1. Single sensor failure detection and identification. We have \( y = Ax \), where \( A \in \mathbb{R}^{m \times n} \) is known, and \( x \in \mathbb{R}^n \) is to be found. Unfortunately, up to one sensor may have failed (but you don’t know which one has failed, or even whether any has failed). You are given \( \tilde{y} \) and not \( y \), where \( \tilde{y} \) is the same as \( y \) in all entries except, possibly, one (say, the \( k \)th entry). If all sensors are operating correctly, we have \( y = \tilde{y} \). If the \( k \)th sensor fails, we have \( \tilde{y}_i = y_i \) for all \( i \neq k \).

The file \texttt{one_bad_sensor.m}, available on the course web site, defines \( A \) and \( \tilde{y} \) (as \( A \) and \( y\tilde{m}d*e\)). Determine which sensor has failed (or if no sensors have failed). You must explain your method, and submit your code.

For this exercise, you can use the matlab code \( \text{rank}([F \ g]) == \text{rank}(F) \) to check if \( g \in \text{range}(F) \). (We will see later a much better way to check if \( g \in \text{range}(F) \).)

Solution. Let \( y^{(i)} \) be the measurement vector \( y \) with the \( i \)th entry removed. Likewise, let \( A^{(i)} \) be the measurement matrix with the \( i \)th row of \( A \) removed. This corresponds to the system without the \( i \)th sensor.

If the \( i \)th sensor is faulty, we will almost surely have \( y \notin \text{range}(A) \) (unless the sensor failure happens to give the same response \( y_i \) as that predicted by \( A \), which is highly unlikely). However, once we remove its faulty measurement, we will certainly have \( y^{(i)} \in \text{range}(A^{(i)}) \).

To test if a vector \( z \) is in \( \text{range}(C) \), we can use matlab and compare \( \text{rank}([C \ z]) = \text{rank}(C) \). If they are equal, \( z \in \text{range}(C) \). Otherwise \( \text{rank}([C \ z]) = \text{rank}(C) + 1 \). To find a faulty sensor, we remove one row of \( A \) at a time, and use the above test.

The following matlab code solves the problem

```matlab
for k=1:m
    withoutk=[1:k-1 k+1:m];
    Atent = A(withoutk,:);
    ytent = ytilde(withoutk);
    if rank([ Atent ytent ]) == rank(Atent)
        k
    end
end
```

The 11th sensor is faulty.
2. Linearizing range measurements. Consider a single (scalar) measurement $y$ of the distance or range of $x \in \mathbb{R}^n$ to a fixed point or beacon at $a$, i.e., $y = \|x - a\|$.

a) Show that the linearized model near $x_0$ can be expressed as $\delta y = k^T \delta x$, where $k$ is the unit vector (i.e., with length one) pointing from $a$ to $x_0$. Derive this analytically, and also draw a picture (for $n = 2$) to demonstrate it.

b) Consider the error $e$ of the linearized approximation, i.e.,

$$e = \|x_0 + \delta x - a\| - \|x_0 - a\| - k^T \delta x.$$ 

The relative error of the approximation is given by $\eta = e/\|x_0 - a\|$. We know, of course, that the absolute value of the relative error is very small provided $\delta x$ is small. In many specific applications, it is possible and useful to make a stronger statement, for example, to derive a bound on how large the error can be. You will do that here. In fact you will prove that

$$0 \leq \eta \leq \frac{\alpha^2}{2},$$

where $\alpha = \|\delta x\|/\|x_0 - a\|$ is the relative size of $\delta x$. For example, for a relative displacement of $\alpha = 1\%$, we have $\eta \leq 0.00005$, i.e., the linearized model is accurate to about $0.005\%$. To prove this bound you can proceed as follows:

- Show that $\eta = -1 + \sqrt{1 + \alpha^2 + 2\beta - \beta}$ where $\beta = k^T \delta x/\|x_0 - a\|$.
- Verify that $|\beta| \leq \alpha$.
- Consider the function $g(\beta) = -1 + \sqrt{1 + \alpha^2 + 2\beta - \beta}$ with $|\beta| \leq \alpha$. By maximizing and minimizing $g$ over the interval $-\alpha \leq \beta \leq \alpha$ show that

$$0 \leq \eta \leq \frac{\alpha^2}{2}.$$

Solution.

a) For the linearized model we have

$$\delta y = \left(\frac{\partial y}{\partial x}\right) \delta x$$

so all we have to do is to compute the matrix $\partial y/\partial x$. Since $y = \|x - a\|$ we have $y^2 = (x - a)^T (x - a)$ and differentiating both sides with respect to $x$ gives

$$2 \frac{\partial y}{\partial x} y = 2(x - a)^T$$

and therefore

$$\frac{\partial y}{\partial x} = \frac{(x - a)^T}{y} = \frac{(x - a)^T}{\|x - a\|},$$

so $\delta y = k^T \delta x$ with $k = (x - a)/\|x - a\|$. Clearly, $k$ points from $a$ to $x$ and is of length one since

$$k^T k = \frac{(x - a)^T (x - a)}{\|x - a\|^2} = 1.$$
\textbf{b)  • First we show that} \quad \eta = -1 + \sqrt{1 + \alpha^2 + 2\beta} - \beta \quad \text{where} \quad \beta = k^T \delta x / \|x_0 - a\|. \text{ Note that}

\[ e = \|x_0 + \delta x - a\| - \|x_0 - a\| - k^T \delta x \]

\[ = \|x_0 - a\| \left( \left\| \frac{x_0 - a}{\|x_0 - a\|} + \frac{\delta x}{\|x_0 - a\|} \right\| - 1 - \frac{k^T \delta x}{\|x_0 - a\|} \right), \]

\text{and after dividing both sides by} \quad \|x_0 - a\| \text{ and using} \quad k = (x_0 - a) / \|x_0 - a\|, \quad \beta = k^T \delta x / \|x_0 - a\| \quad \text{and} \quad \eta = e / \|x_0 - a\| \quad \text{we get}

\[ \eta = \|k + \frac{\delta x}{\|x_0 - a\|} - 1 - \beta. \]

\text{But}

\[ \|k + \frac{\delta x}{\|x_0 - a\|}\| = \sqrt{\left( k + \frac{\delta x}{\|x_0 - a\|} \right)^T \left( k + \frac{\delta x}{\|x_0 - a\|} \right)} \]

\[ = \sqrt{\|k\|^2 + 2 \frac{k^T \delta x}{\|x_0 - a\|} + \frac{\|\delta x\|^2}{\|x_0 - a\|^2}}. \]

\text{Since} \quad \|k\| = 1 \text{ and by substituting the values for} \quad \alpha \text{ and} \quad \beta \quad \text{we have}

\[ \left\| k + \frac{\delta x}{\|x_0 - a\|} \right\| = \sqrt{1 + 2\beta + \alpha^2}. \]

\text{Therefore (??) can be written as}

\[ \eta = \sqrt{1 + 2\beta + \alpha^2} - 1 - \beta. \]

\text{• It is easy to see that} \quad |\beta| \leq \alpha. \text{ Simply we can use the Cauchy-Schwarz inequality for the vectors} \quad k \text{ and} \quad \delta x / \|x_0 - a\|, \text{ i.e.,}

\[ \left| \frac{k^T \delta x}{\|x_0 - a\|} \right| \leq \|k\| \frac{\|\delta x\|}{\|x_0 - a\|} \]

\text{and since} \quad \|k\| = 1 \text{ we immediately get} \quad |\beta| \leq \alpha.
At this point, all we need to do to derive a bound on how large the error can be is to maximize and minimize the function \( g(\beta) = \sqrt{1 + 2\beta + \alpha^2} - 1 - \beta \) over the interval \( |\beta| \leq \alpha \) or \( -\alpha \leq \beta \leq \alpha \). The maximum or minimum of a smooth function \( g(\beta) \) over a given interval \(( -\alpha \leq \beta \leq \alpha )\) can only occur at the endpoints of the interval \(( \beta = \pm \alpha )\) or at the extremums (points \( \beta \) with \( g'(\beta) = 0 \)). For \( g(\beta) \) we have:

- **Value at endpoint \( \beta = \alpha \).**
  \[
g(\alpha) = \sqrt{1 + 2\alpha + \alpha^2} - 1 - \alpha
  = \sqrt{(1 + \alpha)^2} - 1 - \alpha = 1 + \alpha - 1 - \alpha = 0.
  \]

- **Value at endpoint \( \beta = -\alpha \).**
  \[
g(-\alpha) = \sqrt{1 - 2\alpha + \alpha^2} - 1 - \alpha
  = \sqrt{(1 - \alpha)^2} - 1 + \alpha
  = |1 - \alpha| - (1 - \alpha)
  = \begin{cases} 0; & 0 \leq \alpha \leq 1 \\ 2(\alpha - 1); & \alpha > 1. \end{cases}
  \]

Therefore \( g(-\alpha) \geq 0 \) for all \( \alpha \) because \( 2(\alpha - 1) > 0 \) for \( \alpha > 1 \).

- **Extremum value.**
  \[
g'(\beta) = \frac{1}{\sqrt{1 + 2\beta + \alpha^2}} - 1.
  \]

Setting \( g'(\beta) = 0 \) we get \( \sqrt{1 + 2\beta + \alpha^2} = 1 \) or \( 1 + 2\beta + \alpha^2 = 1 \) and therefore \( \beta_{\text{ex.}} = -\alpha^2/2 \). The function value at the extremum \( \beta_{\text{ex.}} = -\alpha^2/2 \) is

\[
g(\beta_{\text{ex.}}) = \sqrt{1 - \alpha^2 + \alpha^2} - 1 + \frac{\alpha^2}{2}
  = \frac{\alpha^2}{2}.
  \]

Clearly, \( g(\beta) \geq 0 \) for all \( \beta \) satisfying \( |\beta| \leq \alpha \) because the value of \( g(\beta) \) at the endpoints \( \beta = \pm \alpha \) at the extremum \( \beta = \alpha^2/2 \) are all non-negative. Thus we have achieved the lower bound on the relative error \( \eta \), i.e., we have shown that \( \eta \geq 0 \). For the upper bound we need to be a bit more careful. The upper bound we get is either \( g(\alpha) \), \( g(-\alpha) \) or \( g(\beta_{\text{ex.}}) \). First note that \( g(\alpha) = 0 \) is always less than or equal to \( g(\beta_{\text{ex.}}) = \alpha^2/2 \geq 0 \) so the choice of \( g(\alpha) \) is immediately ruled out as the maximum of \( g \). Now consider \( g(-\alpha) \) and \( g(\beta_{\text{ex.}}) \). For \( 0 \leq \alpha \leq 1 \) we obviously have \( g(\beta_{\text{ex.}}) \geq g(-\alpha) = 0 \). For \( \alpha > 1 \) we also have \( g(\beta_{\text{ex.}}) \geq g(-\alpha) = 2(\alpha - 1) \) because \( \alpha^2/2 \geq 2(\alpha - 1) \) is equivalent to \( \alpha^2 - 4\alpha + 4 \geq 0 \) which is true since \( \alpha^2 - 4\alpha + 4 = (\alpha - 2)^2 \) is a complete square. Thus, we achieve an upper bound on \( g(\beta) \) for all \( \beta \) satisfying \( |\beta| \leq \alpha \) as \( g(\beta) \leq \alpha^2/2 \). Therefore we have shown that
η ≤ α^2/2 and we are done.
(Note: when β_{ex} falls outside the interval β ≤ |α|, it is possible to achieve a tighter upper bound for \( g \). In this case, the maximum of \( g \) over \( β ≤ |α| \) is obtained at the endpoint \( β = -α/2 \). The extremum \( β_{ex} = -α^2/2 \) falls outside \( β ≤ |α| \) when \( α^2/2 > α \) or \( α > 2 \). Therefore, a tighter upper bound on η for \( α > 2 \) becomes \( η ≤ g(-α) = 2(α - 1) \).)

3. **Reverse engineering a smoothing filter.** A smoothing filter takes an input vector \( u ∈ \mathbb{R}^n \) and produces an output vector \( y ∈ \mathbb{R}^n \). (We will assume that \( n ≥ 3 \).) The output \( y \) is obtained as the minimizer of the objective

\[
J = J^{\text{track}} + \lambda J^{\text{norm}} + \mu J^{\text{cont}} + κ J^{\text{smooth}},
\]

where \( λ, μ, \) and \( κ \) are positive constants (weights), and

\[
J^{\text{track}} = \sum_{i=1}^{n} (u_i - y_i)^2, \quad J^{\text{norm}} = \sum_{i=1}^{n} y_i^2
\]

are the tracking error and norm-squared of \( y \), respectively, and

\[
J^{\text{cont}} = \sum_{i=2}^{n} (y_i - y_{i-1})^2, \quad J^{\text{smooth}} = \sum_{i=2}^{n-1} (y_{i+1} - 2y_i + y_{i-1})^2
\]

are measures of the continuity and smoothness of \( y \), respectively.

Here is the problem: You have access to one input-output pair, i.e., an input \( u \), and the associated output \( y \). Your goal is to find the weights \( λ, μ, \) and \( κ \). In other words, you will reverse engineer the smoothing filter, working from an input-output pair.

a) Explain how to find \( λ, μ, \) and \( κ \). (You do not need to worry about ensuring that these are positive; you can assume this will occur automatically.)

b) Carry out your method on the data found in `rev_eng_smooth_data.m`. Give the values of the weights.

**Solution.** We first define matrices \( D_1 ∈ \mathbb{R}^{(n-1)×n} \) and \( D_2 ∈ \mathbb{R}^{(n-2)×n} \) as

\[
D_1 = \begin{bmatrix}
-1 & 1 & 0 & \ldots & 0 \\
0 & -1 & 1 & \ldots & 0 \\
0 & \vdots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & -1 & 1
\end{bmatrix}, \quad D_2 = \begin{bmatrix}
1 & -2 & 1 & 0 & \ldots & 0 \\
0 & 1 & -2 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & 1 & -2 & 1
\end{bmatrix}.
\]

Multiplying a vector by these matrices gives the first and second differences:

\[
(D_1 y)_i = y_{i+1} - y_i, \quad (D_2 y)_i = y_{i+2} - 2y_{i+1} + y_i.
\]

Now we can write the objective components in the compact form

\[
J^{\text{track}} = \|y - u\|^2, \quad J^{\text{norm}} = \|y\|^2, \quad J^{\text{cont}} = \|D_1 y\|^2, \quad J^{\text{smooth}} = \|D_2 y\|^2.
\]
Since \( y \) minimizes \( J \), the gradient of \( J \) with respect to \( y \) must be zero, \textit{i.e.},
\[
\nabla_y J = 2(y - u) + 2\lambda y + 2\mu D_1^T D_1 y + 2\kappa D_2^T D_2 y = 0.
\]

We are interested in finding \( \lambda, \mu, \) and \( \kappa \), so we re-write these equations as a set of linear equations involving \( \lambda, \mu, \) and \( \kappa \):
\[
\begin{bmatrix}
    y & D_1^T D_1 y & D_2^T D_2 y
\end{bmatrix}
\begin{bmatrix}
    \lambda \\
    \mu \\
    \kappa
\end{bmatrix}
= u - y.
\]

This is a set of \( n \) equations in the three unknowns \( \lambda, \mu, \) and \( \kappa \). For \( n > 3 \) this is a set of overdetermined linear equations; but we know that the equations are solvable.

We didn’t require you to do so, but we can say when the matrix above is full rank (in which case, we can recover \( \lambda, \mu, \) and \( \kappa \) exactly). It is full rank precisely when \( y \) does not satisfy a two-term recursion, \textit{i.e.}, \( y_{t+1} = \alpha y_t + \beta y_{t-1} \) for some \( \alpha, \beta \in \mathbb{R} \). It was OK with us for you to simply assume the matrix is full rank, or to check that for the given \( y \), it is full rank.

We can calculate \( \lambda, \mu, \kappa \) as
\[
\begin{bmatrix}
    \lambda \\
    \mu \\
    \kappa
\end{bmatrix}
= \left[ y \ D_1^T D_1 y \ D_2^T D_2 y \right] ^\dagger (u - y).
\]

(It’s good practice to check that the computed \( (\lambda, \mu, \kappa) \) do indeed satisfy the equations above, as they must.)

The following matlab script implements the solution.

```matlab
rev_eng_smooth_data;

% construct the first difference operator, D1
D1 = zeros(n-1,n);
for i = 2 : n
    D1(i-1,i-1) = -1;
    D1(i-1,i) = 1;
end

% construct the second difference operator, D2
D2 = zeros(n-2,n);
for i = 2:n-1
    D2(i-1,i-1) = 1;
    D2(i-1,i) = -2;
    D2(i-1,i+1) = 1;
end

% solve for the weights (i.e., lambda, mu, kappa)
weights = ([y D1'*D1*y D2'*D2*y]')\( (u-y) ;
%
% let’s check that the equations are satisfied exactly (up to numerical errors)
```
norm([y D1'*D1*y D2'*D2*y]*weights-(u-y))

lambda = weights(1)
mu = weights(2)
kappa = weights(3)

We find that $\lambda = 0.1$, $\mu = 2$ and $\kappa = 10$. (The script verifies that the overdetermined equations are indeed satisfied.)

4. **Coin collector robot.** Consider a robot with unit mass which can move in a frictionless two dimensional plane. The robot has a constant unit speed in the $y$ direction (towards north), and it is designed such that we can only apply force in the $x$ direction. We will apply a force at time $t$ given by $f_j$ for $2j-2 \leq t < 2j$ where $j = 1, \ldots, n$, so that the applied force is constant over time intervals of length 2. The robot is at the origin at time $t = 0$ with zero velocity in the $x$ direction.

There are $2n$ coins in the plane and the goal is to design a sequence of input forces for the robot to collect the maximum possible number of coins. The robot is designed such that it can collect the $i$th coin only if it exactly passes through the location of the coin $(x_i, y_i)$. In this problem, we assume that $y_i = i$.

a) Find the coordinates of the robot at time $t$, where $t$ is a positive integer. Your answer should be a function of $t$ and the vector of input forces $f \in \mathbb{R}^n$.

b) Given a sequence of $k$ coins $(x_1, y_1), \ldots, (x_{2n}, y_{2n})$, describe a method to find whether the robot can collect them.

c) For the data provided in `robot_coin_collector.m`, show that the robot cannot collect all the coins.

d) Suppose that there is an arrangement of the coins such that it is not possible for the robot to collect all the coins. Suggest a way to check if the robot can collect all but one of the coins.

e) Run your method on data in `robot_coin_collector.m` and report which coin cannot be collected. Report the input that results in collecting $2n-1$ coins. Plot the location of the coins and the location of the robot at integer times.

**Solution.**

a) The second coordinate at time $t$ is simply equal to $t$.

Consider $A \in \mathbb{R}^{2n \times n}$ such that

$$A_{ij} = \begin{cases} 1 & j = \lfloor \frac{i+1}{2} \rfloor \\ 0 & \text{Otherwise.} \end{cases}$$

Then we will have $Af = [f_1, f_1, f_2, \ldots, f_n]$. Similar to the mass/force example, the first coordinate at time $t$ will be equal to $b_t^T Af$ where

$$b_t = [t - \frac{1}{2}, \ldots, \frac{1}{2}, 0, \ldots, 0]^T.$$
b) According to part a, the only possible time to collect the $i$th coin is at time $t = y_i = i$. Define $l_i$ to be the first coordinate of the location of the robot at time $t = i$. From part a, we see that

$$l_i = b_i^T Af.$$ 

Let $B \in \mathbb{R}^{n\times n}$ be a matrix whose $i$th column is $b_i$ and define $C = B^T A$. Then we will have $l = Cf$.

Hence, we see that the necessary and sufficient condition to collect all the coins is that $x \in \text{range}(C)$. This can be simply examined with $\text{rank}([C \ x]) = \text{rank}(C)$.

c) The code to solve parts c,e can be find at the bottom.

d) In part b, we saw that $l = Cf$. We know that there exists a sequence of input forces $f$ such that all but one of the $2n$ equations are satisfied, but we don’t know which one. Let $x^{(i)}$ be the location vector $x$ with the $i$th entry removed. Likewise, let $C^{(i)}$ be the transition matrix with the $i$th row of $C$ removed. If we can collect all coins but the $i$th one, then we will certainly have $x^{(i)} \in \text{range}(C^{(i)})$. We will loop over the coins and see whether it’s possible to collect all coins but one.

e) The following code solves the problem:

```matlab
clc
clear all
close all
robot_coin_collector

BT = zeros(2*n,2*n);
for i=1:2*n
    BT(i,1:i) = i-1/2:-1:1/2;
end

A = zeros(2*n,n);
for i=1:n
    A([2*i-1,2*i],i)=1;
end
C = BT*A;

%part c
if rank([C,x]) == rank(C)
    fprintf('All coins can be collected!\n')
else
    fprintf('All coins cannot be collected!\n')
end

%part e
for i=1:2*n
    xt = x([1:i-1,i+1:end]);
```

8
Ct = C([1:i-1,i+1:end],:);
if rank([Ct,xt])==rank(Ct)
    fprintf('The robot can collect all coins but %dth,' i);
    fprintf('and the input will be: 
')
    input = Ct\xt;
    disp(input)
end
end

hold on
plot(C*input,1:2*n,'r')
hold off

We see that all coins but the 7th can be collected and the associated input will be

\[ f = [1.0000, -4.0000, 7.0000, -10.0000, 20.0000, -35.0000]. \]

![Figure 1: Location of the coins and the trajectory of the robot](image)

5. **Projection matrices.** A matrix \( P \in \mathbb{R}^{n \times n} \) is called a projection matrix if \( P = P^T \), and \( P^2 = P \). (These properties are sometimes called symmetry and idempotency, respectively.)

   a) Show that if \( P \) is a projection matrix, then \( I - P \) is also a projection matrix.
b) Suppose \( U \in \mathbb{R}^{n \times k} \) has orthonormal columns. Show that \( UU^T \) is a projection matrix.
(The converse is also true: every projection matrix can be written as \( UU^T \) for some matrix \( U \) with orthonormal columns; you do not need to prove this.)

c) Suppose \( A \in \mathbb{R}^{n \times k} \) is skinny, and full rank. Show that \( A(A^T A)^{-1}A^T \) is a projection matrix.

d) Given \( S \subset \mathbb{R}^n \), and \( x \in \mathbb{R}^n \), the point \( \hat{x} \in S \) that is closest to \( x \) is called the projection of \( x \) onto \( S \). Show that if \( P \) is a projection matrix, then \( \hat{x} = Px \) is the projection of \( x \) onto \( \text{range}(P) \). (This is the origin of the term “projection matrix.”)

**Solution.**

a) Suppose \( P \) is a projection matrix. Since transposition distributes over sums, and \( P \) is symmetric, we have that

\[
(I - P)^T = I^T - P^T = I - P.
\]

Similarly, because \( P \) is idempotent, we have that

\[
(I - P)^2 = I - IP - PI + P^2 = I - P - P + P = I - P.
\]

Thus, \( I - P \) is symmetric and idempotent: that is, \( I - P \) is a projection matrix.

b) Since the transpose of a product is the product of the transposes in the reverse order, we have that

\[
(UU^T)^T = (U^T)^T U^T = UU^T.
\]

Because \( U \) has orthonormal columns, we have that \( U^TU = I \), and hence that

\[
(UU^T)^2 = U(U^TU)U^T = UIU^T = UU^T.
\]

Thus, \( UU^T \) is symmetric and idempotent: that is, \( UU^T \) is a projection matrix.

c) Using various properties of transposition, we have that

\[
(A(A^T A)^{-1}A^T)^T = (A^T)^T((A^T A)^{-1})^T A^T = A((A^T A)^T)^{-1}A^T
\]

\[
= A(A^T(A^T)^T)^{-1}A^T = A(A^T A)^{-1}A^T.
\]

Additionally, we have that

\[
(A(A^T A)^{-1}A^T)^2 = A(A^T A)^{-1}(A^T A)(A^T A)^{-1}A^T = A(A^T A)^{-1}A^T.
\]

Thus, \( A(A^T A)^{-1}A^T \) is symmetric and idempotent: that is, \( A(A^T A)^{-1}A^T \) is a projection matrix.

d) If \( P \) is a projection matrix, then we have that

\[
P^T(x - Px) = Px - P^2x = Px - Px = 0,
\]
where the first step uses the fact that $P$ is symmetric, and the second step uses the fact that $P$ is idempotent. An arbitrary element of $\text{range}(P)$ can be written as $Pz$. We have that
\[
\|x - Pz\|^2 = \|(x - Px) + (Px - Pz)\|^2 \\
= \|x - Px\|^2 + 2(x - z)^T P^T (x - Px) + \|P(x - z)\|^2 \\
= \|x - Px\|^2 + \|P(x - z)\|^2 \\
\geq \|x - Px\|^2.
\]

Thus, we see that $Px$ is the closest point to $x$ in $\text{range}(P)$: that is, $Px$ is the projection of $x$ onto $\text{range}(P)$. Intuitively, the fact that $P^T (x - Px) = 0$ means that $x - Px$ is orthogonal to (every vector in) $\text{range}(P)$. Then, for any vector $Pz \in \text{range}(P)$, $x - Pz$ is the hypotenuse of a right triangle with legs $x - Px$ and $Px - Pz$. Since the hypotenuse is always the longest side of a right triangle, this implies that $\|x - Pz\| \geq \|x - Px\|$. The geometry of this intuitive explanation is illustrated in Figure 2. The key step in this analysis is showing that $x - Px$ is orthogonal to $\text{range}(P)$; this is a specific instance of a general idea called the orthogonality principle.

6. **Sensor integrity monitor.** A suite of $m$ sensors yields measurement $y \in \mathbb{R}^m$ of some vector of parameters $x \in \mathbb{R}^n$. When the system is operating normally (which we hope is almost always the case) we have $y = Ax$, where $m > n$. If the system or sensors fail, or become faulty, then we no longer have the relation $y = Ax$. We can exploit the redundancy in our measurements to help us identify whether such a fault has occurred. We’ll call a measurement $y$ **consistent** if it has the form $Ax$ for some $x \in \mathbb{R}^n$. If the system is operating normally then our measurement will, of course, be consistent. If the system becomes faulty, we hope that the resulting measurement $y$ will become inconsistent, i.e., not consistent. (If we are really unlucky, the system will fail in such a way that $y$ is still consistent. Then we’re out of luck.) A matrix $B \in \mathbb{R}^{k \times m}$ is called an **integrity monitor** if the following holds:

- $By = 0$ for any $y$ which is consistent.
- $By \neq 0$ for any $y$ which is inconsistent.
If we find such a matrix $B$, we can quickly check whether $y$ is consistent; we can send an alarm if $By \neq 0$. Note that the first requirement says that every consistent $y$ does not trip the alarm; the second requirement states that every inconsistent $y$ does trip the alarm. Finally, the problem. Find an integrity monitor $B$ for the matrix

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & -2 \\ -2 & 1 & 3 \\ 1 & -1 & -2 \\ 1 & 1 & 0 \end{bmatrix}.$$ 

Your $B$ should have the smallest $k$ (i.e., number of rows) as possible. As usual, you have to explain what you’re doing, as well as giving us your explicit matrix $B$. You must also verify that the matrix you choose satisfies the requirements. **Hints:**

- You might find one or more of the matlab commands `orth`, `null`, or `qr` useful. Then again, you might not; there are many ways to find such a $B$.

- When checking that your $B$ works, don’t expect to have $By$ exactly zero for a consistent $y$; because of roundoff errors in computer arithmetic, it will be really, really small. That’s OK.

- Be very careful typing in the matrix $A$. It’s not just a random matrix.

**Solution.** The key challenge in this problem is to restate everything in common linear algebra and matrix terms. We need to find $B \in \mathbb{R}^{k \times m}$ such that the following hold:

- $By = 0$ for any consistent $y$

- $By \neq 0$ for any inconsistent $y$

Let’s analyze the conditions, starting with the first one. The set of consistent measurements is exactly equal to the range of the matrix $A$; so say that $By = 0$ for every consistent $y$ is the same as saying $\text{range}(A) \subseteq \text{null}(B)$, i.e., every element in the range of $A$ is also in the nullspace of $B$. In terms of matrices, the first condition means that for every $x$, we have $BAx = 0$. That’s true if and only if $BA = 0$. (Recall these are matrices, so we can have $BA = 0$ without $A = 0$ or $B = 0$.) We now consider the second condition. To say that every inconsistent $y$ has $By \neq 0$ is equivalent to saying that whenever $By = 0$, we have $y$ is consistent. This is the same as saying $\text{null}(B) \subseteq \text{range}(A)$. Putting this together with the first condition, we get a really simple condition: $\text{null}(B) = \text{range}(A)$. In other words, we need to find a matrix $B$ whose nullspace is exactly equal to the range of $A$. Now to find such a $B$ with smallest possible number of rows, we need $B$ to be full rank. Its rank must be $m$ minus the dimension of the range of $A$, i.e., $m - \text{rank}(A)$. Now that we know what we’re looking for, there are several ways to find such a $B$, given $A$. Note that whatever method we end up using we can check that we’ve got a solution by checking that $BA = 0$ and $B$ is full rank. One method relies on the fact from lectures that for any matrix $C$, $\text{null}(C)$ and $\text{range}(C^T)$ are orthogonal complements. It follows that $\text{null}(B)$ and $\text{range}(B^T)$ are orthogonal complements, and so are $\text{range}(A)$ and $\text{null}(A^T)$. We require that $\text{null}(B) = \text{range}(A)$, so this means their orthogonal
complements are equal, i.e., range($B^T$) = null($A^T$). In matlab, we can compute a basis for
the nullspace of $A^T$ using the command null. (In fact null gives us an orthonormal basis
for the nullspace, but for this problem all we care about is that we get a basis for the nullspace.)
This approach can be implemented with the simple matlab code:

```matlab
A = [ 1 2 1 ; 1 -1 -2; -2 1 3 ; 1 -1 -2; 1 1 0]; B = null(A')';
B*A
rank(B)
```

The matrix $BA$ does turn out to be zero for all practical purposes; the entries are very, very
small, but nonzero because of roundoff error in computer arithmetic. One subtlety you may
or may not have noticed is that $A$ is not full rank; it has rank 2. In fact, its third column is
equal to its second column minus its first column. That’s why we end with $k = 3$, and not 2,
as you might have expected. Another way to find such a $B$ uses the full QR factorization of
$A$. If we have QR factorization

$$A = [Q_1 Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix},$$

where $[Q_1 Q_2]$ is orthogonal and $R_1$ is upper triangular and invertible, then the columns of
$Q_1$ are an orthonormal basis for the range of $A$, and the columns of $Q_2$ are an orthonormal
basis for the orthogonal complement. Therefore we can take $B = Q_2^T$. This approach can be
carried out in matlab via

```matlab
[Q,R]=qr(A);
Q2 = Q(:,[3,4,5]); % get the last three columns of Q
B = Q2';
B*A rank(B)
```

Two common errors involved the size of $B$. In each case, $B$ satisfies $BA = 0$, so whenever $y$ is
consistent, we have $By = 0$. The first error was to have a $B$ that is too small, i.e., has fewer
than 3 rows. Such a $B$ doesn’t satisfy the second condition; there are inconsistent $y$’s with
$By = 0$. Therefore $B$’s with fewer than 3 rows aren’t integrity monitors. The opposite error,
of having $B$ with more than 3 rows, isn’t quite so bad. In this case, your $B$ doesn’t have the
minimal number of rows, but it is a real integrity monitor.

7. Solving linear equations via QR factorization. Consider the problem of solving the
linear equations $Ax = y$, with $A \in \mathbb{R}^{n \times n}$ nonsingular, and $y$ given. We can use the Gram-
Schmidt procedure to compute the QR factorization of $A$, and then express $x$ as $x = A^{-1}y =
R^{-1}(Q^T x) = R^{-1}z$, where $z = Q^T y$. In this exercise, you’ll develop a method for computing
$x = R^{-1}z$, i.e., solving $Rx = z$, when $R$ is upper triangular and nonsingular (which means its
diagonal entries are all nonzero).

The trick is to first find $x_n$; then find $x_{n-1}$ (remembering that now you know $x_n$); then
find $x_{n-2}$ (remembering that now you know $x_n$ and $x_{n-1}$); and so on. The algorithm you will
discover is called back substitution, because you are substituting known or computed values of
$x_i$ into the equations to compute the next $x_i$ (in reverse order). Be sure to explain why the
algorithm you describe cannot fail.
Solution. Suppose that

\[
\begin{bmatrix}
z_1 \\
z_2 \\
\vdots \\
z_n 
\end{bmatrix}, \quad
\begin{bmatrix}
r_{11} & r_{12} & \cdots & r_{1n} \\
0 & r_{22} & \cdots & r_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & r_{nn} 
\end{bmatrix}, \quad
\begin{bmatrix}
x_1 \\
x_2 \\
\vdots \\
x_n 
\end{bmatrix}
\]

Consider the linear equation corresponding to the last \((n)\)th row of \(R\), \(i.e.,\)

\[
z_n = r_{nn} x_n.
\]

Since \(r_{nn} \neq 0\) we can simply solve for \(x_n\) to get \(x_n = z_n / r_{nn}\). Now consider the linear equation corresponding to the \((n-1)\)th row of \(R\), \(i.e.,\)

\[
z_{n-1} = r_{(n-1)(n-1)} x_{n-1} + r_{(n-1)n} x_n
\]

and since \(r_{(n-1)(n-1)} \neq 0\) we get

\[
x_{n-1} = \frac{1}{r_{(n-1)(n-1)}} (z_{n-1} - r_{(n-1)n} x_n)
\]

with \(x_n\) found from the previous step. In general, if \(x_n, x_{n-1}, \ldots, x_{i+1}\) are known, \(x_i\) can be derived from the linear equation corresponding to the \(i\)th row of \(R\) as (assuming \(r_{ii} \neq 0\))

\[
x_i = \frac{1}{r_{ii}} \left( z_i - \sum_{j=i+1}^{n} r_{ij} x_j \right),
\]

where again we rely on \(r_{ii} \neq 0\), which comes from our assumption that \(A\) is nonsingular. Therefore, the \(x_i\)'s can be computed recursively for \(i = n, n-1, \ldots, 1\) by back substitution. This suggests the following simple algorithm:

\[
i := n; \\
\text{while } i \geq 1 \\
\quad \text{if } r_{ii} \neq 0 \\
\quad \quad x_i := \frac{1}{r_{ii}} \left( z_i - \sum_{j=i+1}^{n} r_{ij} x_j \right); \\
\quad \text{else} \\
\quad \quad \text{unique solution does not exist; break;}
\end{align*}
\[
\text{end}
\]
\[
i := i - 1;
\]
\[
\text{end}
\]