AMATH 584: Numerical Linear Algebra: Exercise 12.1
Sid Meka

Suppose A is a 202×202 matrix with $||A||_2 = 100$ and $||A||_F = 101$. Give the sharpest possible lower bound on the 2 norm condition number $\kappa(A)$.

Write the nonzero singular values of A in descending order as $\sigma_1, \sigma_2, \ldots, \sigma_{202}$ such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{202} \geq 0$. Recall $||A||_2 = \sigma_1$ and $||A||_F = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_{202}^2}$. So, now we know that $\sigma_1 = 100$ and $101 = \sqrt{\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_{202}^2}$.
The Frobenius norm allows us to state:

$$||A||_F^2 = \sigma_1^2 + \sigma_2^2 + \cdots + \sigma_{202}^2$$

Substituting $\sigma_1 = 100$ and $||A||_F = 101$, we get:

$$101^2 = 100^2 + \sigma_2^2 + \cdots + \sigma_{202}^2$$
$$101^2 - 100^2 = \sigma_2^2 + \sigma_3^2 + \cdots + \sigma_{202}^2$$
$$(101 - 100)(101 + 100) = \sigma_2^2 + \sigma_3^2 + \cdots + \sigma_{202}^2$$
$$201 = \sigma_2^2 + \sigma_3^2 + \cdots + \sigma_{202}^2$$
$$\sigma_2^2 + \sigma_3^2 + \cdots + \sigma_{202}^2 = 201$$

By the definition of a 2 norm condition number, we know that $\kappa(A) = \frac{\sigma_1}{\sigma_{202}}$. Remember that we have to find a bound on σ_{202}, which is the smallest singular value. This will help us determine the smallest possible value for the condition number $\kappa(A) = \frac{\sigma_1}{\sigma_{202}}$.

Now, we apply Cauchy-Schwarz Inequality

Applying Cauchy-Schwarz Inequality: We can use Cauchy-Schwarz Inequality to say that

$$\sigma_2^2 + \sigma_3^2 + \cdots + \sigma_{202}^2 \geq 201 \cdot \sigma_{202}^2$$

Solving this inequality:

$$201 \cdot \sigma_{202}^2 \leq 201$$
$$\sigma_{202}^2 \leq 1$$
$$\sigma_{202} \leq 1 \text{ as Singular Values, by definition, are nonnegative}$$

Putting this all together:

$$\kappa(A) = \frac{\sigma_1}{\sigma_{202}}$$
$$\kappa(A) = \frac{100}{\sigma_{202}}$$

Remember that we found when σ_{202} is as large as possible, we get the value 1 meaning that in order to find the smallest value of $\kappa(A)$, we get

$$\kappa(A) = \frac{100}{1}$$
$$\kappa(A) = 100$$

This means that the sharpest lower bound on the 2 norm condition number for $\kappa(A)$ is 100.

21.1. a. Find det(A) from 20.5

A from 20.5: $\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$

we know from 20.5 that:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

we also know that $A = LU$.

$$\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{bmatrix}}_{U}$$

Because $A = LU$, that means $\det(A) = \det(L)\det(U)$.

$$\det(L) = 1 \cdot 1 \cdot 1 \cdot 1 = 1$$

$$\det(U) = 2 \cdot 1 \cdot 2 \cdot 2 = 8$$

$$\det(L)\det(U) = 1 \cdot 8 = 8$$

Thus, $\det(A) = 8$.

b. Find det(A) from 21.3

A from 21.3: $\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$

We know from 21.3 that:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

we also know that $PA = LU$.

$$\begin{bmatrix} & 1 & & \\ & & & 1 \\ & 1 & & \\ 1 & & & \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \frac{3}{4} & 1 & & \\ \frac{1}{2} & -\frac{2}{7} & 1 & \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{bmatrix} \begin{bmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{bmatrix}$$

Because $PA = LU$, that means $\det(P)\det(A) = \det(L)\det(U)$.

To find $\det(P)$, we look for how many row swaps we can do on P in order to find the Identity Matrix.

Note that

$$P=\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

If we swap row 1 with row 4, row 2 with row 3, and the new row 3 with the new row 4, we get the 4×4 Identity matrix. Because 3 swaps were done, we have that $\det(P)=(-1)^3\det(I)$, which means $\det(P)=-1$.

$$\det(L)=1\cdot1\cdot1\cdot1=1$$

$$\det(U)=8\cdot\frac{7}{4}\cdot-\frac{6}{7}\cdot\frac{2}{3}=-\frac{96}{12}=-8$$

$$\det(P)\det(A)=\det(L)\det(U)$$

$$-\det(A)=-8$$

$$\det(A)=8$$

Thus, $\det(A)=8$.

AMATH 584: Numerical Linear Algebra: Exercise 21.1 Part c
Sid Meka

Describe how Gaussian elimination with partial pivoting can be used to find the determinant of a general square matrix.

When performing Gaussian elimination with partial pivoting on a square matrix A, the process can be interpreted as decomposing A into three matrices:

- P: a permutation matrix representing row swaps
- L: a lower triangular matrix with 1's on the diagonal
- U: an upper triangular matrix

So, we have the equation $PA = LU$ where P is an $n \times n$ permutation matrix, L is an $n \times n$ lower triangular matrix, and U is an $n \times n$ upper triangular matrix.

Steps with $PA = LU$ Decomposition:

1. We setup the decomposition as so:
   - Perform Gaussian Elimination with partial pivoting on A
   - Get the Permutation Matrix P
   - Fill in the lower triangular entries for L
   - Produce the upper triangular matrix U
2. Using some properties with regards to $PA = LU$: Since we know that $PA = LU$ in this decomposition, we can use some properties to state that
$$\begin{aligned} PA &= LU \\ \det(PA) &= \det(LU) \\ \det(P)\det(A) &= \det(L)\det(U) \\ \det(A) &= \frac{\det(L)\det(U)}{\det(P)} \end{aligned}$$
3. Determinants of P, L, and U:
   - $\det(P) = (-1)^d$ where d represents the number of row swaps.
   - $\det(L) = 1$ as L has 1's on its diagonal by definition of the decomposition we're working with.
   - The determinant of U is simply the product of its diagonal elements meaning that $\det(U) = \prod_{i=1}^{n} U_{ii}$.
4. Putting this altogether: Recall that
$$\det(A) = \frac{\det(L)\det(U)}{\det(P)}$$
   Substituting our values as found in the step before, we can state that
$$\det(A) = \frac{(1)\prod_{i=1}^{n} U_{ii}}{(-1)^d}$$

Putting this more concisely, we have that:

$$\det(A) = (-1)^d \prod_{i=1}^{n} U_{ii}$$

Thus, we have that:

$$\det(A) = (-1)^d \prod_{i=1}^{n} U_{ii}$$

for a general square matrix.

AMATH 584: Numerical Linear Algebra: Homework 5 A1
Sid Meka

The number $\frac{8}{7}$ obviously has no exact representation in any decimal floating point system ($\beta = 10$) with finite precision t. Is there a finite floating point system (i.e., some finite integer base β and precision t) in which this number does have an exact representation? Answer the same question for the irrational number π.

- For $\frac{8}{7}$, we can choose a base β that allows a finite precision t. One such instance is to have $\beta = 7$. In base 7, $\frac{8}{7} = 1.1_7$. To answer the question, yes, there exists a finite floating point system, such as $\beta = 7$ where we have a finite precision for $\frac{8}{7}$.
- The number π is an irrational number meaning that it has an infinite and non repeating decimal representation in any base. Remember, that π cannot be written in terms of any fraction where the numerator and denominator are both integers. This infinite and non repeating property means that π cannot be represented exactly in any finite floating point system regardless of what β and t are assuming β is an integer. The fact that π is an irrational number prevents any exact representation in a finite system regardless of what β and t are where β is an integer. To answer the question, no, there is no finite floating point system in which π has an exact representation.

Sid Meka

Numerical Linear Algebra

A2:

Part a:

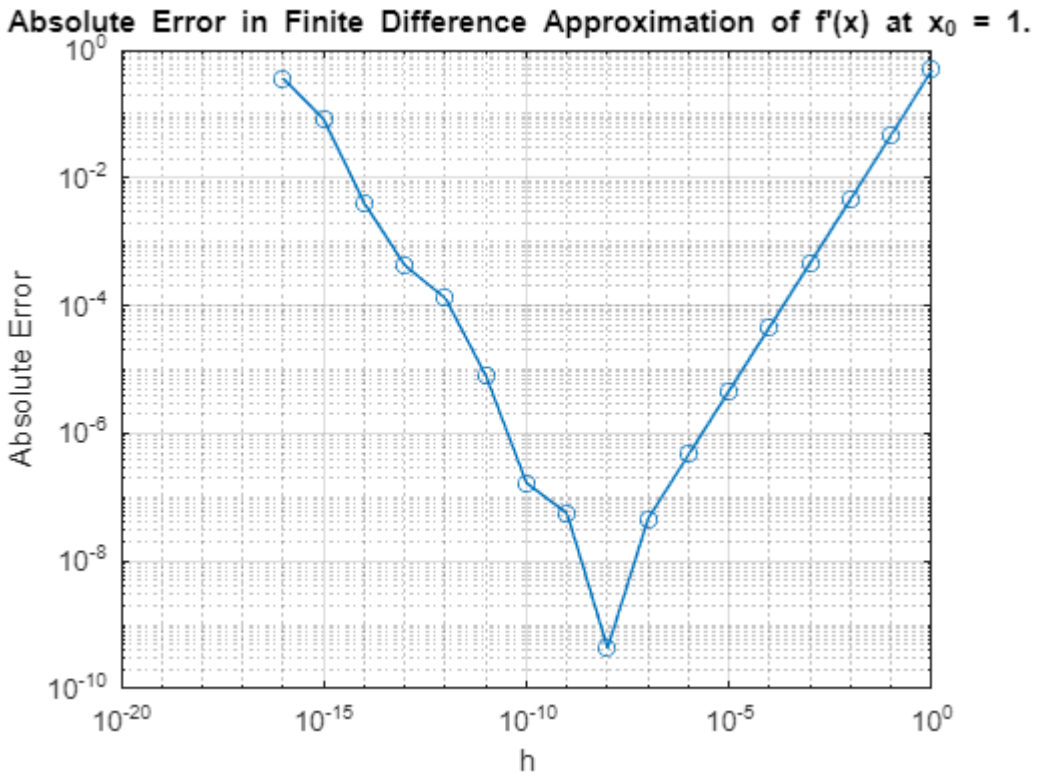
1. Approximation of $f'(x)$ with Finite Differences: For $f(x) = \sin(x)$, we are approximating $f'(x)$ at $x_0 = 1.2$ using: $f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}$. The exact derivative of $\sin(x)$ is $\cos(x)$, so at $x_0 = 1.2$, we know the exact value of the derivative is $\cos(1.2)$.
2. Expanding $f(x_0 + h)$ around x_0 using Taylor series gives us: $f(x_0 + h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + O(h^3)$. Substituting this into the finite difference approximation gives: $\frac{f(x_0+h) - f(x_0)}{h} = f'(x_0) + \frac{h}{2} f''(x_0) + O(h^2)$. Here, $f'(x_0) = \cos(1.2)$ and $f''(x) = -\sin(1.2)$. Thus, the error in the finite difference approximation is: $\left| f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h} \right| = \frac{h}{2} f''(x_0) + O(h^2)$. Furthermore, we can say $\left| f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h} \right| = O(h)$. We see here that the error decreases linearly with h.
3. Note on κ: We have $f'(x) = \cos(x)$ and $f''(x) = -\sin(x)$. So, $\kappa = \left| x \cdot \frac{-\sin(x)}{\cos(x)} \right|$. Evaluating our κ value at 1.2, we get $\kappa = 3.08$.
4. We thus see that the $O(h)$ error term implies that as h gets smaller, the error in the approximation also gets proportionally smaller. This suggests that we have well conditioning as the approximation improves predictably and smoothly as we progress with our Taylor Expansion. We thus see that our finite difference formula scales directly with h and reduces smoothly as $h \to 0$. Thus, we see that we have well conditioning. Small perturbations in h, with reasonable bounds, lead to proportionally small changes in the finite difference approximation.

So, we see well conditioning when evaluating $f'(x)$ at $x_0 = 1.2$.

Part b:

```
% Define the function and its true derivative at x0
f = @(x) sin(x);
x0 = 1.2;
true_derivative = cos(x0);

% Generate h values: 10^i for i from 0 to -16
i_values = 0:-1:-16;
h_values = 10.^i_values;

% Initialize array for storing errors
errors = zeros(size(h_values));

% Loop over each h to calculate the finite difference approximation and error
```

```
for j = 1:length(h_values)
    h = h_values(j);
    approx_derivative = (f(x0 + h) - f(x0)) / h;
    errors(j) = abs(true_derivative - approx_derivative);
end

% Plot absolute error vs. h on a log-log scale
figure;
loglog(h_values, errors, '-o');
xlabel('h');
ylabel('Absolute Error');
title('Absolute Error in Finite Difference Approximation of f''(x) at x_0 =
1.2');
grid on;
```



For larger values of h, the error decreases linearly as h becomes smaller. This confirms that the finite difference approximation is a first order approximation where the truncation error decreases proportionally with h. As h starts getting extremely small, we actually see an increase in error values. This is probably due to round off errors. So, all in all, I actually didn't expect this behavior in the plot.

Part c:

Suppose the only rounding errors made are in rounding $f(x_0 + h)$ and $f(x_0)$, so that the computed values are $f(x_0 + h)(1 + \varepsilon_1)$ and $f(x_0)(1 + \varepsilon_2)$, where $|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_{\text{machine}}$, then there will be a difference between the computed quotient and the true quotient

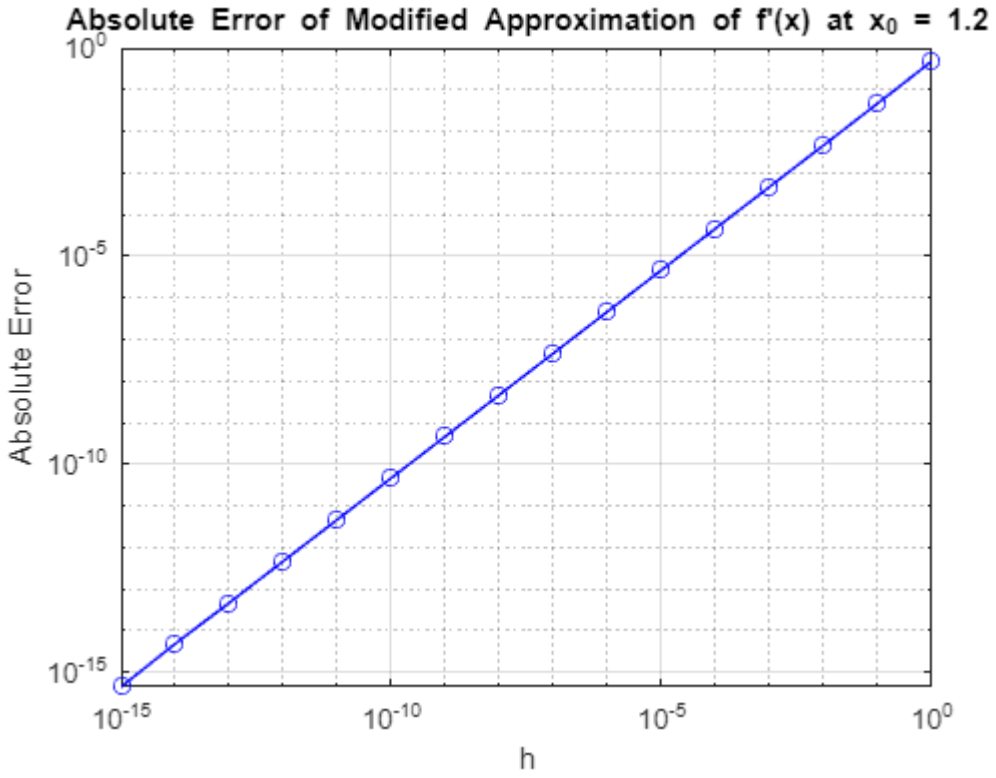
$$\frac{f(x_0+h)(1+\varepsilon_1)-f(x)(1+\varepsilon_2)}{h}=\frac{f(x_0+h)-f(x_0)}{h}+O\left(\frac{\varepsilon_{\text{machine}}}{h}\right)$$

Using this to explain my results in Part b:

The difference between the computed quotient and the true quotient can be explained by rounding errors. When h is relatively large, the term $O(\varepsilon_{\text{machine}})$ is very small and does not impact the accuracy of the approximation, which is dominated by the first term with an error of $O(h)$. However, as h gets significantly small, the term $O\left(\frac{\varepsilon_{\text{machine}}}{h}\right)$ becomes relatively large leading to a sharp increase in error for small h.

Part d:

```
% Define the function and its true derivative at x0
x0 = 1.2;
f_prime_true = cos(x0);

% Create values for h = 10^i where i ranges from 0 to -16
i_values = 0:-1:-16;
h_values = 10 .^ i_values;

% Compute the modified finite difference approximation of f'(x0) using the
new formula
finite_diff_approx = (2 * cos(x0 + h_values / 2) .* sin(h_values / 2)) ./
h_values;

% Compute the absolute error
absolute_errors = abs(f_prime_true - finite_diff_approx);

% Plot the absolute error vs. h on a log-log scale
figure;
loglog(h_values, absolute_errors, 'o-', 'Color', 'b');
xlabel('h');
ylabel('Absolute Error');
title('Absolute Error of Modified Approximation of f''(x) at x_0 = 1.2');
grid on;
```



I actually once again did not expect the plot to behave like this. I expected greater error for smaller h like we have seen before. We see better numerical stability. Furthermore, we observe that the error now decreases linearly with h throughout and breaks only at the end when we start approaching $\varepsilon_{\text{machine}}$ and the error approaches 0. This suggests that the formula being utilized might have properties that give improved numerical stability by reducing the sensitivity to rounding errors as h becomes significantly small.

Sid Meka

Numerical Linear Algebra

A3

Setup: Create Matrix A, Vector x, and Vector b

```
% Step 1: Create the matrix A
A = 2 * eye(100, 100);
b = ones(100);
b = tril(b);
A = A-b;
A(:, 100) = 1;
x = randn(100, 1);
b = A*x;
```

Part a: Computing the 2 norm condition number:

```
% Compute the 2-norm condition number of A
cond_A = cond(A);
disp(['2-norm condition number of A: ', num2str(cond_A)]);
```

```
2-norm condition number of A: 44.8023
```

This is well conditioned as our 2 norm number is less than 100.

Part b:

```
% Solve the linear system Ax = b using Gaussian elimination with partial
pivoting
x_ge = A \ b;

% Compute the relative 2-norm error between the computed solution x_ge and
the true solution x_true
relative_error_ge = norm(x - x_ge, 2) / norm(x, 2);
disp(['Relative 2-Norm Error (Gaussian Elimination): ',
num2str(relative_error_ge)]);
```

```
Relative 2-Norm Error (Gaussian Elimination): 0.64483
```

I do not trust this solution because the solution is pretty far from 0 as there is around a 64% to 68% Error whenever I run this code.

Part c:

```
% Solve Ax = b using QR decomposition
[Q, R] = qr(A, 0);
x_qr = R \ (Q' * b);

% Compute the relative 2-norm error between x_qr and the true solution x
relative_error_qr = norm(x - x_qr) / norm(x);
```

```
disp(['Relative 2-norm error for QR decomposition solution: ',
num2str(relative_error_qr)]);
```

```
Relative 2-norm error for QR decomposition solution: 1.96e-15
```

I do trust this solution as the error here is significantly small.

Part d:

```
function x_cp = gaussian_complete_pivoting(A, b)
    [n, ~] = size(A);
    Aug = [A b]; % Augment matrix A with vector b
    for k = 1:n-1
        % Complete pivoting
        [~, idx] = max(abs(Aug(k:n, k:n)), [], 'all', 'linear');
        [row, col] = ind2sub([n-k+1, n-k+1], idx);
        row = row + k - 1;
        col = col + k - 1;

        % Row interchange
        if row ~= k
            Aug([k, row], :) = Aug([row, k], :);
        end
        % Column interchange
        if col ~= k
            Aug(:, [k, col]) = Aug(:, [col, k]);
        end

        % Gaussian elimination
        for i = k+1:n
            factor = Aug(i, k) / Aug(k, k);
            Aug(i, k:n+1) = Aug(i, k:n+1) - factor * Aug(k, k:n+1);
        end
    end

    % Back substitution
    x_cp = zeros(n, 1);
    for i = n:-1:1
        x_cp(i) = (Aug(i, end) - Aug(i, 1:n) * x_cp) / Aug(i, i);
    end
end

% Solve using Gaussian elimination with complete pivoting
x_cp = gaussian_complete_pivoting(A, b);

% Compute the relative 2-norm error between x_cp and the true solution x
relative_error_cp = norm(x - x_cp) / norm(x);
disp(['Relative 2-norm error for complete pivoting solution: ',
num2str(relative_error_cp)]);
```

```
Relative 2-norm error for complete pivoting solution: 1.5249
```

I do not trust this solution as the error here is pretty large with respect to what we're looking for.