1. Curve-smoothing. We are given a function $F : [0, 1] \to \mathbb{R}$ (whose graph gives a curve in $\mathbb{R}^2$). Our goal is to find another function $G : [0, 1] \to \mathbb{R}$, 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 $f, g \in \mathbb{R}^n$, 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.json` 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 \( \mu \) 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.

**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,
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 \( \mu \) 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

\[
\| f - A \begin{bmatrix} a \\ b \end{bmatrix} \| \quad \text{where } A = \begin{bmatrix}
1 & 1 \\
2 & 1 \\
3 & 1 \\
\vdots & \vdots \\
n & 1
\end{bmatrix}.
\]

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

\[
\left\| \frac{1}{\sqrt{n}} Ig - \frac{1}{\sqrt{n}} f \right\|^2 + \mu \left\| \frac{n^2}{\sqrt{n - 2}} \begin{bmatrix}
-1 & 2 & -1 & 0 & \cdots & 0 \\
0 & -1 & 2 & -1 & \cdots & 0 \\
0 & 0 & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & \cdots & -1 & 2 & -1
\end{bmatrix} \begin{bmatrix} g \\ \vdots \\ g \end{bmatrix} \right\|^2.
\]
This is a multi-objective least-squares problem. The minimizing $g$ is

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

where $\tilde{A} = \left[ \frac{I}{\sqrt{n}} \sqrt{n} \mu S \right]$ and $\tilde{y} = \left[ \frac{f}{\sqrt{n}} 0 \right]$.

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

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

![Optimal tradeoff curve](image_url)
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 $\mu$. As expected, when $\mu = 0$, $g = f$ and no smoothing occurs. At the other extreme, as $\mu$ goes to infinity, we get an affine approximation of $f$. Intermediate values of $\mu$ correspond to approximations of $f$ with different degrees of smoothness.

```matlab
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;
```
\[ a_1 = a_1' \; ; \]
\[ a_2 = \text{ones}(n,1) \; ; \]
\[ A = [a_1 \; a_2] \; ; \]
\[ \text{affine\_param} = \text{inv}(A'*A)*A'*f ; \]
\[ \text{for } i = 1:n \]
\[ \text{g\_no\_curvature}(i) = \text{affine\_param}(1)*i+\text{affine\_param}(2) ; \]
\[ \text{end} \]
\[ \text{g\_no\_curvature} = \text{g\_no\_curvature}' \; ; \]
\[ \text{error\_deviation}(\text{length}(u)+2) = 1/n*\text{norm}(g\_no\_curvature-f)^2 ; \]
\[ \text{error\_curvature}(\text{length}(u)+2) = 0 ; \]
\[ \text{figure}(1) ; \]
\[ \text{plot(error\_deviation, error\_curvature)} ; \]
\[ \text{xlabel}('\text{Sum-square deviation (y intercept = 0.3347)}') ; \]
\[ \text{ylabel}('\text{Sum-square curvature (x intercept = 1.9724e06)}') ; \]
\[ \text{title}('\text{Optimal tradeoff curve}') ; \]
\[ \text{print curve\_extreme.eps} ; \]
\[ u_1 = 10e-7 ; \]
\[ A_{\text{tilde}} = [1/sqrt(n)*I;sqrt(u1)*S] ; \]
\[ y_{\text{tilde}} = [1/sqrt(n)*f;zeros(n-2,1)] ; \]
\[ g_1 = A_{\text{tilde}}\backslash y_{\text{tilde}} ; \]
\[ u_2 = 10e-5 ; \]
\[ A_{\text{tilde}} = [1/sqrt(n)*I;sqrt(u2)*S] ; \]
\[ y_{\text{tilde}} = [1/sqrt(n)*f;zeros(n-2,1)] ; \]
\[ g_2 = A_{\text{tilde}}\backslash y_{\text{tilde}} ; \]
\[ u_3 = 10e-4 ; \]
\[ A_{\text{tilde}} = [1/sqrt(n)*I;sqrt(u3)*S] ; \]
\[ y_{\text{tilde}} = [1/sqrt(n)*f;zeros(n-2,1)] ; \]
\[ g_3 = A_{\text{tilde}}\backslash y_{\text{tilde}} ; \]
\[ \text{figure}(3) ; \]
\[ \text{plot}(f,'*') ; \]
\[ \text{hold} ; \]
\[ \text{plot(}g_{\text{no\_deviation}}\text{)} ; \]
\[ \text{plot(}g_1,'--') ; \]
\[ \text{plot(}g_2,'-.') ; \]
\[ \text{plot(}g_3,'-') ; \]
\[ \text{plot(}g_{\text{no\_curvature}},':') ; \]
\[ \text{axis tight} ; \]
\[ \text{legend('f','u = 0','u = 10e-7', 'u = 10e-5', 'u = 10e-4','u = infinity',0)} ; \]
\[ \text{title('Curves illustrating the trade-off')} ; \]
\[ \text{print curve\_tradeoff.eps} ; \]

\text{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 \( g \in \text{null}(S) \), then \( c = 0 \). For full credit, however, solutions had to show how to choose the vector in \( \text{null}(S) \) 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 \( \mu \to \infty \)).

2. Optimal control of unit mass. In this problem you will use the language you prefer 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 \( \mu \) has to cover a very large range, so it usually works better in practice to give it a logarithmic spacing, using, e.g., \texttt{logspace} in matlab (and similar functions in Julia and Python, etc.). 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, and the final plots produced.
Solution.

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

\[
\int_0^{10} f(t)^2 dt = \int_0^1 x_1^2 dt + \int_1^2 x_2^2 dt + \cdots + \int_9^{10} x_{10}^2 dt
= x_1^2 + x_2^2 + \cdots + x_{10}^2
= \|x\|^2.
\]

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

\[
\begin{bmatrix}
  p(10) \\
  \dot{p}(10) \\
  p(5)
\end{bmatrix} =
\begin{bmatrix}
  1 \\
  0 \\
  0
\end{bmatrix} =
\begin{bmatrix}
  9.5 & 8.5 & 7.5 & 6.5 & 5.5 & 4.5 & 3.5 & 2.5 & 1.5 & 0.5 \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
  4.5 & 3.5 & 2.5 & 1.5 & 0.5 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{10}
\end{bmatrix}.
\]

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:

```matlab
>> 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];
>> y=[1;0;0];
>> x=pinv(A)*y
x =
-0.0455
-0.0076
 0.0303
 0.0682
 0.1061
 0.0939
 0.0318
```

7
-0.0303  
-0.0924  
-0.1545

>> A*x
ans =
1.0000  
0.0000  
-0.0000

>> norm(x)
an =
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)')
>> 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
    0.5000   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
```
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:

$$\begin{bmatrix} p(10) \\ \dot{p}(10) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 9.5 & 8.5 & 7.5 & 6.5 & 5.5 & 4.5 & 3.5 & 2.5 & 1.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} x + \begin{bmatrix} 10 \\ 1 \end{bmatrix}.$$  \hspace{1cm} (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 $x_\mu$ of the expression $J_1 + \mu J_2 = \|Ax - y\|^2 + \mu\|x\|^2$ for different values of $\mu$. As it was proven in the lecture notes,

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

We use matlab to calculate $x_\mu$ and the resulting pairs $(J_1, J_2)$ and plot the tradeoff curve. A sample matlab script is given below:

```matlab
clear; clf;
N = 50;
a1 = linspace(9.5,0.5,10); a2 = ones(1,10); A = [a1; a2];
y = [-10;-1];
```
I = eye(10);
mu = logspace(-2, 4, N);
for i=1:N;
    x_mu = inv(A'*A + mu(i)*I'*I)*A'*y;
    J_1(i) = (norm(A*x_mu - y))^2;
    J_2(i) = (norm(x_mu))^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 \( x_\mu \) 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 \( x_\mu \) 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 \( \| \begin{bmatrix} 10 \\ 1 \end{bmatrix} \|^2 = 101 \).

3. Minimum fuel and minimum peak input solutions. Suppose \( A \in \mathbb{R}^{m \times n} \) is fat and full rank, so there are many \( x \)'s that satisfy \( Ax = y \). In lecture we encountered the least-norm solution given by \( x_{ln} = A^T(AA^T)^{-1}y \). This solution has the minimum (Euclidean) norm among all solutions of \( Ax = y \). In many applications we want to minimize another norm of \( x \) (i.e., measure of size of \( x \)) subject to \( Ax = y \). Two common examples are the 1-norm and \( \infty \)-norm, which are defined as

\[
\|x\|_1 = \sum_{i=1}^{n} |x_i|, \quad \|x\|_\infty = \max_{i=1,...,n} |x_i|.
\]
The 1-norm, for example, is often a good measure of fuel use; the $\infty$-norm is the peak of the vector or signal $x$. There is no simple formula for the least 1-norm or $\infty$-norm solution of $Ax = y$, like there is for the least (Euclidean) norm solution. They can be computed very easily, however. (That’s one of the topics of EE364.) The analysis is a bit trickier as well, since we can’t just differentiate to verify that we have the minimizer. For example, how would you know that a solution of $Ax = y$ has minimum 1-norm? In this problem you will explore this idea. First verify the following inequality, which is like the Cauchy-Schwarz inequality (but even easier to prove): for any $v, w \in \mathbb{R}^p$, the following inequality holds: $w^T v \leq \|v\|_\infty \|w\|_1$.

From this inequality it follows that whenever $v \neq 0$,

$$\|w\|_1 \geq \frac{w^T v}{\|v\|_\infty}.$$ 

Now let $z$ be any solution of $Az = y$, and let $\lambda \in \mathbb{R}^m$ be such that $A^T \lambda \neq 0$. Explain why we must have

$$\|z\|_1 \geq \frac{\lambda^T y}{\|A^T \lambda\|_\infty}.$$ 

Thus, any solution of $Az = y$ must have 1-norm at least as big as the righthand side expression. Therefore if you can find $x_{mf} \in \mathbb{R}^n$ (mf stands for minimum fuel) and $\lambda \in \mathbb{R}^m$ such that $Ax_{mf} = y$ and

$$\|x_{mf}\|_1 = \frac{\lambda^T y}{\|A^T \lambda\|_\infty},$$

then $x_{mf}$ is a minimum fuel solution. (Explain why.) Methods for computing $x_{mf}$ and the mysterious vector $\lambda$ are described in EE364. In the rest of this problem, you’ll use these ideas to verify a statement made during lecture. Now consider the problem from the lecture notes of a unit mass acted on by forces $x_1, \ldots, x_{10}$ for one second each. The mass starts at position $p(0) = 0$ with zero velocity and is required to satisfy $p(10) = 1, \dot{p}(10) = 0$. There are, of course, many force vectors that satisfy these requirements. In the lecture notes, you can see a plot of the least (Euclidean) norm force profile. In class I stated that the minimum fuel solution is given by $x_{mf} = (1/9, 0, \ldots, 0, -1/9)$, i.e., an accelerating force at the beginning, 8 seconds of coasting, and a (braking) force at the end to decelerate the mass to zero velocity at $t = 10$. Prove this. Hint: try $\lambda = (1, -5)$. Verify that the 1-norm of $x_{mf}$ is less than the 1-norm of $x_{ln}$, the (Euclidean) least-norm solution. Feel free to write and execute codes in your preferred language for your purpose. There are several convenient ways to find the 1- and $\infty$-norm of a vector $z$, e.g., $\text{norm}(z, 1)$ and $\text{norm}(z, \infty)$ or $\text{sum}(\text{abs}(z))$ and $\text{max}(\text{abs}(z))$. One last question, for fun: what do you think is the minimum peak force vector $x_{mp}$? How would you verify that a vector $x_{mp}$ (mp for minimum peak) is a minimum $\infty$-norm solution of $Ax = y$? This input, by the way, is very widely used in practice. It is (basically) the input used in a disk drive to move the head from one track to another, while respecting a maximum possible current in the disk drive motor coil. Hints:

- The input is called bang-bang.
- Some people drive this way.
Solution. First, we will prove the inequality $|w^Tv| \leq \|v\|_{\infty} \|w\|_1$ for any $v, w \in \mathbb{R}^p$:

$$
|w^Tv| = \left| \sum_i w_i v_i \right|
\leq \sum_i |w_i v_i|
= \sum_i |w_i||v_i|
\leq \max_i |v_i| \sum_i |w_i|
= \|v\|_{\infty} \|w\|_1
$$

Or $\|w\|_1 \geq \frac{|w^Tv|}{\|v\|_{\infty}}$ if $v \neq 0$. Under what conditions will the equality hold? Note that the equality in the first line holds if $w_i v_i \geq 0$ for $i = 1, \ldots, n$, and the one on the third line holds when $|v_i| = \max_i |v_i| = v_m$, for $i = 1, \ldots, n$. For both equalities to hold, we get $v_i = v_m \text{sgn}(w)$. Now we will use the above inequality to derive a lower bound on $\|z\|_1$, for all $z$ satisfying $Az = y$.

Take transposes of both sides to get $z^T A^T = y^T$, then multiply both sides on the right by a nonzero (but otherwise arbitrary) vector $\lambda$. This yields $z^T A^T \lambda = y^T \lambda$. Now let $w = z$, and $v = A^T \lambda$ in the above inequality to get $|z^T (A^T \lambda)| \leq \|z\|_1 \|A^T \lambda\|_{\infty}$, and therefore

$$
\|z\|_1 \geq \frac{|\lambda^T y|}{\|A^T \lambda\|_{\infty}}
$$

Thus, any solution of $Az = y$ must have $1$-norm at least as large as the righthand side expression for any $\lambda$ we pick. Hence, if there exist a $z$ that satisfies the equality for some value of $\lambda$, then $z$ has the smallest possible $1$-norm, and is the minimum fuel solution. (In fact, this particular value of $\lambda$ maximizes the righthand side, giving the largest possible lower bound to the value $\|z\|_1$) Now we can prove that $x_{mf} = \begin{bmatrix} 1/9 & 0 & \ldots & 0 & -1/9 \end{bmatrix}^T$ is the minimum fuel solution by showing that it achieves the lower bound for the $\lambda$ given in the hint. Using MATLAB:

```
>> A = [linspace(9.5,0.5,10); ones(1,10)];
>> lambda = [1; -5];
>> y=[1;0];
>> x_mf = 1/9*[1; zeros(8,1);-1];
>> norm(x_mf,1)
ans =
0.2222
>> abs(lambda'*[1;0])/norm(A'*lambda,inf)
ans =
0.2222
```

Also, to compare with the (Euclidean) least-norm solution.

```
>> x_ln = pinv(A)*y;
>> norm(x_ln)
ans =
0.6348
```
In order to verify that a given $x_{mp}$ is the minimum peak solution, we can proceed similarly, starting from the same inequality we proved above, and just switching all the 1-norms and $\infty$-norms to get

$$\|z\|_{\infty} \geq \frac{|\lambda^T y|}{\|A^T \lambda\|_1}$$

And then show that $x_{mp}$ and a certain value of $\lambda$ (that is found by maximizing the right-hand side expression over $\lambda$) in fact satisfy the equality. For this problem, the minimum peak force $x_{mp}$ turns out to be constant at $1/25$ for 5 seconds, and then a constant $-1/25$ for the remaining 5 seconds. This is called a bang-bang input because it consists of a maximum acceleration for 5 seconds, followed by maximum deceleration for 5 seconds. So don’t make fun of people from the Boolean or ‘full accelerator/ full brake’ driving school; they’re just minimizing the peak force on the vehicle.

4. **Iteratively reweighted least squares for 1-norm approximation.** In an ordinary least squares problem, we are given $A \in \mathbb{R}^{m \times n}$ (skinny and full rank) and $y \in \mathbb{R}^m$, and we choose $x \in \mathbb{R}^n$ in order to minimize

$$\|Ax - y\|_2^2 = \sum_{i=1}^{m} (\tilde{a}_i^T x - y_i)^2.$$ 

Note that the penalty that we assign to a measurement error does not depend on the sensor from which the measurement was taken. However, this is not always the right thing to do: if we believe that one sensor is more accurate than another, we might want to assign a larger penalty to an error in the measurement from the more accurate sensor. We can account for differences in the accuracies of our sensors by assigning sensor $i$ a weight $w_i > 0$, and then minimizing

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2.$$ 

By giving larger weights to more accurate sensors, we can account for differences in the precision of our sensors.

a) **Weighted least squares.** Explain how to choose $x$ in order to minimize

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2,$$

where the weights $w_1, \ldots, w_n > 0$ are given.

b) **Iteratively reweighted least squares for $\ell_1$-norm approximation.** Consider a cost function of the form

$$\sum_{i=1}^{m} w_i (x)(\tilde{a}_i^T x - y_i)^2.$$ 

(3)

One heuristic for minimizing a cost function of the form given in (3) is iteratively reweighted least squares, which works as follows. First, we choose an initial point
Then, we generate a sequence of points \( x^{(1)}, x^{(2)}, \ldots \in \mathbb{R}^n \) by choosing \( x^{(k+1)} \) in order to minimize

\[
\sum_{i=1}^{m} w_i(x^{(k)})(\tilde{a}_i^T x^{(k+1)} - y_i)^2.
\]

Each step of this algorithm involves updating our weights, and solving a weighted least squares problem. Suppose we want to use this method to solve minimize the \( \ell_1 \)-norm approximation error, which is defined to be

\[
\| Ax - y \|_1 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i|,
\]

where the matrix \( A \in \mathbb{R}^{m \times n} \) and the vector \( y \in \mathbb{R}^m \) are given. How should we choose the weights \( w_i(x) \) to make the cost function in (3) equal to the \( \ell_1 \)-norm approximation error?

c) Numerical example. The file \texttt{l1_irwls_data.json} contains data \((t_1, y_1), \ldots, (t_m, y_m)\). We want to fit an affine model to this data:

\[
y_i = x_1 + x_2 t_i, \quad i = 1, \ldots, m.
\]

Choose \( x^{(0)} \) to be the vector of least-squares parameter estimates: that is, choose \( x^{(0)} \) in order to minimize

\[
\sum_{i=1}^{m} ((x_1^{(0)} + x_2^{(0)} t_i) - y_i)^2.
\]

Generate \( x^{(1)}, x^{(2)}, \ldots \) using iteratively reweighted least squares for \( \ell_1 \)-norm approximation. You can stop generating iterates when \( \|x^{(k+1)} - x^{(k)}\| < 10^{-6} \). Report your values of \( x^{(0)} \) and the final \( x^{(k)} \) in your sequence of points. Draw a scatterplot of the data points \((t_i, y_i)\). Add the fitted lines corresponding to \( x^{(0)} \) and the final \( x^{(k)} \) to your scatterplot. What do you observe?

Remark. Suppose we fit the least-squares line to some data. Then, a point that is very far from the least-squares line may be an outlier: that is, a point that does not seem to follow the same model as the rest of the data. Because such points may not follow the same model as the rest of data, it may make sense to give such points less weight. This idea is the intuition behind iteratively reweighted least squares for \( \ell_1 \)-norm approximation.

Solution.

a) We can express the objective function as

\[
\sum_{i=1}^{m} w_i(\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} (\sqrt{w_i} \tilde{a}_i^T x - \sqrt{w_i} y_i)^2
\]

\[
= \left\| \begin{bmatrix} \sqrt{w_1} \tilde{a}_1^T & \cdots & \sqrt{w_m} \tilde{a}_m^T \end{bmatrix} x - \begin{bmatrix} \sqrt{w_1} y_1 \\ \vdots \\ \sqrt{w_m} y_m \end{bmatrix} \right\|^2
\]

\[
= \left\| \frac{1}{2} W^\frac{1}{2} A x - \frac{1}{2} W^\frac{1}{2} y \right\|^2,
\]

where

\[
W = \text{Diag}(w_1, \ldots, w_m)
\]
where $W^{1/2} = \text{diag}(\sqrt{w_1}, \ldots, \sqrt{w_m})$. Note that $(W^{1/2})^2 = W = \text{diag}(w_1, \ldots, w_m)$. Thus, minimizing this objective function is a least-squares problem; the solution is

$$x = ((W^{1/2}A)^T(W^{1/2}A))^{-1}(W^{1/2}A)^T(W^{1/2}y) = (A^TWA)^{-1}A^Ty.$$

b) If we choose $w_i(x) = 1/|\tilde{a}_i^T x - y_i|$, then we have that

$$\sum_{i=1}^{m} w_i(x)(\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} \frac{1}{|\tilde{a}_i^T x - y_i|} (\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i| = \|Ax - y\|_1.$$

Note that $w_i(x)$ is undefined if $w_i(x) = |\tilde{a}_i^T x - y_i| = 0$. In this case, we can just take $w_i(x)$ to be some large value $w_{\text{max}}$. However, note that in practice it is extremely unlikely that one of the residuals will be exactly equal to zero.

c) For fitting a line in two-dimensions, we have that $\tilde{a}_i = (1, t_i)$. The following code applies iteratively reweighted least squares for $\ell_1$-norm approximation to the data defined in $l1_irwls_data.m$.

```matlab
clear all; close all; clc
l1_irwls_data;

A = [ones(m,1) , t];

kmax = 100;
x = nan(2,kmax);
x(:,1) = A\y;
for k = 1:kmax
    W = diag(1./abs(A*x(:,k)-y));
x(:,k+1) = (A'*W*A) \ (A'*W*y);
    if norm(x(:,k) - x(:,k+1)) < 1e-6
        break;
    end
end
x0 = x(:,1)
k
xk = x(:,k+1)

figure();
plot(t , y , 'ko');
hold on;
    plot([0 1] , x(1,1) + x(2,1) * [0 1] , 'r');
    plot([0 1] , x(1,k) + x(2,k) * [0 1] , 'b');
hold off;
xlabel('t');
ylabel('y');
legend('data','l2 fit','l1 fit');
print -depsc l1-l2-fits.eps
```
The initial and final parameter estimates are
\[ x^{(0)} = \begin{bmatrix} 3.5084 \\ 5.5540 \end{bmatrix} \quad \text{and} \quad x^{(20)} = \begin{bmatrix} 2.0839 \\ 7.8694 \end{bmatrix}. \]

The $\ell_2$- and $\ell_1$-norm approximations of the data are given in the plot below. We see that the $\ell_2$-norm approximation is much more sensitive to the outliers than the $\ell_1$-norm approximation.