Math Foundations of ML, Fall 2022

Homework #5

Due Monday October 31, at 5:00pm ET

As stated in the syllabus, unauthorized use of previous semester course materials is strictly prohibited in this course.

1. 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 $w_0, \ldots, w_3$ that minimizes

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

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

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

where $\boldsymbol{A}$ 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].

[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.

Solution.
Please see "P1.ipynb" for the code and Figure 1 for the plot of $\widehat{f}$ overlaid on the samples. The regression gives coefficients

$$\widehat{\boldsymbol{w}} = [0.6124, -0.4617, -0.0569, 0.6134]^\top.$$

The sample error is 9.2511.

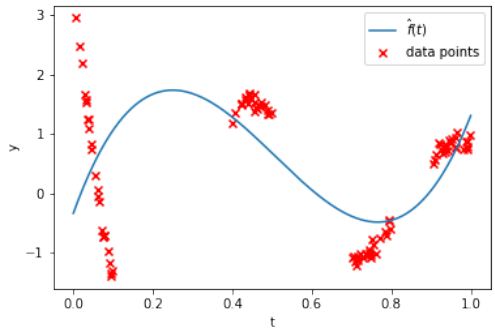

Figure 1: $\widehat{f}$ overlaid on the samples.

(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.

Solution.
Please see "P1.ipynb" for the code. The generalization error is 1.7878.

(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 $\boldsymbol{A}$. 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.

Solution.
Please see "P1.ipynb" for the code. We report the sample errors, generalization errors, and the largest and smallest singular values of the matrix $\boldsymbol{A}$ in Figure 2. Please see Figure 3 for the plot of the sample and generalization error versus N, and Figure 4 for the plot of $\widehat{f}$ overlaid on the samples and f_{true} for each value of $p = 5, 10, 15, 20, 25$.

[3] 5000 might be overkill here, but these computations are cheap.

It appears that least-squares starts to fall apart somewhere between $p = 10$ and $p = 15$, since this is the range where we start to observe an increasing generalization error and also $\widehat{f}$ begins to become extreme in regions between clusters of samples. Sample error decreases monotonically with p because the space of $(p+1)$th order polynomials contains the space of pth order polynomials. Generalization error is determined using the true function, information which is hidden from the regression, therefore there's no guarantee on how the generalization error may behave for increasing p. In general, we will observe generalization error decrease as p increases, until a point where the regression begins overfitting the sample data. At this point, one should expect generalization error to increase as p increases and overfitting ensues.

p	Sample Error	Generalization Error	Largest Singular Value	Smallest Singular Value
5	1.5272	0.7173	15.5774	4.5046
10	0.77	0.1642	16.5247	0.5323
15	0.7208	1.0754	17.3409	0.0871
20	0.6974	4.006	18.664	0.0126
25	0.6869	39.1053	19.892	0.0015

Figure 2: Sample errors, generalization errors and min/max singular values.

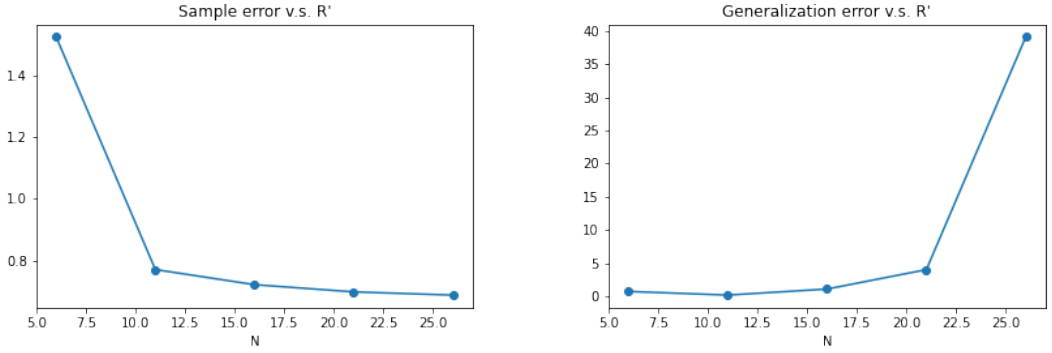

Figure 3: Sample and generalization error versus N.

(d) Set $N = 25$. Plot the singular values of $\boldsymbol{A}$ 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.

Solution.

Please see "P1.ipynb" for the code and Figure 10 for the plot of the singular values of $\boldsymbol{A}$. We chose $R' = 18$ because after this point, the singular values decrease far less quickly, and all further singular values are roughly 1 or less. For this truncation, the sample error is 0.8013 and the generalization error is 0.9344.

(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.

Solution.

Please see "P1.ipynb" for the code and Figure 6 for the plot of the sample and

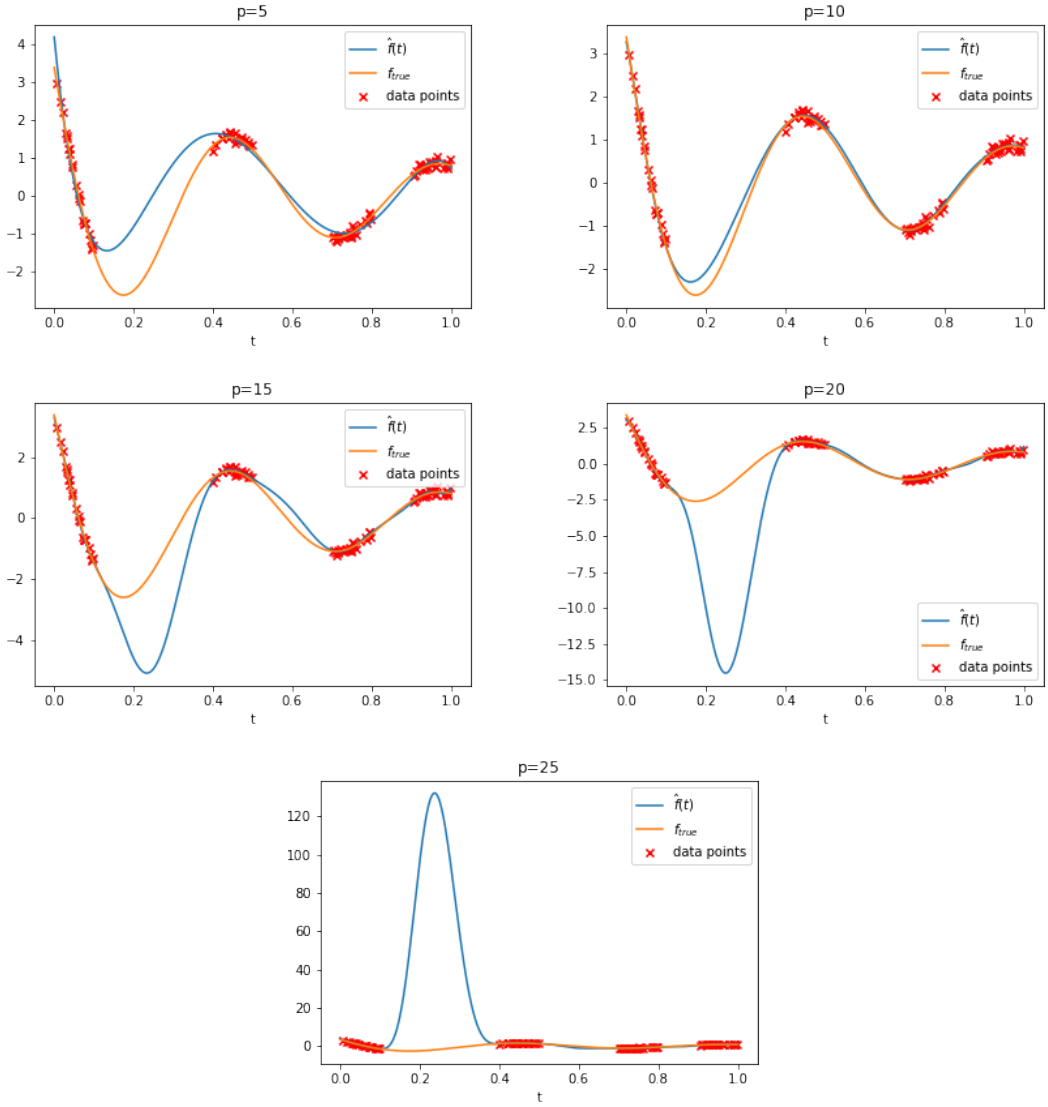

Figure 4: $\widehat{f}$ overlaid on the samples and f_{true} for $p = 5, 10, 15, 20, 25$.

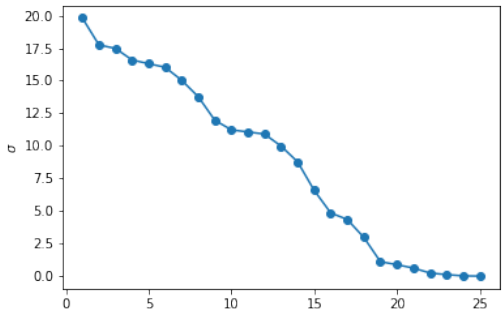

Figure 5: Singular values of $\boldsymbol{A}$.

generalization error versus R'. Null space error remains constant regardless of R', whereas the truncation/approximation error increases as R' decreases and the noise error decreases as R' decreases.

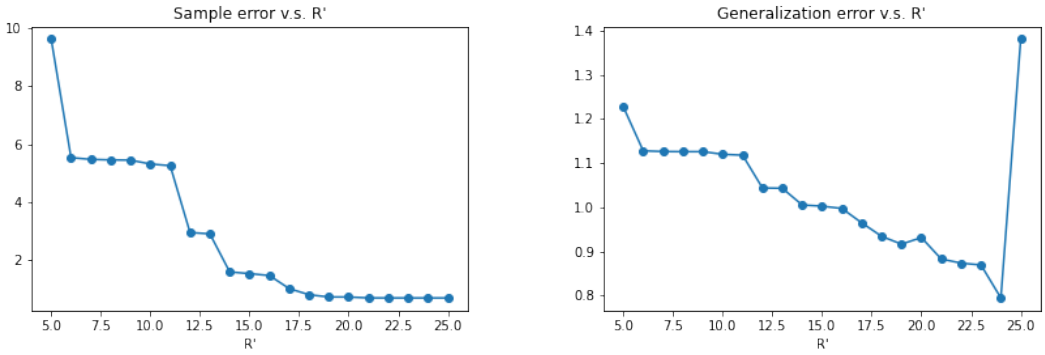

Figure 6: Sample and generalization error versus R'.

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

$$\underset{\boldsymbol{w}}{\text{minimize}} \, \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}\|_2^2 + \delta\|\boldsymbol{w}\|_2^2$$

Again examining the plot of the singular values of $\boldsymbol{A}$, 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.

Please see "P1.ipynb" for the code. We chose $\delta = 10^{-3}$ because the singular values associated to the Tikhonov inverse at this value closely matched those of the naive pseudo-inverse, except for the end values where large values yield extreme noise error. Please see Figure 7 for the plot of $\widehat{f}$ overlaid on the samples and f_{true}. The sample error is 0.6903 and the generalization error is 0.8287. In Figure 8, we see the impact of the choice of δ on sample and generalization errors. Sample error seems to only increase as δ does, and it appears that having a small δ of possibly 10^{-5} may be ideal in terms of generalization error. Having a slight amount of regularization is beneficial, but too much can ruin the regression.

2. Let

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

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

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{x}.$$

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

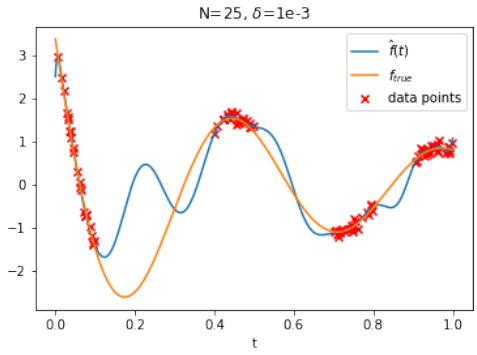

Figure 7: $\widehat{f}$ overlaid on the samples and f_{true}.

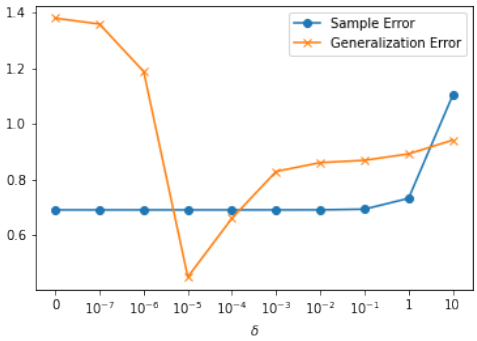

Figure 8: Sample and generalization error versus δ.

Solution.
Since at the minimum value of the function, we have

$$\nabla f(x) = Hx - b = 0.$$

Thus, at

$$x^* = H^{-1}b = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} -1 \\ -3 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ -\frac{5}{3} \end{bmatrix},$$

f achieves the smallest value

$$f(x^*) = -\frac{7}{3}.$$

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

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

Solution.

$$f(\boldsymbol{x}) = \underline{1}x_1^2 + \underline{1}x_2^2 + \underline{1}x_1x_2 + \underline{1}x_1 + \underline{3}x_2.$$

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

Solution.

Please see "P2.ipynb" for the code and Figure 9 for the contour plot of f.

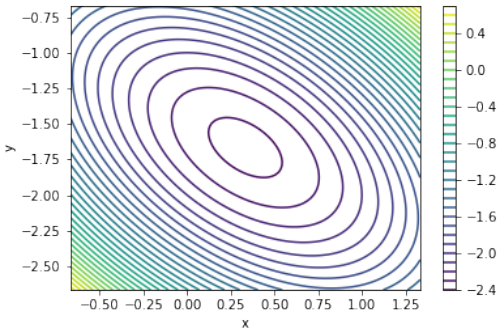

Figure 9: Contour plot of f around its minimizer.

The eigenvectors of H is:

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

and the eigenvalues are

$$\lambda_1 = 3 \text{ and } \lambda_2 = 1.$$

Since $\nabla f(x^*) = 0$, we have for any $x \in \mathbb{R}^2$:

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

For any $x \neq x^*$, we know $f(x) > f(x^*)$, i.e., $f(x) - f(x^*) > 0$, since H is a positive definite matrix. Moreover, the eigenvalue decomposition of H is

$$H = [v_1, v_2] \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} [v_1, v_2]^\top = V \Lambda V^\top.$$

Letting $y = V^\top(x - x^*)$, then for any $z > f(x^*)$, we have

$$f(x) = z \iff \frac{3y_1^2 + y_2^2}{2\,(z - f(x^*))} = 1 \iff \frac{y_1^2}{\frac{2}{3}(z - f(x^*))} + \frac{y_2^2}{2(z - f(x^*))} = 1.$$

Therefore, the level curves are ellipse centered at x^*, where the line $x^* + \lambda v_1$ is the minor axis and the line $x^* + \lambda v_2$ is the major axis. The major axis length is inverse proportional to $\sqrt{\lambda_2}$ and the minor axis length is inverse proportional to $\sqrt{\lambda_1}$.

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

Solution.

Please see "P2.ipynb" for the code and Figure 10 for the contour plot of f along with the first 4 steps of the gradient descent algorithm.

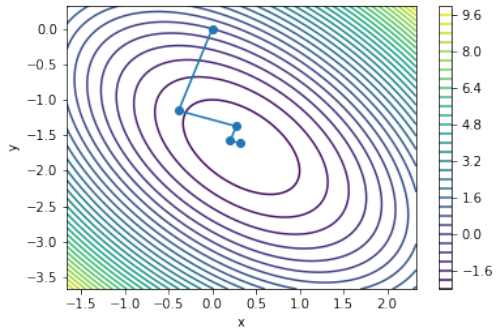

Figure 10: Contour plot of f with the first 4 steps of the gradient descent algorithm.

(e) On top of the contour plot, trace out the two steps of the conjugate gradient method starting at $\boldsymbol{x}_0 = \mathbf{0}$.

Solution.

Please see "P2.ipynb" for the code and Figure 11 for the contour plot of f along with the first 2 steps of the conjugate gradient method.

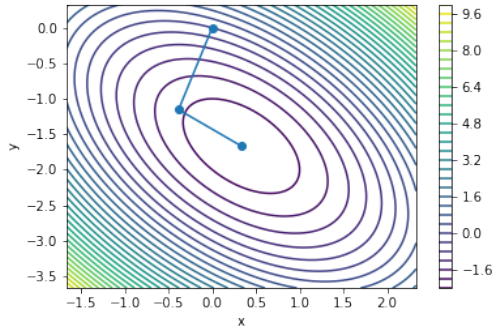

Figure 11: Contour plot of f with the first 2 steps of the conjugate gradient method.

3. (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 $\|\boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_k\|_2/\|\boldsymbol{b}\|_2$ 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 $\boldsymbol{H}$ and $\boldsymbol{b}$ in the file `hw05p3_data.mat` for a `tol` of 10^{-6}. Report the number of iterations needed for convergence, and for your solution $\widehat{\boldsymbol{x}}$ verify that $\|\boldsymbol{b} - \boldsymbol{H}\widehat{\boldsymbol{x}}\|_2/\|\boldsymbol{b}\|_2$ is within the specified tolerance.

Solution.
Please see "P3.ipynb" for the code. The number of iterations needed for convergence is 178 and $\frac{\|b-H\hat{x}\|_2}{\|b\|_2} = 9.067 \cdot 10^{-7} < 10^{-6}$.

(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 $\boldsymbol{H}$ and $\boldsymbol{b}$ 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 $\widehat{\boldsymbol{x}}$ verify that $\|\boldsymbol{b} - \boldsymbol{H}\widehat{\boldsymbol{x}}\|_2/\|\boldsymbol{b}\|_2$ is within the specified tolerance. Compare against the gradient descent algorithm in part (a).

Solution.
Please see "P3.ipynb" for the code. The number of iterations needed for convergence is 40 and $\frac{\|b-H\hat{x}\|_2}{\|b\|_2} = 7.618 \cdot 10^{-7} < 10^{-6}$. Compared with steepest descent, conjugate gradient method converges much faster given the same tolerance.

4. Let $\boldsymbol{A}$ be a banded tridiagonal matrix:

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

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

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

where $*$ signifies a non-zero term.

Solution.
If we assume that LU factorization is known to exist for all matrices, then the sparsity patterns above are guaranteed to yield the desired sparsity pattern in $\boldsymbol{A}$. If we don't assume that the LU factorization exist for all matrices, then the sparsity patterns are one indicator for the feasibility of such a factorization. Another indicatior for feasibility is that the factorization leaves more degrees of freedom ($4N-2$ unknowns) in the factorization than in the matrix ($3N-2$ unkonwns).

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

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

I will get you started:

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

Solution.

$u_1 = d_1$
for $k = 2, \ldots, N$
  $g_{k-1} = c_{k-1}$
  $\ell_{k-1} = f_{k-1}/u_{k-1}$
  $u_k = d_k - g_{k-1}\ell_{k-1}$
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 $\boldsymbol{Ax} = \boldsymbol{b}$ scale with N?

Solution.
Since each computation in the loop of the algorithm above takes constant time to compute, the complexity of the algorithm is $O(N)$. Given the LU factorization above, solving $\boldsymbol{Ax} = \boldsymbol{b}$ consists of solving $\boldsymbol{Ly} = \boldsymbol{b}$ and then solving $\boldsymbol{Ux} = \boldsymbol{y}$. Each of these subproblems require $O(N)$ sequential substitution operations therefore the whole solution process is also $O(N)$.

5. The Gram-Schmidt process is a method for orthonormalizing a set of vectors in an inner product space. The method works as follows: if $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_N\}$ 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 $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_N\}$ such that

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

using

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

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

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

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

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

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

Solution.

Please see "P5.ipynb" for the code.

$$\boldsymbol{q}_1 = \begin{bmatrix} 0.4472136 \\ 0.4472136 \\ -0.4472136 \\ -0.4472136 \\ -0.4472136 \end{bmatrix}, \quad \boldsymbol{q}_2 = \begin{bmatrix} 0.54772256 \\ 0.54772256 \\ 0.36514837 \\ 0.36514837 \\ 0.36514837 \end{bmatrix}, \quad \boldsymbol{q}_3 = \begin{bmatrix} 0.46291005 \\ -0.46291005 \\ 0.3086067 \\ -0.6172134 \\ 0.3086067 \end{bmatrix}.$$

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

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

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

Solution.

Please see "P5.ipynb" for the code.

$$\boldsymbol{R} = \begin{bmatrix} 2.23606798 & -0.4472136 & -0.4472136 \\ 0 & 2.19089023 & 0.36514837 \\ 0 & 0 & 2.1602469 \end{bmatrix}.$$

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

Solution.
The columns of $\boldsymbol{Q}$ are defined by the Gram-Schmidt algorithm, and the entries of $\boldsymbol{R}$ are defined as follows:

$$R[i,j] = \begin{cases} \langle \boldsymbol{a}_j, \boldsymbol{q}_i \rangle, & i < j, \\ \|\boldsymbol{w}_j\|_2, & i = j, \\ 0, & i > j. \end{cases}$$

(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 $\boldsymbol{A}$ implies that all of the entries along the diagonal of $\boldsymbol{R}$ will be nonzero?

Solution.
The determinant of a triangular matrix is simply the product of its diagonal elements. It is known that a matrix is invertible if and only if its determinant is nonzero. As a result, if an upper triangular matrix has only nonzero diagonal elements then the determinant is nonzero and the matrix is invertible. If an upper triangular matrix is invertible, it must also have nonzero determinant and therefore only has nonzero diagonal elements. Combining these two statements, we know that an upper triangular matrix is invertible if and only if none of the elements along the diagonal are zero. So long as the columns of $\boldsymbol{A}$ are linearly independent, we know that Gram-Schmidt will yield nonzero $\boldsymbol{w}_k$ vectors. Since the diagonal elements of $\boldsymbol{R}$ are simply the norms of these $\boldsymbol{w}_k$ vectors, we know that $\boldsymbol{A}$ having linearly independent columns implies $\boldsymbol{R}$ has only nonzero diagonal elements and is hence invertible.

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

$$\underset{\boldsymbol{x} \in \mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$$

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

Solution.
Notice that by construction $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q} = \mathbf{I}$. The solution to the least squares problem is:

$$\begin{aligned} \widehat{\boldsymbol{x}} &= \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} \\ &= \left(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{Q}\boldsymbol{R}\right)^{-1}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} \\ &= \left(\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R}\right)^{-1}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y} \\ &= \boldsymbol{R}^{-1}\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{y}. \end{aligned}$$