

EE263 homework 5

1. *Curve-smoothing*. We are given a function $F : [0, 1] \rightarrow \mathbf{R}$ (whose graph gives a curve in \mathbf{R}^2). Our goal is to find another function $G : [0, 1] \rightarrow \mathbf{R}$, 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 .

- 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 $\mu \rightarrow \infty$ (*i.e.*, as we put more and more weight on minimizing curvature).
- 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 $\mu \rightarrow \infty$, 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 $\mu \rightarrow \infty$. 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, $[a \ b]^T = (A^T A)^{-1} A^T f$. In the general case, minimizing $d + \mu c$, is the same as choosing g to minimize

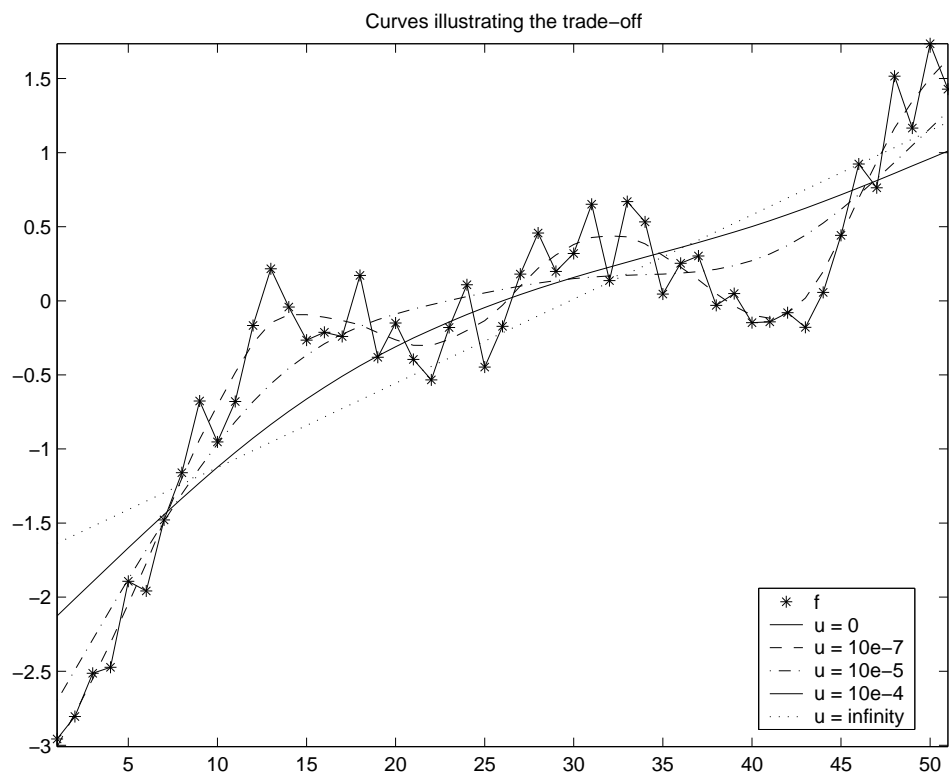
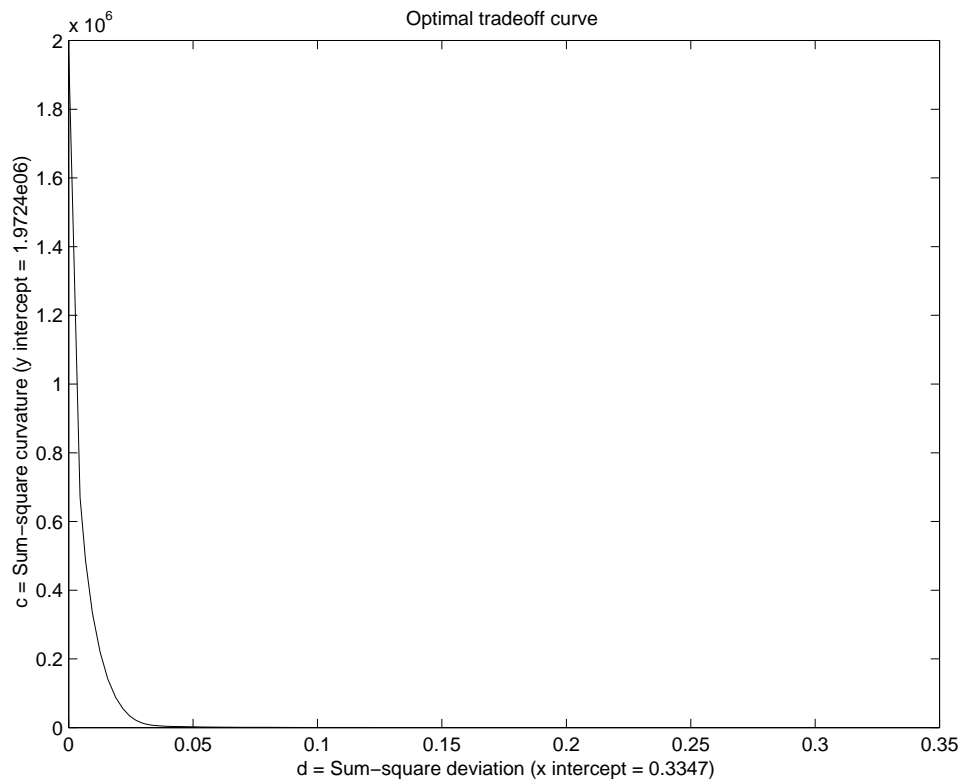
$$\left\| \frac{1}{\sqrt{n}} I g - \frac{1}{\sqrt{n}} f \right\|^2 + \mu \left\| \underbrace{\frac{n^2}{\sqrt{n-2}} \begin{bmatrix} -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \end{bmatrix}}_S g \right\|^2.$$

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

- (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;
```

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

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

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

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

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.

3. *Modifying measurements to satisfy known conservation laws.* A vector $y \in \mathbf{R}^n$ contains n measurements of some physical quantities $x \in \mathbf{R}^n$. The measurements are good, but not perfect, so we have $y \approx x$. From physical principles it is known that the quantities x must satisfy some linear equations, *i.e.*,

$$a_i^T x = b_i, \quad i = 1, \dots, m,$$

where $m < n$. As a simple example, if x_1 is the current in a circuit flowing into a node, and x_2 and x_3 are the currents flowing out of the node, then we must have $x_1 = x_2 + x_3$. More generally, the linear equations might come from various conservation laws, or balance equations (mass, heat, energy, charge ...). The vectors a_i and the constants b_i are known, and we assume that a_1, \dots, a_m are independent. Due to measurement errors, the measurement y won't satisfy the conservation laws (*i.e.*, linear equations above) exactly, although we would expect $a_i^T y \approx b_i$. An engineer proposes to adjust the measurements y by adding a correction term $c \in \mathbf{R}^n$, to get an adjusted estimate of x , given by

$$y_{\text{adj}} = y + c.$$

She proposes to find the smallest possible correction term (measured by $\|c\|$) such that the adjusted measurements y_{adj} satisfy the known conservation laws. Give an explicit formula for the correction term, in terms of y , a_i , b_i . If any matrix inverses appear in your formula, explain why the matrix to be inverted is nonsingular. Verify that the resulting adjusted measurement satisfies the conservation laws, *i.e.*, $a_i^T y_{\text{adj}} = b_i$.

Solution. The correction c must satisfy the linear equations

$$a_i^T y_{\text{adj}} = a_i^T y + a_i^T c = b_i, \quad i = 1, \dots, m,$$

i.e.,

$$a_i^T c = b_i - a_i^T y \quad i = 1, \dots, m.$$

We'll write that as $Ac = b - Ay$, where $A \in \mathbf{R}^{m \times n}$ has rows a_1, \dots, a_m . We need to find the smallest c that satisfies these linear equations, *i.e.*, the least-norm solution of $Ac = b - Ay$. That is given by

$$c = A^T(AA^T)^{-1}(b - Ay).$$

Here we take the inverse of AA^T , which is invertible since A is fat and full rank. Let's check that the conservation laws are satisfied, *i.e.*, $Ay_{\text{adj}} = b$:

$$\begin{aligned}
Ay_{\text{adj}} &= A(y + c) \\
&= Ay + AA^T(AA^T)^{-1}(b - Ay) \\
&= b
\end{aligned}$$

as required.

4. *General rank update formula* We proved a special case of the rank-1 update formula in class. For this problem, prove the more general rank update formula:

$$(A + UV)^{-1} = A^{-1} - A^{-1}U(I + VA^{-1}U)^{-1}VA^{-1}$$

and explain why such a formula might be computationally beneficial. You may assume that all matrices being inverted are invertible.

Solution. We will make use of the *push-through* identity we saw in class. Namely:

$$(I + XY)^{-1}X = X(I + YX)^{-1}$$

It is easy to prove this by multiplying out the inverses on both sides.

To prove the rank update formula, multiply by $(A + UV)$.

$$\begin{aligned} [A^{-1} - A^{-1}U(I + VA^{-1}U)^{-1}VA^{-1}] (A + UV) & \\ = I + A^{-1}UV - A^{-1}U(I + VA^{-1}U)^{-1}V(I + A^{-1}UV) & \\ = I + A^{-1}UV - A^{-1}UV(I + A^{-1}UV)^{-1}(I + A^{-1}UV) & \\ = I + A^{-1}UV - A^{-1}UV & \\ = I & \end{aligned}$$

where we made use of the push-through identity in the second step.

Such a formula can be computationally beneficial if the circumstances are right. Suppose for example that $A \in \mathbf{R}^{N \times N}$, $U \in \mathbf{R}^{N \times k}$, $V \in \mathbf{R}^{k \times N}$, and N is much larger than k . Further suppose that we have already computed A^{-1} , but we must now compute $(A + UV)^{-1}$.

By naively computing the inverse, the computational cost is on the order of N^3 . There are also costs associated with matrix multiplication and addition, but these will be much less, as they only scale with N^2 . If we use the rank update formula instead, we must still compute an inverse, $(I + VA^{-1}U)^{-1}$, but this matrix is $k \times k$ instead of $N \times N$. So the cost will be much less. Indeed, the cost will be proportional to k^3 instead of N^3 , and the bulk of the computation will likely be due to the matrix multiplications—an N^2 cost. So using the rank update formula will save us on computation by a factor of around N , a significant savings.