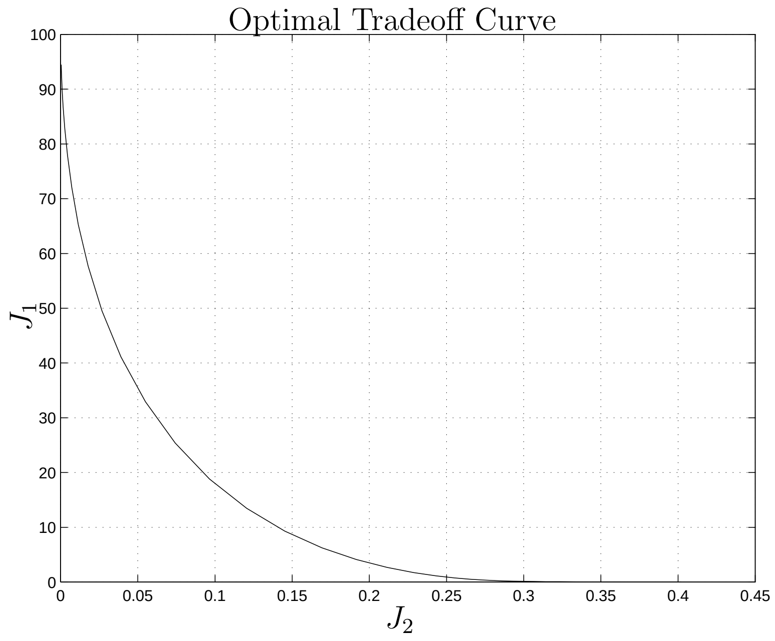
$$x_\mu = (A^T A + \mu I)^{-1} A^T y.$$

We use Matlab to calculate x_μ and the resulting pairs (J_1, J_2) and plot the tradeoff curve. A sample Matlab script is given below:

```
clear; clf;
N = 50;
a1 = linspace(9.5,0.5,10); a2 = ones(1,10); A  = [a1; a2];
y = [-10; -1];
F = eye(10);
```

```
mu = logspace(-2, 4, N);
for i=1:N;
f = inv(A'*A + mu(i)*F'*F)*A'*y;
J_1(i) = (norm(A*f - y))^2;
J_2(i) = (norm(f))^2;
end;
plot(J_2,J_1); title('Optimal Tradeoff Curve'); xlabel('J_2 =
||x||^2') ylabel('J_1 = p(10)^2 + p_dot(10)^2');
```

The resulting tradeoff curve is shown below:



We can now use the curve to decide which x_μ we are going to use. The choice will depend on the importance we decide to attach to each of the two objectives. A final comment: Note that there is a value of x_μ after which J_1 becomes equal to 0. This value is the minimum-norm solution, because it corresponds to the minimum $J_2 = \|x\|^2$ for which $Ax = y$. On the other hand, we cannot drive the system to or near the desired state without spending some energy, so J_1 is large for small values of J_2. For $J_2 = 0$, J_1 is equal to $\left\| \begin{bmatrix} 10 \\ 1 \end{bmatrix} \right\|^2 = 101$.

6.5 *Interpolation with rational functions.* In this problem we consider a function $f : \mathbf{R} \to \mathbf{R}$ of the form

$$f(x) = \frac{a_0 + a_1 x + \cdots + a_m x^m}{1 + b_1 x + \cdots + b_m x^m},$$

where $a_0, \ldots, a_m$, and $b_1, \ldots, b_m$ are parameters, with either $a_m \neq 0$ or $b_m \neq 0$. Such a function is called a *rational function of degree m*. We are given data points $x_1, \ldots, x_N \in \mathbf{R}$ and $y_1, \ldots, y_N \in \mathbf{R}$, where $y_i = f(x_i)$. The problem is to find a rational function of smallest degree that is consistent with this data. In other words, you are to find m, which should be as small as possible, and $a_0, \ldots, a_m$, $b_1, \ldots, b_m$, which satisfy $f(x_i) = y_i$. Explain how you will solve this problem, and then carry out your method on the problem data given in `ri_data.m`. (This contains two vectors, `x` and `y`, that give the values $x_1, \ldots, x_N$, and $y_1, \ldots, y_N$, respectively.) Give the value of m you find, and the coefficients $a_0, \ldots, a_m$, $b_1, \ldots, b_m$. Please show us your verification that $y_i = f(x_i)$ holds (possibly with some small numerical errors). *Solution.* The interpolation condition $f(x_i) = y_i$ is

$$f(x_i) = \frac{a_0 + a_1 x_i + \cdots + a_m x_i^m}{1 + b_1 x_i + \cdots + b_m x_i^m} = y_i, \quad i = 1, \ldots, N.$$

This is a set of complicated nonlinear functions of the coefficient vectors a and b. If we multiply out by the denominator, we get

$$y_i(1 + b_1 x_i + \cdots + b_m x_i^m) - (a_0 + a_1 x_i + \cdots + a_m x_i^m) = 0, \quad i = 1, \ldots, N.$$

These equations are *linear* in a and b. We can write these equations in matrix form as

$$G \begin{bmatrix} a \\ b \end{bmatrix} = y, \tag{3}$$

where

$$a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix},$$

and

$$G = \begin{bmatrix} 1 & x_1 & \cdots & x_1^m & -y_1 x_1 & -y_1 x_1^2 & \cdots & -y_1 x_1^m \\ 1 & x_2 & \cdots & x_2^m & -y_2 x_2 & -y_2 x_2^2 & \cdots & -y_2 x_2^m \\ 1 & x_3 & \cdots & x_3^m & -y_3 x_3 & -y_3 x_3^2 & \cdots & -y_3 x_3^m \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & x_N & \cdots & x_N^m & -y_N x_N & -y_N x_N^2 & \cdots & -y_N x_N^m \end{bmatrix}.$$

Thus, we can interpolate the data if and only if the equation (3) has a solution. Our problem is to find the smallest m for which these linear equations can be solved, or, equivalently, $y \in \mathcal{R}(G)$. We can do this by finding the smallest value of m for which

$$\mathbf{Rank}(G) = \mathbf{Rank}([G\ y]).$$

Then we can find a set of coefficients by solving the equation (3) for a and b. The following Matlab code carries out this method.

```
clear all
close all
```

```
rat_int_data
for m=1:20 %we sweep over different values of m
G=ones(N,1);
for i=1:m;
G=[G x.^i];
end
for i=1:m
G=[G -x.^i.*y];
end
if rank(G)== rank([G y])
break;
end
end
ab=G\y;
a=ab(1:m+1);
b=ab(m+2:2*m+1);
m
a
b
```

Matlab produces the following output:

```
m =
5
a =
0.2742
1.0291
1.2906
-5.8763
-2.6738
6.6845
b =
-1.2513
-6.5107
3.2754
17.3797
6.6845
```

Thus, we find that $m = 5$ is the lowest order rational function that interpolates the data, and a rational function that interpolates the data is given by

$$f(x) = \frac{0.2742 + 1.0291x + 1.2906x^2 - 5.8763x^3 - 2.6738x^4 + 6.6845x^5}{1 - 1.2513x - 6.5107x^2 + 3.2754x^3 + 17.3797x^4 + 6.6845x^5}$$

(we have truncated the coefficients to shorten the formula). We now verify that this expression interpolates the given points.

```
num=zeros(N,1);
for i=1:m+1
num=a(i)*x.^(i-1)+num;
end
den=ones(N,1);
for i=1:m
den=b(i)*x.^i+den;
end
f=num./den;
err=norm(f-y)
```

Matlab produces the following output

```
err =
7.7649e-14.
```

This shows that the output is interpolated up to numerical precision.

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

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

where

- y_i is the ith (scalar) measurement
- $x \in \mathbf{R}^n$ is the vector of parameters we wish to estimate from the measurements
- v_i is the sensor or measurement error of the ith 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 ith 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 $x \in \mathbf{R}^n$, the offset $\alpha \in \mathbf{R}$, and the drift coefficient $\beta \in \mathbf{R}$. 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. *Solution:*
Substituting the expression for the noise into the measurement equation gives

$$y_i = a_i^T x + \alpha + \beta iT + w_i \qquad i = 1, \ldots, m.$$

In matrix form we can write this as

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} a_1^T & 1 & T \\ a_2^T & 1 & 2T \\ \vdots & \vdots & \vdots \\ a_m^T & 1 & mT \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x \\ \alpha \\ \beta \end{bmatrix}}_{\tilde{x}}.$$

If $\tilde{A}$ is skinny ($m \geq n+2$) and full-rank, least-squares can be used to estimate x, α, and β. In that case,

$$\tilde{x}_{\text{ls}} = \begin{bmatrix} x_{\text{ls}} \\ \alpha_{\text{ls}} \\ \beta_{\text{ls}} \end{bmatrix} = (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T y.$$

The requirement that $\tilde{A}$ be skinny (or at least, square) makes perfect sense: you can't extract $n+2$ parameters (*i.e.*, x, α, β) from fewer than $n+2$ measurements. Even is A is skinny and full-rank, $\tilde{A}$ may not be. For example, with

$$A = \begin{bmatrix} 2 & 0 \\ 2 & 1 \\ 2 & 0 \\ 2 & 1 \end{bmatrix},$$

we have

$$\tilde{A} = \begin{bmatrix} 2 & 0 & 1 & T \\ 2 & 1 & 1 & 2T \\ 2 & 0 & 1 & 3T \\ 2 & 1 & 1 & 4T \end{bmatrix},$$

which is not full-rank. In this example we can understand exactly what happened. The first column of A, which tells us how the sensors respond to the first component of x, has exactly the same form as an offset, so it is not possible to separate the offset from the signal induced by x_1. In the general case, $\tilde{A}$ is not full rank only if some linear combinations of the sensor signals looks exactly like an offset or drift, or some linear combination of offset and drift. Some people asserted that $\tilde{A}$ is full rank if the vector of ones and the vector $(T, 2T, \ldots, mT)$ are not in the span of the columns of A. This is false. Several people went into the case when $\tilde{A}$ is not full rank in great detail, suggesting regularization and other things. We weren't really expecting you to go into such detail about this case. Most people who went down this route, however, failed to mention the *most important thing* about what happens. When $\tilde{A}$ is not full rank, *you cannot separate the offset and drift parameters from the parameter x by any means at all.* Regularization means that the numerics will work, but the result will be quite meaningless. Some people pointed out that the normal eqautions always have a solution, even when $\tilde{A}$ is not full rank. Again, this is true, but the most important thing here is that even if you solve the normal equations, the results are meaningless, since you cannot separate out the offset and drift terms from the parameter x. Accounting for known noise characteristics (like offset and drift) can greatly improve estimation accuracy. The following matlab code shows an example.

```
T = 60;              % Time between samples
alpha = 3;           % sensor offset
beta =  .05;         % sensor drift constant
num_x = 3;
num_y = 8;
A =[ 1     4     0
2     0     1
-2    -2     3
-1     1    -4
-3     1     1
0    -2     2
3     2     3
0    -4    -6   ]; % matrix whose rows are a_i^T
x = [-8; 20; 5];
, with v a (Gaussian, random) noise
y = A*x;
for i = 1:num_y;
y(i) = y(i)+alpha+beta*T*i+randn;
```

```
end;
x_ls = A\y;
for i = 1:num_y
last_col(i) = T*i;
end
A = [A ones(num_y,1) last_col'];
x_ls_with_noise_model = A\y;
norm(x-x_ls)
norm(x-x_ls_with_noise_model(1:3))
```

Additional comments. Many people correctly stated that $\tilde{A}$ needed to be full rank and then presented a condition they claimed was equivalent. Unfortunately, many of these statements were incorrect. The most common error was to claim that if neither of the two column vectors that were appended to A in creating $\tilde{A}$ was in $\mathcal{R}(A)$, then $\tilde{A}$ was full rank. As a counter-example, take

$$A = \begin{bmatrix} 2 \\ 3 \\ \vdots \\ m+1 \end{bmatrix},$$

and

$$\tilde{A} = \begin{bmatrix} 2 & 1 & 1 \\ 3 & 1 & 2 \\ \vdots & \vdots & \vdots \\ m+1 & 1 & m \end{bmatrix}.$$

Since the first column of $\tilde{A}$ is the sum of the last two, $\tilde{A}$ has rank 2, not 3.

6.14 *Identifying a system from input/output data.* We consider the standard setup:

$$y = Ax + v,$$

where $A \in \mathbf{R}^{m\times n}$, $x \in \mathbf{R}^n$ is the input vector, $y \in \mathbf{R}^m$ is the output vector, and $v \in \mathbf{R}^m$ is the noise or disturance. We consider here the problem of estimating the matrix A, given some input/output data. Specifically, we are given the following:

$$x^{(1)}, \ldots, x^{(N)} \in \mathbf{R}^n, \qquad y^{(1)}, \ldots, y^{(N)} \in \mathbf{R}^m.$$

These represent N samples or observations of the input and output, respectively, possibly corrupted by noise. In other words, we have

$$y^{(k)} = Ax^{(k)} + v^{(k)}, \quad k = 1, \ldots, N,$$

where $v^{(k)}$ are assumed to be small. The problem is to estimate the (coefficients of the) matrix A, based on the given input/output data. You will use a least-squares criterion to form an estimate $\hat{A}$ of A. Specifically, you will choose as your estimate $\hat{A}$ the matrix that minimizes the quantity

$$J = \sum_{k=1}^{N} \|Ax^{(k)} - y^{(k)}\|^2$$

over A.

(a) Explain how to do this. If you need to make an assumption about the input/output data to make your method work, state it clearly. You may want to use the matrices $X \in \mathbf{R}^{n\times N}$ and $Y \in \mathbf{R}^{m\times N}$ given by

$$X = \left[\begin{array}{ccc} x^{(1)} & \cdots & x^{(N)} \end{array}\right], \qquad Y = \left[\begin{array}{ccc} y^{(1)} & \cdots & y^{(N)} \end{array}\right]$$

in your solution.

(b) On the course web site you will find some input/output data for an instance of this problem in the mfile `sysid_data.m`. Executing this mfile will assign values to m, n, and N, and create two matrices that contain the input and output data, respectively. The $n \times N$ matrix variable `X` contains the input data $x^{(1)}, \dots, x^{(N)}$ (*i.e.*, the first column of `X` contains $x^{(1)}$, etc.). Similarly, the $m \times N$ matrix `Y` contains the output data $y^{(1)}, \dots, y^{(N)}$. You must give your final estimate $\hat{A}$, your source code, and also give an explanation of what you did.

Solution.

(a) We start by expressing the objective function J as

$$\begin{aligned} J &= \sum_{k=1}^{N} \|Ax^{(k)} - y^{(k)}\|^2 \\ &= \sum_{k=1}^{N}\sum_{i=1}^{m} (Ax^{(k)} - y^{(k)})_i^2 \\ &= \sum_{k=1}^{N}\sum_{i=1}^{m} (a_i^T x^{(k)} - y_i^{(k)})^2 \\ &= \sum_{i=1}^{m} \left(\sum_{k=1}^{N} (a_i^T x^{(k)} - y_i^{(k)})^2 \right), \end{aligned}$$

where a_i is the ith row of A. The last expression shows that J is a sum of expressions J_i (shown in parentheses), each of which only depends on a_i. This means that to minimize J, we can minimize each of these expressions separately. That makes sense: we can estimate the rows of A separately. Now let's see how to minimize

$$J_i = \sum_{k=1}^{N} (a_i^T x^{(k)} - y_i^{(k)})^2,$$

which is the contribution to J from the ith row of A. First we write it as

$$J_i = \left\| \begin{bmatrix} x^{(1)T} \\ \vdots \\ x^{(N)T} \end{bmatrix} a_i - \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(N)} \end{bmatrix} \right\|^2 .$$

Now that we have the problem in the standard least-squares format, we're pretty much done. Using the matrix $X \in \mathbf{R}^{n\times N}$ given by

$$X = \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix},$$

we can express the estimate as

$$\hat{a}_i = (XX^T)^{-1}X \begin{bmatrix} y_i^{(1)} \\ \vdots \\ y_i^{(N)} \end{bmatrix}.$$

Using the matrix $Y \in \mathbf{R}^{m\times N}$ given by

$$Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(N)} \end{bmatrix},$$

we can express the estimate of A as

$$\hat{A}^T = (XX^T)^{-1}XY^T.$$

Transposing this gives the final answer:

$$\hat{A} = YX^T(XX^T)^{-1}.$$

(b) Once you have the neat formula found above, it's easy to get matlab to compute the estimate. It's a little inefficient, but perfectly correct, to simply use

```
Ahat = Y*X'*inv(X*X');
```

This yields the estimate

$$\hat{A} = \begin{bmatrix} 2.03 & 5.02 & 5.01 \\ 0.01 & 7 & 1.01 \\ 7.04 & 0 & 6.94 \\ 7 & 3.98 & 4 \\ 9.01 & 1.04 & 7 \\ 4.01 & 3.96 & 9.03 \\ 4.99 & 6.97 & 8.03 \\ 7.94 & 6.09 & 3.02 \\ 0.01 & 8.97 & -0.04 \\ 1.06 & 8.02 & 7.03 \end{bmatrix}.$$

Once you've got $\hat{A}$, it's a good idea to check the residuals, just to make sure it's reasonable, by comparing it to

$$\sum_{k=1}^{N} \|y^{(k)}\|^2.$$

Here we get $(64.5)^2$, around 4.08%. There are several other ways to compute $\hat{A}$ in matlab. You can calculate the rows of $\hat{A}$ one at a time, using

```
a1hat = (X'\(Y(i,:)'))';
```

In fact, the backslash operator in matlab solves multiple least-squares problems at once, so you can use

```
AhatT = X' \ (Y');
Ahat = AhatT';
```

or

```
Ahat = (X'\(Y'))';
```

In any case, it's not exactly a long matlab program ...

6.26 *Estimating direction and amplitude of a light beam.* A light beam with (nonnegative) amplitude a comes from a direction $d \in \mathbf{R}^3$, where $\|d\| = 1$. (This means the beam travels in the direction $-d$.) The beam falls on $m \geq 3$ photodetectors, each of which generates a scalar signal that depends on the beam amplitude and direction, and the direction in which the photodetector is pointed. Specifically, photodetector i generates an output signal p_i, with

$$p_i = a\alpha \cos\theta_i + v_i,$$

where θ_i is the angle between the beam direction d and the outward normal vector q_i of the surface of the ith photodetector, and α is the photodetector sensitivity. You can interpret $q_i \in \mathbf{R}^3$, which we assume has norm one, as the direction the ith photodetector is pointed. We assume that $|\theta_i| < 90^\circ$, *i.e.*, the beam illuminates the top of the photodetectors. The numbers v_i are small measurement errors.

You are given the photodetector direction vectors $q_1, \ldots, q_m \in \mathbf{R}^3$, the photodetector sensitivity α, and the noisy photodetector outputs, $p_1, \ldots, p_m \in \mathbf{R}$. Your job is to estimate the beam direction $d \in \mathbf{R}^3$ (which is a unit vector), and a, the beam amplitude.

To describe unit vectors $q_1, \ldots, q_m$ and d in $\mathbf{R}^3$ we will use azimuth and elevation, defined as follows:

$$q = \begin{bmatrix} \cos\phi\cos\theta \\ \cos\phi\sin\theta \\ \sin\phi \end{bmatrix}.$$

Here ϕ is the elevation (which will be between 0° and 90°, since all unit vectors in this problem have positive 3rd component, *i.e.*, point upward). The azimuth angle θ, which varies from $0°$ to $360°$, gives the direction in the plane spanned by the first and second coordinates. If $q = e_3$ (*i.e.*, the direction is directly up), the azimuth is undefined.

(a) Explain how to do this, using a method or methods from this class. The simpler the method the better. If some matrix (or matrices) needs to be full rank for your method to work, say so.

(b) Carry out your method on the data given in `beam_estim_data.m`. This mfile defines `p`, the vector of photodetector outputs, a vector `det_az`, which gives the azimuth angles of the photodetector directions, and a vector `det_el`, which gives the elevation angles of the photodetector directions. Note that both of these are given in *degrees*, not radians. Give your final estimate of the beam amplitude a and beam direction d (in azimuth and elevation, in degrees).

Solution.

(a) Since $\cos\theta_i = q_i^T d/(\|q_i\|\|d\|) = q_i^T d$ (using $\|q_i\| = \|d\| = 1$), we have

$$p_i = a\alpha q_i^T d + v_i.$$

In this equation we are given p_i, α, and q_i; we are to estimate $a \in \mathbf{R}$ and $d \in \mathbf{R}^3$, using the given information that v_i is small. At first glance it looks like a nonlinear problem, since two of the variables we need to estimate, a and d, are multiplied together in this formula.

But a little thought reveals that things are actually much simpler. Let's define $x \in \mathbf{R}^3$ as $x = ad$. We can just as well work with x since given any nonzero $x \in \mathbf{R}^3$, we have $a = \|x\|$ and $d = x/\|x\|$. (Conversely, given any a and d, we have $x = ad$ by definition.)

We can therefore express the problem in terms of the variable x as

$$p = \alpha \begin{bmatrix} q_1^T \\ \vdots \\ q_m^T \end{bmatrix} x + v = \alpha Q x + v,$$

where $p = (p_1, \ldots, p_m)$, $v = (v_1, \ldots, v_m)$, and Q is the matrix with rows q_i^T.

Now we can get a reasonable guess of x using least-squares. Assuming Q is full rank, we have the least-squares estimate

$$\hat{x} = (1/\alpha)(Q^T Q)^{-1} Q^T p.$$

We then form estimates of a and d using $\hat{a} = \|\hat{x}\|$, $\hat{d} = \hat{x}/\|\hat{x}\|$.

The matrix Q is full rank (*i.e.*, rank 3), if and only if the vectors $\{q_1, \ldots, q_m\}$ span $\mathbf{R}^3$. In other words, we cannot have all photodetectors pointing in a common plane.

(b) The following code solves the problem for the given data.

```
beam_estim_data

for i=1:m
    Q(i,:)=[  cosd(det_el(i))*cosd(det_az(i)),...
              cosd(det_el(i))*sind(det_az(i)),...
              sind(det_el(i))  ];
end

xhat=(1/alpha)*(Q\p);
ahat=norm(xhat);
dhat=xhat/norm(xhat);

elevation=asind(dhat(3))
azimuth=acosd(dhat(1)/cosd(elevation))
```

The result is $\hat{a} = 5.0107$, $\hat{\phi}_d = 38.7174$, and $\hat{\theta}_d = 77.6623$.

7.3 *Curve-smoothing.* We are given a function $F : [0, 1] \to \mathbf{R}$ (whose graph gives a curve in $\mathbf{R}^2$). Our goal is to find another function $G : [0, 1] \to \mathbf{R}$, which is a *smoothed* version of F. We'll judge the smoothed version G of F in two ways:

- *Mean-square deviation from F,* defined as

$$D = \int_0^1 (F(t) - G(t))^2 \, dt.$$

- *Mean-square curvature,* defined as

$$C = \int_0^1 G''(t)^2 \, dt.$$

We want *both* D and C to be small, so we have a problem with two objectives. In general there will be a trade-off between the two objectives. At one extreme, we can choose $G = F$, which makes $D = 0$; at the other extreme, we can choose G to be an affine function (*i.e.*, to have $G''(t) = 0$ for all $t \in [0, 1]$), in which case $C = 0$. The problem is to identify the optimal trade-off curve between C and D, and explain how to find smoothed functions G on the optimal trade-off curve. To reduce the problem to a finite-dimensional one, we will represent the functions F and G (approximately) by vectors $f, \ g \in \mathbf{R}^n$, where

$$f_i = F(i/n), \quad g_i = G(i/n).$$

You can assume that n is chosen large enough to represent the functions well. Using this representation we will use the following objectives, which approximate the ones defined for the functions above:

- *Mean-square deviation,* defined as

$$d = \frac{1}{n}\sum_{i=1}^{n}(f_i - g_i)^2.$$

- *Mean-square curvature,* defined as

$$c = \frac{1}{n-2}\sum_{i=2}^{n-1}\left(\frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2}\right)^2.$$

In our definition of c, note that

$$\frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2}$$

gives a simple approximation of $G''(i/n)$. You will only work with this approximate version of the problem, *i.e.*, the vectors f and g and the objectives c and d.

(a) Explain how to find g that minimizes $d+\mu c$, where $\mu \geq 0$ is a parameter that gives the relative weighting of sum-square curvature compared to sum-square deviation. Does your method always work? If there are some assumptions you need to make (say, on rank of some matrix, independence of some vectors, etc.), state them clearly. Explain how to obtain the two extreme cases: $\mu = 0$, which corresponds to minimizing d without regard for c, and also the solution obtained as $\mu \to \infty$ (*i.e.*, as we put more and more weight on minimizing curvature).

(b) Get the file `curve_smoothing.m` from the course web site. This file defines a specific vector f that you will use. Find and plot the optimal trade-off curve between d and c. Be sure to identify any critical points (such as, for example, any intersection of the curve with an axis). Plot the optimal g for the two extreme cases $\mu = 0$ and $\mu \to \infty$, and for three values of μ in between (chosen to show the trade-off nicely). On your plots of g, be sure to include also a plot of f, say with dotted line type, for reference. Submit your Matlab code.

Solution:

(a) Let's start with the two extreme cases. When $\mu = 0$, finding g to minimize $d+\mu c$ reduces to finding g to minimize d. Since d is a sum of squares, $d \geq 0$. Choosing $g = f$ trivially achieves $d = 0$. This makes perfect sense: to minimize the deviation measure, just take the smoothed version to be the same as the original function. This yields zero deviation, naturally, but also, it yields no smoothing! Next, consider the extreme case where $\mu \to \infty$. This means we want to make the curvature as small as possible. Can we drive it to zero? The answer is yes, we can: the curvature is zero if and only if g is an affine function, *i.e.*, has the form $g_i = ai + b$ for some constants a and b. There are lots of vectors g that have this form; in fact, we have one for every pair of numbers a, b. All of these vectors g make c zero. Which one do we choose? Well, even if μ is huge, we still have a small contribution to $d+\mu c$ from d, so among all g that make $c=0$, we'd like the one that minimizes d. Basically, we want to find the best affine approximation, in the sum of squares sense, to f. We want to find a and b that minimize

$$\left\| f - A \begin{bmatrix} a \\ b \end{bmatrix} \right\| \text{ where } A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ \vdots & \vdots \\ n & 1 \end{bmatrix}.$$

For $n \geq 2$, A is skinny and full rank, and a and b can be found using least-squares. Specifically, $[a\ b]^T = (A^TA)^{-1}A^Tf$. In the general case, minimizing $d+\mu c$, is the same as choosing g to minimize

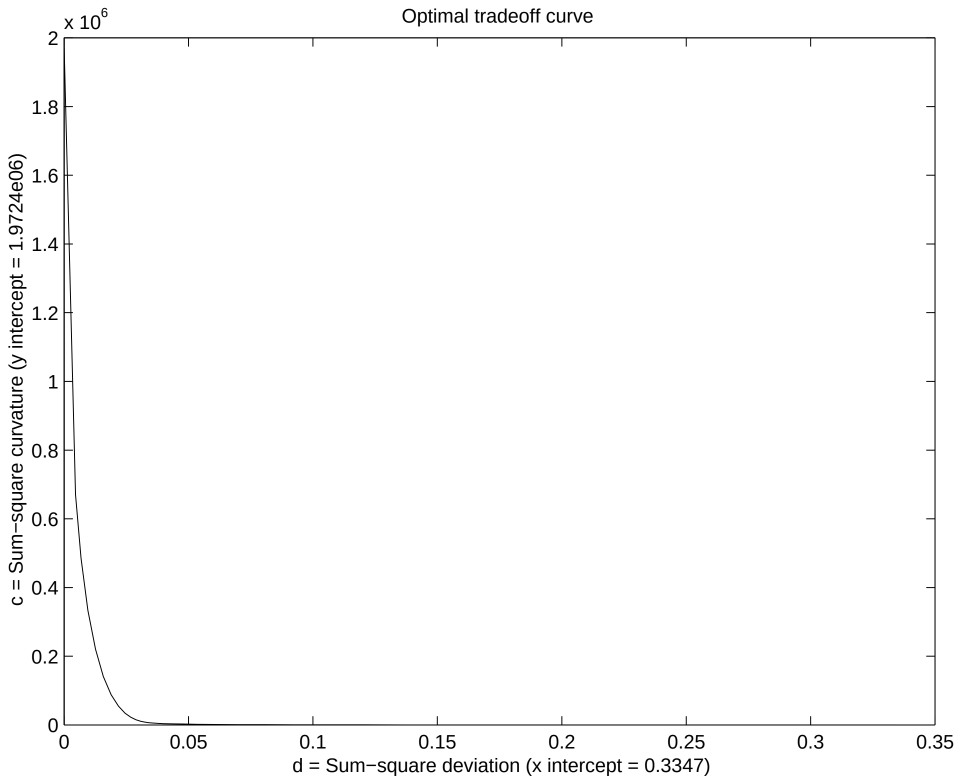
$$\left\| \frac{1}{\sqrt{n}} Ig - \frac{1}{\sqrt{n}} f \right\|^2 + \mu \left\| \underbrace{\frac{n^2}{\sqrt{n-2}} \begin{bmatrix} -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \end{bmatrix}}_{S} g \right\|^2 .$$

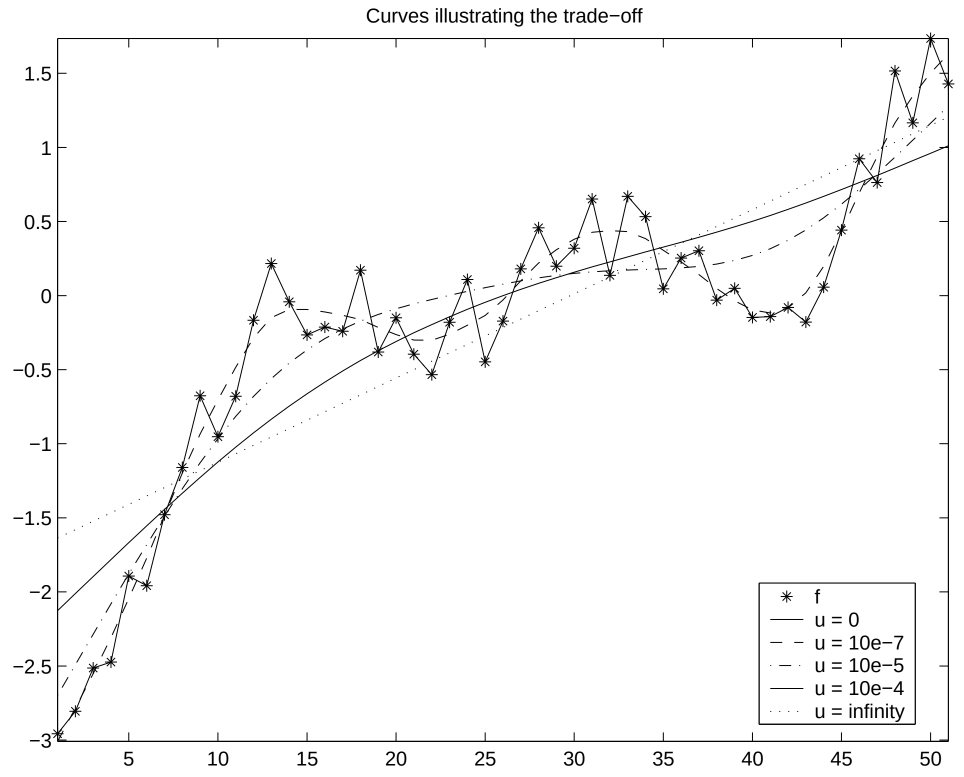
This is a multi-objective least-squares problem. The minimizing g is

$$g = (\tilde{A}^T\tilde{A})^{-1}\tilde{A}^T\tilde{y} \text{ where } \tilde{A} = \begin{bmatrix} \frac{I}{\sqrt{n}} \\ \sqrt{\mu}S \end{bmatrix} \text{ and } \tilde{y} = \begin{bmatrix} \frac{f}{\sqrt{n}} \\ 0 \end{bmatrix}.$$

The inverse of $\tilde{A}$ always always exists because I is full rank. The expression can also be written as $g = (\frac{I}{n} + \mu S^TS)^{-1}\frac{f}{n}$.

(b) The following plots show the optimal trade-off curve and the optimal g corresponding to representative μ values on the curve.





The following matlab code finds and plots the optimal trade-off curve between d

and c. It also finds and plots the optimal g for representative values of μ. As expected, when $\mu = 0$, $g = f$ and no smoothing occurs. At the other extreme, as μ goes to infinity, we get an affine approximation of f. Intermediate values of μ correspond to approximations of f with different degrees of smoothness.

```
close all;
clear all;
curve_smoothing
S = toeplitz([-1; zeros(n-3,1)],[-1 2 -1 zeros(1,n-3)]);
S = S*n^2/(sqrt(n-2));
I = eye(n);
g_no_deviation = f;
error_curvature(1) = norm(S*g_no_deviation)^2;
error_deviation(1) = 0;
u = logspace(-8,-3,30);
for i = 1:length(u)
A_tilde = [1/sqrt(n)*I; sqrt(u(i))*S];
y_tilde = [1/sqrt(n)*f; zeros(n-2,1)];
g = A_tilde\y_tilde;
error_deviation(i+1) = norm(1/sqrt(n)*I*g-f/sqrt(n))^2;
error_curvature(i+1) = norm(S*g)^2;
end
a1 = 1:n;
a1 = a1';
a2 = ones(n,1);
A = [a1 a2];
affine_param = inv(A'*A)*A'*f;
for i = 1:n
g_no_curvature(i) = affine_param(1)*i+affine_param(2);
end
g_no_curvature = g_no_curvature';
error_deviation(length(u)+2) = 1/n*norm(g_no_curvature-f)^2;
error_curvature(length(u)+2) = 0;
figure(1);
plot(error_deviation, error_curvature);
xlabel('Sum-square deviation (y intercept = 0.3347)');
ylabel('Sum-square curvature (x intercept = 1.9724e06)');
title('Optimal tradeoff curve ');
print curve_extreme.eps;
u1 = 10e-7;
A_tilde = [1/sqrt(n)*I;sqrt(u1)*S];
y_tilde = [1/sqrt(n)*f;zeros(n-2,1)];
g1 = A_tilde\y_tilde;
```

```
u2 = 10e-5;
A_tilde = [1/sqrt(n)*I;sqrt(u2)*S];
y_tilde = [1/sqrt(n)*f;zeros(n-2,1)];
g2 = A_tilde\y_tilde;
u3 = 10e-4;
A_tilde = [1/sqrt(n)*I;sqrt(u3)*S];
y_tilde = [1/sqrt(n)*f;zeros(n-2,1)];
g3 = A_tilde\y_tilde;
figure(3);
plot(f,'*');
hold;
plot(g_no_deviation);
plot(g1,'--');
plot(g2,'-.');
plot(g3,'-');
plot(g_no_curvature,':');
axis tight;
legend('f','u = 0','u = 10e-7', 'u = 10e-5', 'u = 10e-4','u = infinity',0);
title('Curves illustrating the trade-off');
print curve_tradeoff.eps;
```

Note: Several exams had a typo that defined

$$c = \frac{1}{n-1}\sum_{i=2}^{n-1}\left(\frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2}\right)^2$$

instead of

$$c = \frac{1}{n-2}\sum_{i=2}^{n-1}\left(\frac{g_{i+1} - 2g_i + g_{i-1}}{1/n^2}\right)^2.$$

The solutions above reflect the second definition. Full credit was given for answers consistent with either definition. *Some common errors*

- Several students tried to approximate f using low-degree polynomials. While fitting f to a polynomial does smooth f, it does not necessarily minimize $d+\mu c$ for some $\mu \geq 0$, nor does it illustrate the trade-off between curvature and deviation.
- In explaining how to find the g that minimizes $d + \mu c$ as $\mu \to \infty$, many people correctly observed that if $g \in \mathcal{N}(S)$, then $c = 0$. For full credit, however, solutions had to show how to choose the vector in $\mathcal{N}(S)$ that minimizes d.
- Many people chose to zoom in on a small section of the trade-off curve rather than plot the whole range from 0 to $\mu \to \infty$. Those solutions received full-credit provided they calculated the intersections with the axes (i.e. provided they found the minimum value for $d + \mu c$ when $\mu = 0$ and when $\mu \to \infty$).

8.2 *Simultaneous left inverse of two matrices.* Consider a system where

$$y = Gx, \quad \tilde{y} = \tilde{G}x$$

where $G \in \mathbf{R}^{m\times n}$, $\tilde{G} \in \mathbf{R}^{m\times n}$. Here x is some variable we wish to estimate or find, y gives the measurements with some set of (linear) sensors, and $\tilde{y}$ gives the measurements with some *alternate* set of (linear) sensors. We want to find a *reconstruction matrix* $H \in \mathbf{R}^{n\times m}$ such that $HG = H\tilde{G} = I$. Such a reconstruction matrix has the nice property that it recovers x perfectly from *either* set of measurements (y or $\tilde{y}$), *i.e.*, $x = Hy = H\tilde{y}$. Consider the specific case

$$G = \begin{bmatrix} 2 & 3 \\ 1 & 0 \\ 0 & 4 \\ 1 & 1 \\ -1 & 2 \end{bmatrix}, \quad \tilde{G} = \begin{bmatrix} -3 & -1 \\ -1 & 0 \\ 2 & -3 \\ -1 & -3 \\ 1 & 2 \end{bmatrix}.$$

Either find an explicit reconstruction matrix H, or explain why there is no such H.
Solution:
The requirements $HG = I$ and $H\tilde{G} = I$ are a set of linear equations in the variables H_{ij}. Since $H \in \mathbf{R}^{n\times m}$ there are mn unknowns; each equation of the form $HG = I$ gives n^2 equations, so all together we have $2n^2$ equations. Roughly speaking, it's reasonable to expect a solution to exist when there are more variables than equations, *i.e.*, $mn \geq 2n^2$, which implies that $m \geq 2n$. This condition makes sense: to invert two different sensor measurements we need a redundancy factor of two. Now let's look at the specific case given. Suppose that

$$H = \begin{bmatrix} h_1^T \\ h_2^2 \end{bmatrix}$$

where $h_1, h_2 \in \mathbf{R}^5$. $HG = I$ implies that $h_1^T G = e_1^T$ and $h_2^T G = e_2^T$ where e_1 and e_2 are the unit vectors in $\mathbf{R}^2$. Similarly we should have $h_1^T\tilde{G} = e_1^T$ and $h_2^T\tilde{G} = e_2^T$. In block matrix form

$$\begin{bmatrix} G^T & 0 \\ \tilde{G}^T & 0 \\ 0 & G^T \\ 0 & \tilde{G}^T \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} e_1 \\ e_1 \\ e_2 \\ e_2 \end{bmatrix}.$$

Now by defining

$$A = \begin{bmatrix} G^T & 0 \\ \tilde{G}^T & 0 \\ 0 & G^T \\ 0 & \tilde{G} \end{bmatrix}, \quad x = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}, \quad b = \begin{bmatrix} e_1 \\ e_2 \\ e_1 \\ e_2 \end{bmatrix}$$

we get the standard form $Ax = b$. If $b \in \mathcal{R}(A)$ a solution exists and H can be found. In this case, A happens to be full rank so $(AA^T)^{-1}$ exists and we can set $x = A^T(AA^T)^{-1}b$. Using Matlab:

```
>> G=[2 3;1 0;0 4;1 1;-1 2]
G =
2     3
1     0
0     4
1     1
-1     2
>> tilde_G=[-3 -1;-1 0;2 -3;-1 -3;1 2]
tilde_G =
-3    -1
-1     0
2    -3
-1    -3
1     2
>> zero=zeros(2,5)
zero =
0     0     0     0     0
0     0     0     0     0
>> A=[G' zero;tilde_G' zero;zero G';zero tilde_G']
A =
2     1     0     1    -1     0     0     0     0     0
3     0     4     1     2     0     0     0     0     0
-3    -1     2    -1     1     0     0     0     0     0
-1     0    -3    -3     2     0     0     0     0     0
0     0     0     0     0     2     1     0     1    -1
0     0     0     0     0     3     0     4     1     2
0     0     0     0     0    -3    -1     2    -1     1
0     0     0     0     0    -1     0    -3    -3     2
>> b=[1;0;1;0;0;1;0;1]
b =
1
0
1
0
0
1
0
1
>> x=pinv(A)*b
x =
-0.2782
2.0944
```

```
0.8609
-1.2284
-0.6904
0.1616
0.2273
0.0808
-0.3030
0.2475
>> H=[x(1:5)'; x(6:10)']
H =
-0.2782    2.0944    0.8609   -1.2284   -0.6904
0.1616    0.2273    0.0808   -0.3030    0.2475
>>
```

Finally we can check that $HG = H\tilde{G} = I$ using Matlab:

```
>> H*G
ans =
1.0000   -0.0000
-0.0000    1.0000
>> H*tilde_G
ans =
1.0000    0.0000
0.0000    1.0000
>>
```

Of course, there are other solutions as well. Indeed, since there are 10 variables and 8 independent equations, there is a 2 dimensional set of H's that satisfy the required condition.