College of Engineering
Daniel Guggenheim School of Aerospace Engineering

ISYE 7750: Mathematical Foundations of Machine Learning

Homework 5

Professor:
Ashwin Pananjady
Gtech ISYE Professor

Student:
Tomoki Koike
AE MS Student

October 29, 2022

Table of Contents

I Problem One **2**

II Problem Two **9**

III Problem Three **12**

IV Problem Four **13**

V Problem Five **16**

VI Appendix **20**

I Problem One

In this problem, we will solve a stylized regression problem using the data set `hw05p1_clusterdata.mat`. This file contains (noisy) samples of a function $f(t)$ for $t \in [0,1]$. In fact, the function is the same as the one we used in Homework 4:

$$f_{\text{true}}(t) = \frac{\sin(12(t+0.2))}{t+0.2}.$$

The sample locations are in the vector `T`; the sample values are in `y`. If you plot these, you will see that the samples come in four clusters. We are going to use nonlinear regression using the orthobasis of Legendre polynomials — in the notes, these polynomials were defined on $[-1,1]$, we will use the analogous functions on $[0,1]$. To compute these, we can use the `legendreP` command in MATLAB[1]. If we define the function handle

```
lpoly = @(p,z) sqrt(2)*sqrt((2*p+1)/2)*legendreP(p, 2*z-1);
```

Then, for example, `lpoly(3,T)` will return samples of the 3rd order Legendre polynomial at locations in `T`.

(a) Find the best cubic fit to the data using least-squares. That is, find $w_0, \dots, w_3$ that minimizes

$$\min_{\mathbf{w}} \sum_{m=1}^{M} (y_m - f(t_m))^2 \quad \text{where} \quad f(t) = \sum_{n=0}^{3} w_n v_n(t),$$

where $v_n(t)$ is the nth order Legendre polynomial adapted to $[0,1]$. Let $\widehat{\mathbf{w}}$ be the solution to the above, and $\widehat{f}$ the corresponding cubic polynomial. Compute the sample error

$$\text{sample error} = \left(\sum_{m=1}^{M} (y_m - \widehat{f}(t_m))^2 \right)^{1/2} = \|\mathbf{y} - \mathbf{A}\widehat{\mathbf{w}}\|_2,$$

where $\mathbf{A}$ is the matrix you set up to solve the least-squares problem. Plot your solution $\widehat{f}(t)$ for $t \in [0,1]$, and overlay the sample values (t_m, y_m) — **the sample values should not have lines connecting them**[2].

(b) Compute the generalization error

$$\text{generalization error} = \left(\int_0^1 |\widehat{f}(t) - f_{\text{true}}(t)|^2 \, dt \right)^{1/2}.$$

You can either use some numerical integration package to do this (`integral()` in MATLAB), or you can simply sample the functions at 5000 points[3], take the sum of the squared difference, divide by 5000, then take the square root.

(c) Repeat part (a) and (b) for polynomials of order $p = 5, 10, 15, 20, 25$ (note that the number of basis functions is always $N = p + 1$). For each experiment, report the sample error, generalization error, and the largest and smallest singular value of the matrix $\mathbf{A}$. Make a plot of the sample error versus N and the generalization error versus N. For what value of N does least-squares start to fall apart and why? Why does the sample error go down monotonically with N but the generalization error does not? Answer these questions, and for each value of p, make a plot of the sample values, $\widehat{f}(t)$, and $f_{\text{true}}(t)$ overlaid on one another.

(d) Set $N = 25$. Plot the singular values of $\mathbf{A}$ for this N. Stabilize your least-squares solution by truncating the SVD; make a judgement call about what R' should be given your singular value plot, and explain your reasoning. Compute the sample and generalization errors.

[1] For Python, see `https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.legendre.html`.
[2] Use `plot(t,y,'o')` in MATLAB, for example.
[3] 5000 might be overkill here, but these computations are cheap.

(e) Again with $N = 25$, repeat part (d) and compute the truncated SVD estimate for $R' = 5, 6, \ldots, 25$. Plot the sample and generalization error versus R', and interpret in terms of noise error, approximation error, and null space error.

(f) Fix $N = 25$. Now we will consider the ridge regression estimate

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{A}\mathbf{w}\|_2^2 + \delta\|\mathbf{w}\|_2^2$$

Again examining the plot of the singular values of $\mathbf{A}$, come up with a reasonable value of δ to use above. Perform the regression, make a plot of your result, again overlaid on the samples and $f_{\text{true}}(t)$, and report the sample and generalization error. Also comment on what happens to both the sample and generalization error as you sweep the value of δ — provide representative plots.

Solution

Question (1)

The matrix $\mathbf{A}$ can be computed by taking `lpoly()` for the 0th to the 3rd order which spits out 100x1 data points for each computation. If we concatenate these 100x1 data points we obtain $\mathbf{A} \in \mathbb{R}^{100\times 4}$, a tall-skinny matrix. For such $\mathbf{A}$ matrix, the least square solution becomes

$$\hat{\mathbf{w}} = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top \mathbf{y}. \tag{I.1}$$

The sampling error for the found solution is 9.2511, and the plot of the solution and sample values are as follows.

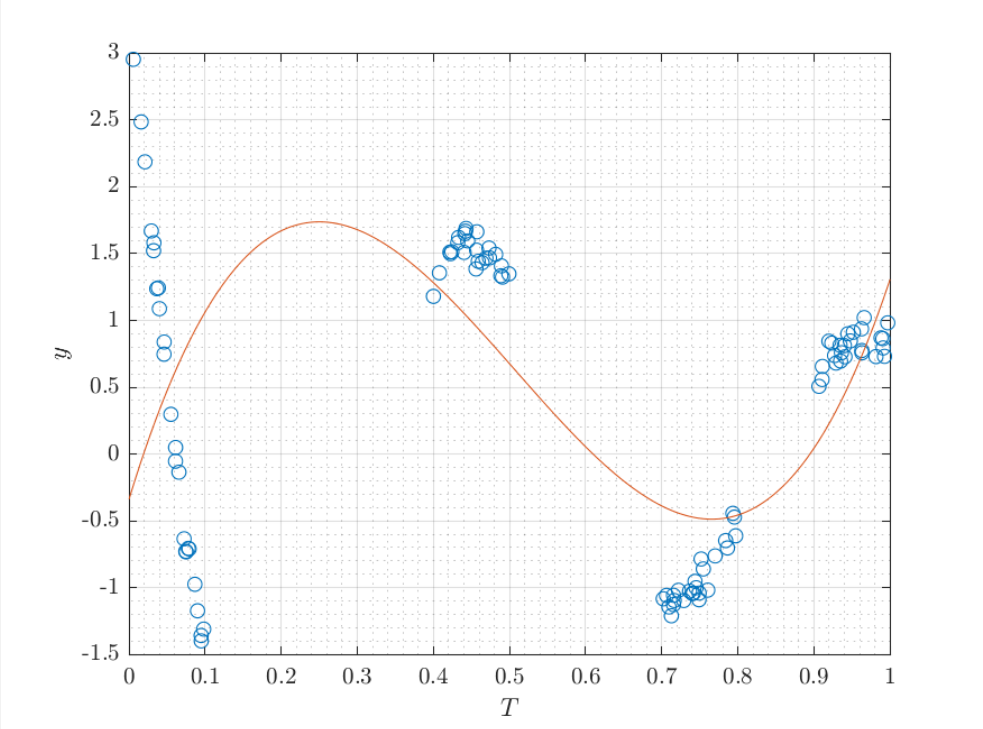

Figure 1: Least square solution of Problem (a) overlaid with sample values.

Question (b)

Let the f_{true} and $\hat{f}$ be defined in MATLAB as follows.

```
% Legendre Polynomials in [0, 1]
lpoly = @(p,z) sqrt(2)*sqrt((2*p+1)/2)*legendreP(p, 2*z-1);
% True function
ftrue = @(t) sin(12.*(t+0.2))./(t+0.2);
% Solution nonlinear regression
nr = @(t) 0;
for i = 0:3
  nr = @(t) nr(t) + what(i+1) * lpoly(i,t);
end
```

Then the generalized error can be found by

```
% Compute the generalized error
GE = sqrt(integral(@(t) (ftrue(t) - nr(t)).^2, 0, 1));
```

and the value is 1.7878.

Question (c)

N	Sample Error	Generalized Error	σ_{max}	$\sigma_{min}(\neq 0)$
6	1.5273	0.7173	15.5774	4.5046
11	0.7700	0.1642	16.5247	0.5323
16	0.7208	1.0754	17.3409	0.0871
21	0.6974	4.0060	18.6640	0.0126
26	0.6869	39.1053	19.8920	0.0015

Table 1: Results of $N = 6, 11, 16, 21, 26$ for least square solution.

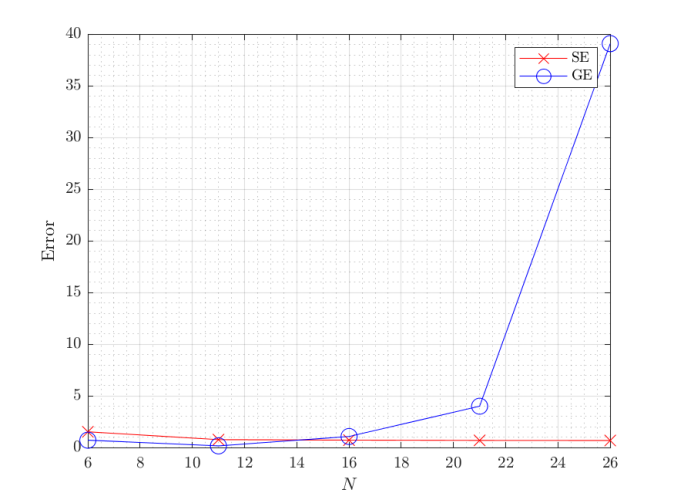

Figure 2: Sample error and generalized error trends for different orders of Legendre polynomial.

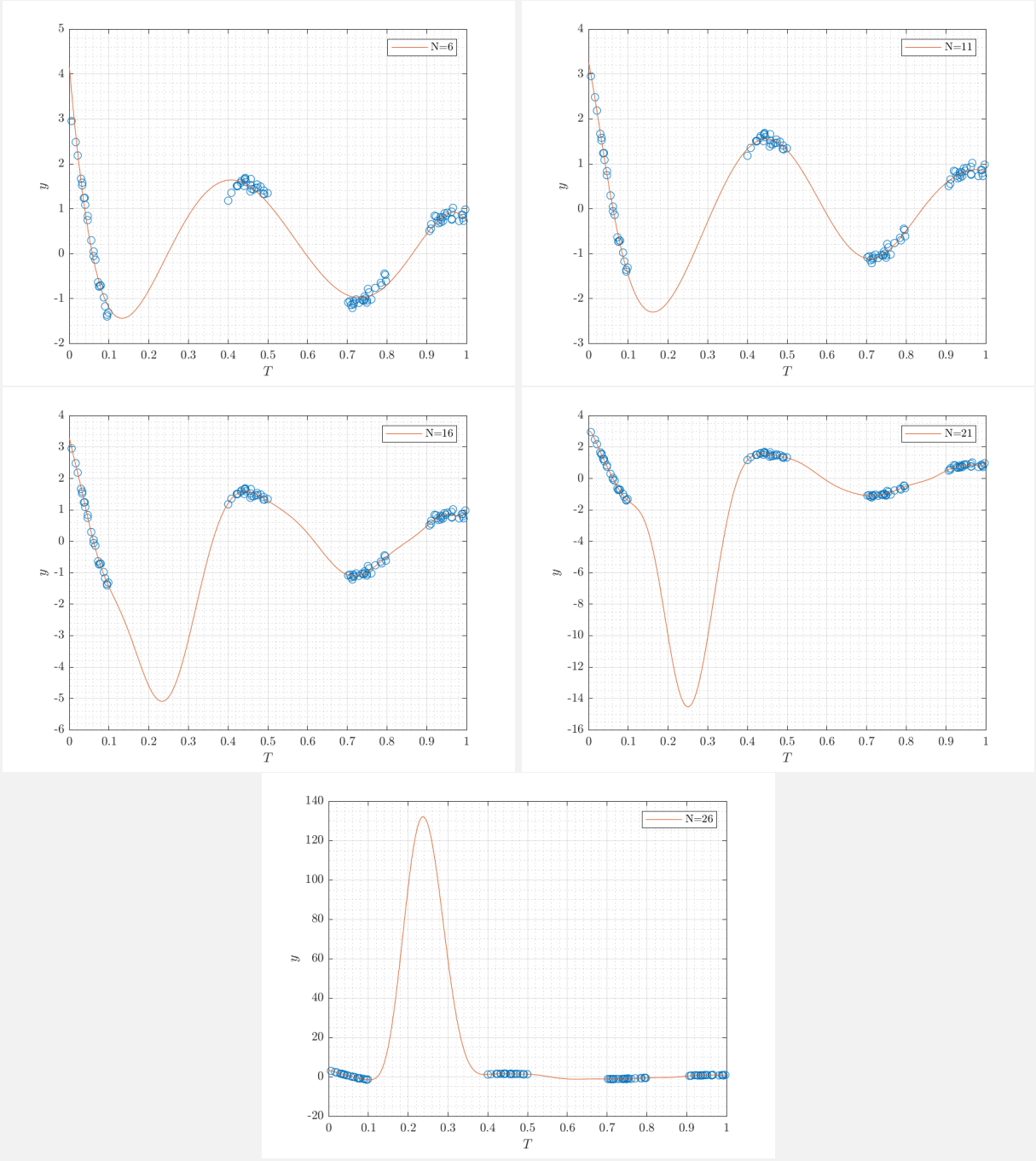

Figure 3: Plot of least square solution and sample values overlaid.

From Figure 2, we can tell that the generalized error spikes significantly after $N = 21$ which means that the least-squares solution starts to fall apart at this point. The reason behind this is that the min/max singular value, which governs the error of the least-squares solution, becomes large after $N = 21,\ 26$ since the value of the minimum non-zero singular value becomes an order smaller. This causes the maximum possible error to be much larger. Further, in Figure 3, the solution starts to be overfitted to the sample values and there is a large fluctuation for $N = 21,\ 26$ which causes the solution to diverge from the true sinusoidal function but minimizes the error between the sample points.

Question (d)

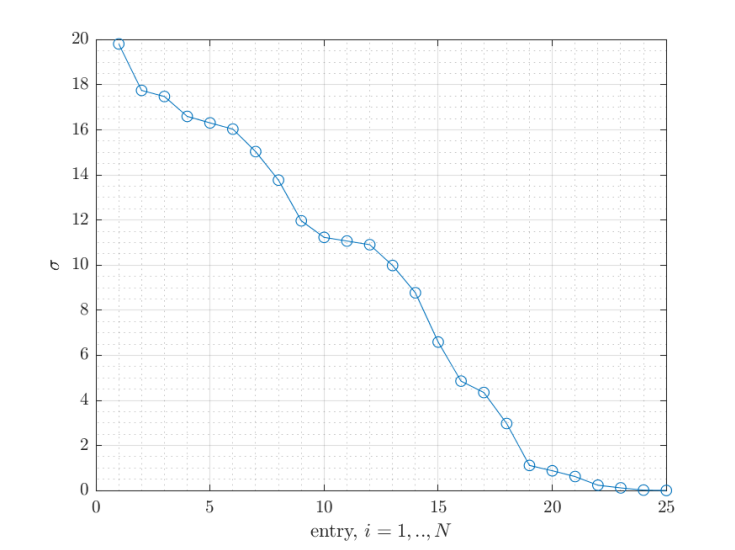

Figure 4: Singular values of $N = 25$.

From the Table 1 and Figure 4, we truncate the SVD so that the minimum singular value is larger than 0.1. That is, $N = 23$. Now, the solution for the truncated SVD becomes

$$\hat{\mathbf{w}}_t = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^\top \mathbf{y}$$

With this truncated solution we have a sample error of 0.6903 and generalized error of 0.8695. Before truncated they were 0.6902 and 1.3808 respectively.

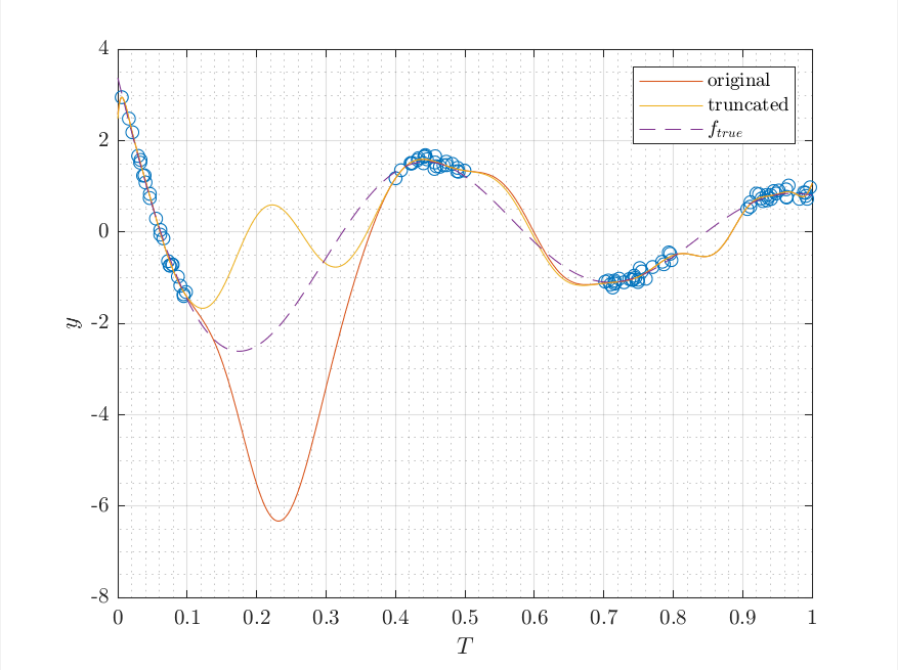

Figure 5: Least-squares solution of before and after truncating with SVD and f_{true}.

Question (e)

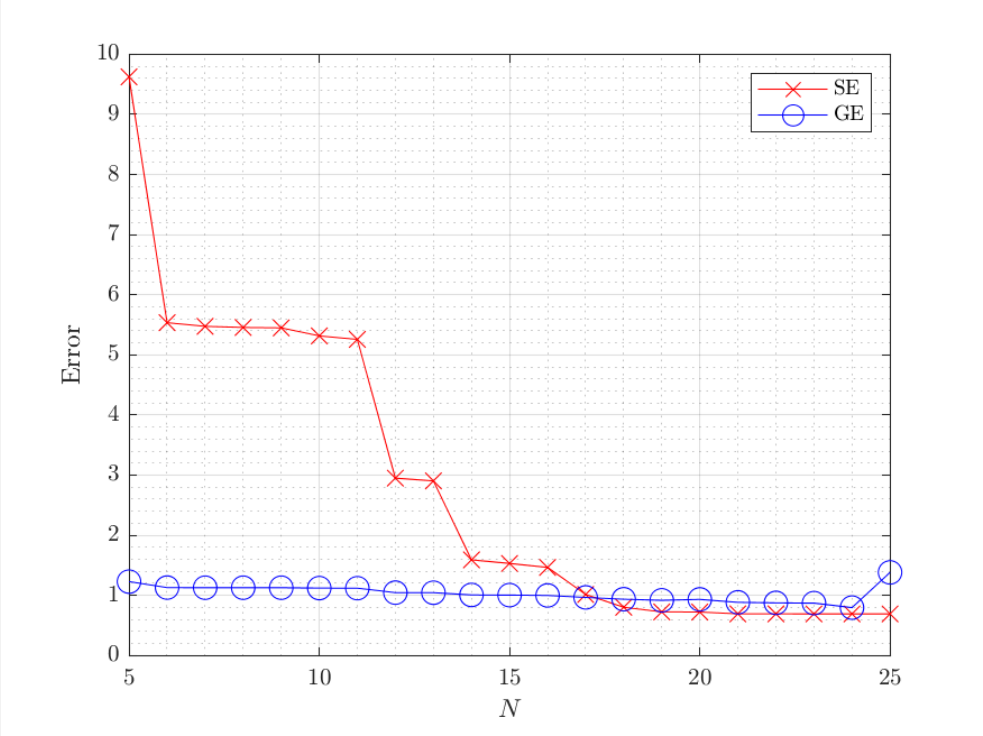

Figure 6: Sample error and generalized error of the truncated SVD least-squares solution.

The null space error which is fixed and does not change depending on the truncation by SVD is most likely to be presented in the plot of the generalized error above, and how the error has a constant amount present. This is also observable from how the sample error has some degree of error for truncation dimensions larger than 20. As the degree of truncation becomes larger the truncation error or approximation error becomes larger as can be seen in the sample error becoming much larger for $N \leq 11$. However, as the approximation error increases the dimension of the column space involved in the least-squares solution decreases meaning that the noise error effectively decreases.

Question (f)

The solution using ridge/Tikhonov regression is

$$\hat{\mathbf{w}}_r = (\mathbf{A}^\top \mathbf{A} + \delta \mathbf{I})^{-1} \mathbf{A}^\top \mathbf{y}$$

For this we select $\delta = 0.01$ and this gives us

$$\begin{aligned} \text{sample error} &= 0.6904 \\ \text{generalized error} &= 0.8606 \end{aligned}$$

The least-squares solution for this value of δ is plotted below.

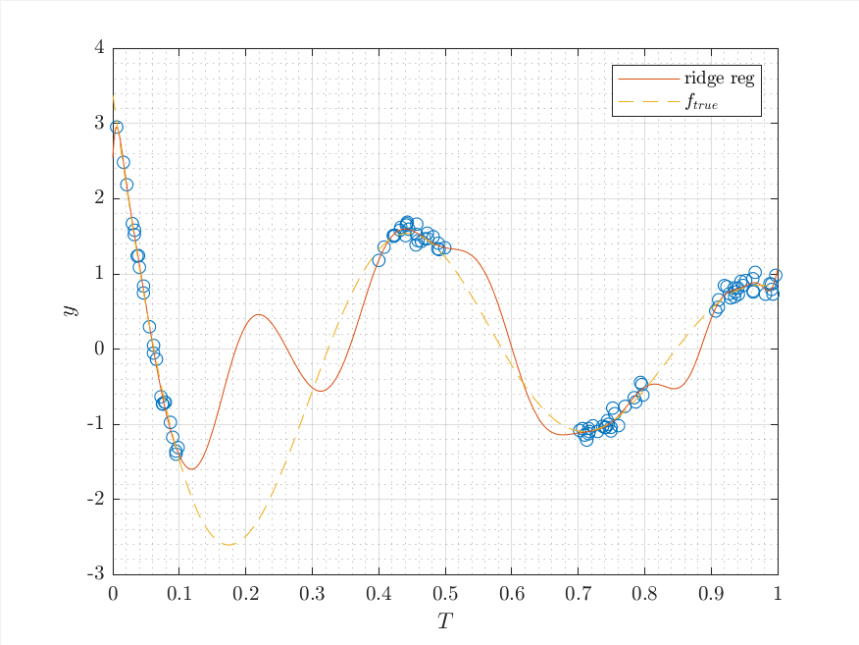

Figure 7: Least-squares solution for ridge regression.

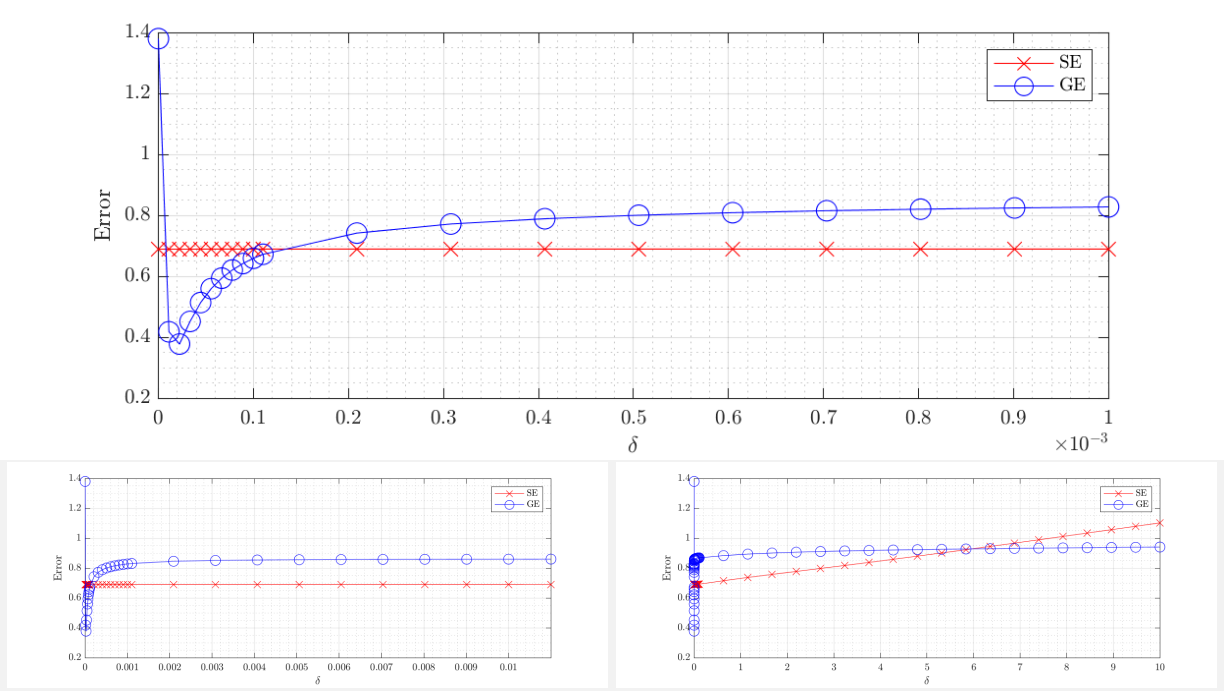

Figure 8: Sample error and generalized error trends w.r.t to the δ value.

As you sweep the δ values we see that the generalized errors initially start to decrease but it converges to some value as it becomes to get larger. However, the sample error steadily increases as the δ value becomes larger. Thus, it is best to use a δ value in between 0.001 to 0.2 to get a minimal error.

II Problem Two

Let

$$\mathbf{H} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -1 \\ -3 \end{bmatrix},$$

and let $f : \mathbb{R}^2 \to \mathbb{R}$ be

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{H}\mathbf{x} - \mathbf{b}^\top \mathbf{x}.$$

(a) What is the smallest value that f takes on $\mathbb{R}^2$? At what $\mathbf{x}$ does it achieve this minimum value?

(b) Write $f(\mathbf{x})$ out as a quadratic function in x_1, x_2 where $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. In other words, fill in the blanks below

$$f(\mathbf{x}) = \_\_x_1^2 + \_\_x_2^2 + \_\_x_1x_2 + \_\_x_1 + \_\_x_2.$$

(c) Using MATLAB or Python, make a contour plot of $f(\mathbf{x})$ around its minimizer in $\mathbb{R}^2$. Compute the eigenvectors and eigenvalues of $\mathbf{H}$, and discuss what role they are playing in the geometry of your sketch.

(d) On top of the contour plot, trace out the first four steps of the gradient descent algorithm starting at $\mathbf{x}_0 = \mathbf{0}$.

(e) On top of the contour plot, trace out the two steps of the conjugate gradient method starting at $\mathbf{x} = \mathbf{0}$.

Solution

Question (1)

By taking the gradient of $f(\mathbf{x})$ we have

$$\nabla_{\mathbf{x}} f = \frac{1}{2}(\mathbf{H} + \mathbf{H}^\top)\mathbf{x} - \mathbf{b}$$

Then the minimum value of f is achieved when $\nabla_{\mathbf{x}} f = 0$, i.e. $\mathbf{x} = 2(\mathbf{H} + \mathbf{H}^\top)^{-1}\mathbf{b}$. Hence,

$$\min f(\mathbf{x}) = -2.3333 = -\frac{7}{3} \qquad \text{where} \quad \mathbf{x} = \begin{bmatrix} 0.3333 \\ -1.6667 \end{bmatrix} = \begin{bmatrix} 1/3 \\ -5/3 \end{bmatrix}. \tag{II.1}$$

Question (b)

This is simply

$$f(\mathbf{x}) = x_1^2 + x_2^2 + x_1x_2 + x_1 + 3x_2. \tag{II.2}$$

Question (c)

The eigenvalues and eigenvectors of the matrix $\mathbf{H}$ are

$$\lambda_1 = 1 \quad \text{and} \quad \lambda_2 = 3,$$

$$\mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \quad \text{and} \quad \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

The eigenvalues are acting as the radius of the semi-major and semi-minor axes of the ellipse shape, and the eigenvectors represent the directions of those axes as depicted in the next figure.

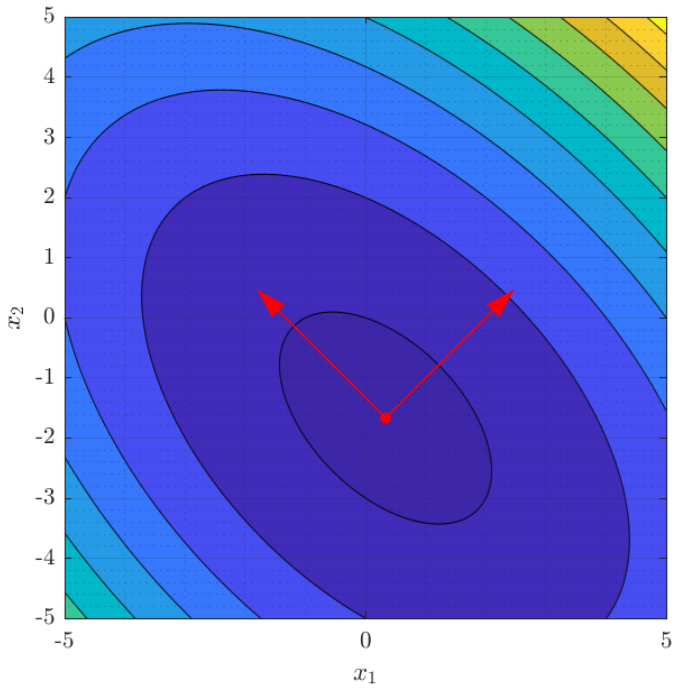

Figure 9: Contour plot of $f(\mathbf{x})$ with minimum noted as red dot.

Question (d)

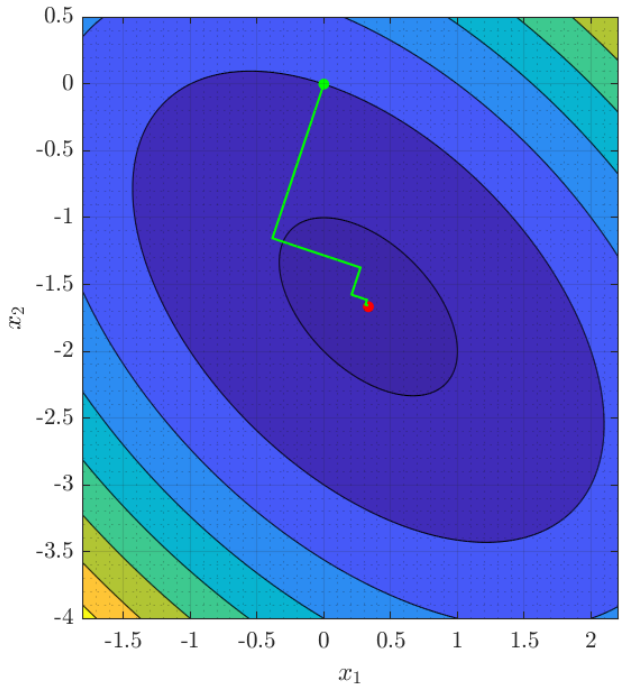

Figure 10: First 4 steps of gradient descent.

Question (e)

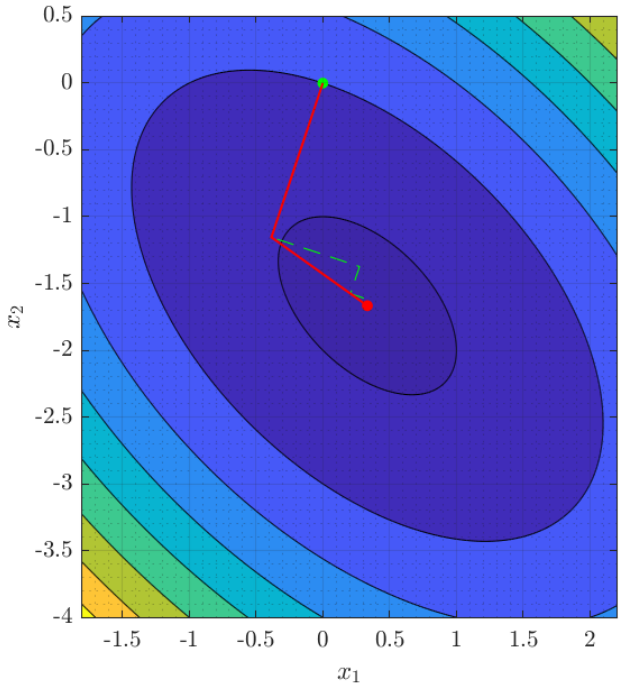

Figure 11: First 2 steps of conjugate descent in red overlaid with gradient descent in green.

III Problem Three

(a) Write a MATLAB (or Python) function `gdsolve` that implements the gradient descent algorithm for solving linear systems of equations. The function should be called as

```
[x, iter] = gdsolve(H, b, tol, maxiter);
```

where `H` is a $N \times N$ symmetric positive definite matrix, `b` is a vector of length N, and `tol` and `maxiter` specify the halting conditions. Your algorithm should iterate until $\|\mathbf{b} - \mathbf{H}\mathbf{x}_k\|_2 / \|\mathbf{b}\|_2$ is less than `tol` or the maximum number of iterations `maxiter` has been reached. For the outputs: `x` is your solution, and `iter` is the number of iterations that were performed. Run your code on the $\mathbf{H}$ and $\mathbf{b}$ in the file `hw05p3_data.mat` for a `tol` of 10^{-6}. Report the number of iterations needed for convergence, and for your solution $\widehat{\mathbf{x}}$ verify that $\|\mathbf{b} - \mathbf{H}\widehat{\mathbf{x}}\|_2 / \|\mathbf{b}\|_2$ is within the specified tolerance.

(b) Write a MATLAB (or Python) function `cgsolve.m` that implements the conjugate gradient method. The function should be called as

```
[x, iter] = cgsolve(H, b, tol, maxiter);
```

where the inputs and outputs have the same interpretation as in the previous problem. Run your code on the $\mathbf{H}$ and $\mathbf{b}$ in the file `hw05p3_data.mat` (same data as in part(a)) for a `tol` of 10^{-6}. Report the number of iterations needed for convergence, and for your solution $\widehat{\mathbf{x}}$ verify that $\|\mathbf{b} - \mathbf{H}\widehat{\mathbf{x}}\|_2 / \|\mathbf{b}\|_2$ is within the specified tolerance. Compare against the gradient descent algorithm in part (a).

Solution

Question (a)

The code is submitted separately. For the gradient descent algorithm we have the following results.

$$\begin{aligned} \text{iterations} &= 263 \\ \|\mathbf{b} - \mathbf{H}\hat{\mathbf{x}}\|_2 \,/\, \|b\|_2 &= 9.0447\text{e}{-7} \end{aligned}$$

Question (b)

Similarly for the conjugate gradient method we have the following results.

$$\begin{aligned} \text{iterations} &= 57 \\ \|\mathbf{b} - \mathbf{H}\hat{\mathbf{x}}\|_2 \,/\, \|b\|_2 &= 8.3597\text{e}{-7} \end{aligned}$$

IV Problem Four

Let $\mathbf{A}$ be a banded tridiagonal matrix:

$$\mathbf{A} = \begin{bmatrix} d_1 & c_1 & 0 & 0 & 0 & \cdots & 0 \\ f_1 & d_2 & c_2 & 0 & 0 & \cdots & 0 \\ 0 & f_2 & d_3 & c_3 & 0 & \cdots & 0 \\ 0 & 0 & f_3 & d_4 & c_4 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & f_{N-2} & d_{N-1} & c_{N-1} \\ 0 & 0 & 0 & \cdots & & f_{N-1} & d_N \end{bmatrix}$$

(a) Argue that the LU factorization of $\mathbf{A}$ has the form

$$\mathbf{A} = \begin{bmatrix} * & 0 & 0 & 0 & \cdots & 0 \\ * & * & 0 & 0 & \cdots & 0 \\ 0 & * & * & 0 & \cdots & 0 \\ 0 & 0 & * & * & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & * & * \end{bmatrix} \begin{bmatrix} * & * & 0 & 0 & 0 & \cdots & 0 \\ 0 & * & * & 0 & 0 & \cdots & 0 \\ 0 & 0 & * & * & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \\ 0 & 0 & \cdots & & & * & * \\ 0 & 0 & \cdots & & & 0 & * \end{bmatrix},$$

where $*$ signifies a non-zero term.

(b) Write down an algorithm that computes the LU factorization of $\mathbf{A}$, meaning the $\{\ell_i\}, \{u_i\}$, and $\{g_i\}$ below

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \ell_1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \ell_2 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \ell_3 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & \ell_{N-1} & 1 \end{bmatrix} \begin{bmatrix} u_1 & g_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & u_2 & g_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & u_3 & g_3 & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \\ 0 & 0 & \cdots & & & u_{N-1} & g_{N-1} \\ 0 & 0 & \cdots & & & 0 & u_N \end{bmatrix},$$

I will get you started:

$u_1 = d_1$
for $k = 2, \ldots, N$
  $g_{k-1} =$
  $\ell_{k-1} =$
  $u_k =$
end

(c) What is the computational complexity of the algorithm above? (How does the number of computations scale with N?) Once the LU factorization is in hand, how does solving $\mathbf{Ax} = \mathbf{b}$ scale with N?

Solution

Question (1)

Let us use a $\mathbf{A} \in \mathbb{R}^{3\times 3}$ for an example of computation. Let

$$\mathbf{A} = \begin{bmatrix} d_1 & c_2 & 0 \\ f_1 & d_2 & c_2 \\ 0 & f_2 & d_3 \end{bmatrix}$$

Then the LU-decomposition progresses as follows.

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -f_1/d_1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{L}_1} \mathbf{A} = \underbrace{\begin{bmatrix} d_1 & c_1 & 0 \\ 0 & d_2 - c_1 f_1/d_1 & c_2 \\ 0 & f_2 & d_3 \end{bmatrix}}_{\mathbf{A}_1}$$

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -f_2/(d_2 - c_1 f_1/d_1) & 1 \end{bmatrix}}_{\mathbf{L}_2} \mathbf{A}_1 = \underbrace{\begin{bmatrix} d_1 & c_1 & 0 \\ 0 & d_2 - c_1 f_1/d_1 & c_2 \\ 0 & 0 & d_3 - c_2 f_2/(d_2 - c_1 f_1/d_1) \end{bmatrix}}_{\mathbf{A}_2}$$

Thus,

$$\mathbf{A} = \underbrace{\mathbf{L}_1^{-1}\mathbf{L}_2^{-1}}_{\mathbf{L}} \underbrace{\mathbf{A}_2}_{\mathbf{U}}$$

Now if we let $u_1 = d_1$ we can say

$$\begin{aligned} g_{k-1} &= c_{k-1} \\ l_{k-1} &= -\frac{f_{k-1}}{u_{k-1}} \\ u_k &= d_k + g_{k-1} l_{k-1} \end{aligned} \tag{IV.1}$$

To prove by induction, we assume that the k-th iteration holds true for the tridiagonal LU-decomposition. Then it is natural that the $k+1$ iteration is the same algorithm repeated for the next elements. Thus, it shows that the decomposition holds the form of

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \ell_1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \ell_2 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \ell_3 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & \ell_{N-1} & 1 \end{bmatrix} \begin{bmatrix} u_1 & g_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & u_2 & g_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & u_3 & g_3 & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \\ 0 & 0 & \cdots & & & u_{N-1} & g_{N-1} \\ 0 & 0 & \cdots & & & 0 & u_N \end{bmatrix}$$

Question (b)

As represented in (IV.1), we can write the algorithm as

Algorithm 1: LU-decomposition of tridiagonal matrix

Input: matrix $\mathbf{H}$
Output: lower triangular matrix $\mathbf{L}$, upper triangular matrix $\mathbf{U}$
Function TDLU(H)
  $\mathbf{L} \leftarrow \mathbf{I}$;
  $\mathbf{U} \leftarrow \mathbf{0}$;
  $u_1 \leftarrow d_1$;
  $\mathbf{U}[1,1] \leftarrow u_1$;
  for $k = 2,\ldots,N$ **do**
    $g_{k-1} \leftarrow c_{k-1}$;
    $l_{k-1} \leftarrow -\frac{f_{k-1}}{u_{k-1}}$;
    $u_k \leftarrow d_k + g_{k-1} l_{k-1}$;
    $\mathbf{L}[k, k-1] \leftarrow l_{k-1}$;
    $\mathbf{U}[k,k] \leftarrow u_k$;
    $\mathbf{U}[k-1,k] \leftarrow g_k$;
  return $\mathbf{L}$, $\mathbf{U}$

Question (c)

So we have 3 operations for $N-1$ times so the algorithm we would have flops of $3(N-1)$. Then solving for $\mathbf{Ax} = \mathbf{b}$ we have $c \cdot \mathcal{O}(N^2)$ where $0 \leq c \leq 1$. Since it involves a LU decomposition but with a much more smaller order of computation since it is a tridiagonal.

V Problem Five

The Gram-Schmidt process is a method for orthonormalizing a set of vectors in an inner product space. The method works as follows: if $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N\}$ is a set of linearly independent vectors in $\mathbb{R}^M$ (so clearly $N \leq M$) then we can generate a sequence of orthonomal vectors $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N\}$ such that

$$\text{Span}\left(\{\mathbf{a}_1, \ldots, \mathbf{a}_N\}\right) = \text{Span}\left(\{\mathbf{q}_1, \ldots, \mathbf{q}_N\}\right)$$

using

$$\mathbf{q}_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|_2}$$

and for $k = 2, \ldots, N$:

$$\mathbf{w}_k = \mathbf{a}_k - \sum_{\ell=1}^{k-1} \langle \mathbf{a}_k, \mathbf{q}_\ell \rangle \, \mathbf{q}_\ell,$$
$$\mathbf{q}_k = \frac{\mathbf{w}}{\|\mathbf{w}_k\|_2}.$$

(a) As a warm-up, find $\mathbf{q}_1, \mathbf{q}_2$ and $\mathbf{q}_3$ when

$$\mathbf{a}_1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}.$$

Feel free to use a computer to do the calculations; just explain what you did in your write-up.

(b) For $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3\}$ as in part (a), let

$$\mathbf{A} = \begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ | & | & | \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} | & | & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \\ | & | & | \end{bmatrix}.$$

Show how we can write $\mathbf{A} = \mathbf{QR}$, where $\mathbf{R}$ is upper triangular. Do this by explicitly calculating $\mathbf{R}$. (Hint: just keep track of your work while doing part (a)).

(c) Suppose I run the algorithm above on a general $M \times N$ matrix $\mathbf{A}$ with linearly independent columns (full column rank). Explain how the Gram-Schmidt algorithm can be interpreted as finding a $M \times N$ matrix $\mathbf{Q}$ with orthonormal columns and an upper triangular matrix $\mathbf{R}$ such that $\mathbf{A} = \mathbf{QR}$. Do this by explicitly writing what the entries of $\mathbf{R}$ are in terms of the quantities that appear in the algorithm. This is called the *QR decomposition* of $\mathbf{A}$.

(d) Prove that an upper triangular matrix is invertible if and only if the elements along the diagonal are nonzero. Using this to show that the linear independence of the columns of $\mathbf{A}$ implies that all of the entries along the diagonal of $\mathbf{R}$ will be nonzero?

(e) Suppose that an $M \times N$ matrix $\mathbf{A}$ has full column rank. Show that the solution to the least-squares problem

$$\min_{\mathbf{x} \in \mathbb{R}^N} \; \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$$

is $\widehat{\mathbf{x}} = \mathbf{R}^{-1}\mathbf{Q}^\top \mathbf{y}$, where $\mathbf{A} = \mathbf{QR}$ is the QR decomposition of $\mathbf{A}$.

Solution

Question (a)

Using the Gram-Schmidt algorithm we normalize the vector $\mathbf{a}_1$ to get vector $\mathbf{q}_1$, which is

$$\mathbf{q}_1 = \begin{bmatrix} 0.4472 \\ 0.4472 \\ -0.4472 \\ -0.4472 \\ -0.4472 \end{bmatrix}.$$

Then compute

$$\mathbf{w}_2 = \mathbf{a}_2 - \langle \mathbf{a}_2, \mathbf{q}_1 \rangle \, \mathbf{q}_1,$$

then

$$\mathbf{q}_2 = \frac{\mathbf{w}_2}{\|w_2\|_2}.$$

Finally, repeat this step again by

$$\mathbf{w}_3 = \mathbf{a}_3 - \langle \mathbf{a}_3, \mathbf{q}_1 \rangle \, \mathbf{q}_1 - \langle \mathbf{a}_3, \mathbf{q}_2 \rangle \, \mathbf{q}_2,$$

then

$$\mathbf{q}_3 = \frac{\mathbf{w}_3}{\|w_3\|_2}.$$

Which gives

$$\mathbf{q}_2 = \begin{bmatrix} 0.5477 \\ 0.5477 \\ 0.3651 \\ 0.3651 \\ 0.3651 \end{bmatrix}, \qquad \mathbf{q}_3 = \begin{bmatrix} 0.4629 \\ -0.4629 \\ 0.3086 \\ -0.6172 \\ 0.3086 \end{bmatrix}.$$

Question (b)

Since we can reorganize the Gram-Schmidt to be

$$\mathbf{a}_k = \mathbf{w}_k + \sum_{\ell=1}^{k-1} \langle \mathbf{a}_k, \mathbf{q}_\ell \rangle \, \mathbf{q}_\ell$$

We can say that

$$\mathbf{A} = \begin{bmatrix} | & | & | \\ \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \langle \mathbf{a}_1, \mathbf{q}_1 \rangle & \langle \mathbf{a}_2, \mathbf{q}_1 \rangle & \cdots & \langle \mathbf{a}_N, \mathbf{q}_1 \rangle \\ 0 & \langle \mathbf{a}_2, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{a}_N, \mathbf{q}_2 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle \mathbf{a}_N, \mathbf{q}_N \rangle \end{bmatrix}.$$

Which gives us

$$\mathbf{R} = \begin{bmatrix} 2.2361 & -0.4472 & -0.4472 \\ 0 & 2.1909 & 0.3651 \\ 0 & 0 & 2.1602 \end{bmatrix}.$$

Question (c)

As found in Question (b) the entries of the upper triangular matrix are the inner products between the **a** column vectors and **q**. Thus the algorithm is merely identical to the Gram-Schmidt but storing the **q** vectors and the inner products into a matrix for each iteration. Where the implementation of the algorithm in MATLAB would be as follows.

```
function [Q, R] = QRfactor(A)
% QRFACTOR: Computes the QR factorization for nonsingular matrix A.
% Author: Tomoki Koike
% Organization: Georgia Institute of Technology, Aerospace Engineering
%
% >>Input
%   A: Nonsingular matrix where column spans some Euclidean space.
%
% <<Output
%   Q: matrix consisting of orthonormal basis vectors.
%   R: upper triangular matrix.
[M,N] = size(A);
Q = zeros(M,N);
R = zeros(N,N);
Q(:,1) = A(:,1) / norm(A(:,1));
R(1,1) = dot(A(:,1), Q(:,1));

for k = 2:N
  wk = A(:,k);
  for l = 1:k-1
    tmp = dot(A(:,k), Q(:,l));
    wk = wk - tmp * Q(:,l);
    R(l,k) = tmp;
  end
  Q(:,k) = wk / norm(wk);
  R(k,k) = dot(A(:,k), Q(:,k));
end
end
```

Question (d)

($\Longleftarrow$) If the upper triangular matrix of $\mathbf{R} \in \mathbb{R}^{N\times N}$ has all nonzero entries then it has N number of pivots meaning that the reduced order echelon form creates an identity matrix and rank $(\mathbf{R}) = N$. Thus, all columns of the matrix are linear independent, and therefore, the matrix is invertible.

($\Longrightarrow$) We shall prove this by contradiction. Let the matrix be invertible. Suppose that the r_{ii} or the i-th diagonal entry of the matrix $\mathbf{R}$ is zero. Since the determinant of a triangular matrix is the product of all diagonal entries. Hence, if even one of the diagonal entries are 0 the determinant of the matrix will be 0 which implies that the matrix is singular. This contradicts with our assumption, and therefore, all diagonal entries must be nonzero.

■

Question (e)

The solution of a typical least-squares problem is as

$$\hat{\mathbf{x}} = (\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top \mathbf{y}$$

if we plug in the QR decomposition we have

$$\hat{\mathbf{x}} = (\mathbf{R}^\top \mathbf{Q}^\top \mathbf{Q}\mathbf{R})^{-1}\mathbf{R}^\top \mathbf{Q}^\top \mathbf{y}$$

since $\mathbf{Q}$ is an orthogonal matrix $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$. Hence,

$$\hat{\mathbf{x}} = (\mathbf{R}^\top \mathbf{Q}^\top \mathbf{Q}\mathbf{R})^{-1}\mathbf{R}^\top \mathbf{Q}^\top \mathbf{y} = \mathbf{R}^{-1}(\mathbf{R}^\top)^{-1}\mathbf{R}^\top \mathbf{Q}^\top \mathbf{y} = \mathbf{R}^{-1}\mathbf{Q}^\top \mathbf{y}$$

■

VI Appendix