PS 4

1. We have

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 1 & 5 & -1 \\ -3 & 1 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 5 \\ 6 \\ 8 \end{bmatrix}$$

Since the columns of A are lin ind., $K = A^T A$ is pos. def. + invertible.

LSS is $x^* = (A^T A)^{-1} A^T b$

$$= \left(\begin{bmatrix} 1 & 0 & 1 & -3 \\ 2 & -2 & 5 & 1 \\ -1 & 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 1 & 5 & -1 \\ -3 & 1 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 0 & 1 & -3 \\ 2 & -2 & 5 & 1 \\ -1 & 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 5 \\ 6 \\ 8 \end{bmatrix}$$

$$= \begin{pmatrix} 11 & 4 & -5 \\ 4 & 34 & -12 \\ -5 & -12 & 12 \end{pmatrix}^{-1} \begin{pmatrix} -18 \\ 28 \\ 17 \end{pmatrix} = \boxed{\begin{pmatrix} -1 \\ 2 \\ 3 \end{pmatrix} = x^*}$$

Then our LSE $= \|Ax^* - b\|^2$

$Ax^* = \begin{pmatrix} 0 \\ 5 \\ 6 \\ 8 \end{pmatrix} = b$, so $\|Ax^* - b\|^2 = \boxed{0 = LSE}$

2. a) We want to approx $\int_a^b f(x)\,dx$ with only

2 interpolation points $x_0 = a$, $x_1 = b$

Our approx using trapezoid rule will be the following: $\dfrac{p_1(b) + p_1(a)}{2}(b-a)$

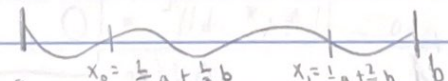
We find $p_1(x)$ by the following

$$p_1(x) = m(x-a) + b, \quad m = \frac{f(b) - f(a)}{b-a}, \quad b = f(a)$$

so, $\dfrac{p_1(b) + p_1(a)}{2}(b-a)$

$$= \frac{\frac{f(b)-f(a)}{b-a}(b-a) + f(a) + \frac{f(b)-f(a)}{b-a}\underbrace{(a-a)}_{0}}{2}(b-a)$$

$$= \boxed{\frac{f(b) + f(a)}{2}(b-a)}$$

b) 

We formulate $p_1(x)$ based off x_0 and x_1 but then use end points a, b for trapezoid rule

$$p_1(x) = m(x - x_0) + b, \quad m = \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \quad b = f(x_0)$$

$$m = \frac{f(x_1) - f(x_0)}{\left(\frac{1}{3}a + \frac{2}{3}b\right) - \left(\frac{2}{3}a + \frac{1}{3}b\right)} = \frac{f(x_1) - f(x_0)}{\frac{1}{3}(b-a)}$$

$$p_1(x) = \frac{f(x_1) - f(x_0)}{\frac{1}{3}(b-a)}(x - x_0) + f(x_0)$$

$$\frac{p_1(b) + p_1(a)}{2}(b-a)$$

$$= (b-a)\,\frac{\frac{f(x_1)-f(x_0)}{\frac{1}{3}(b-a)}\underbrace{\left(a - \left(\tfrac{2}{3}a + \tfrac{1}{3}b\right)\right)}_{\frac{1}{3}(a-b)} + f(x_0) + \frac{f(x_1)-f(x_0)}{\frac{1}{3}(b-a)}\underbrace{\left(b - \left(\tfrac{2}{3}a + \tfrac{1}{3}b\right)\right)}_{\frac{2}{3}(b-a)} + f(x_0)}{2}$$

$$= (b-a)\,\frac{f(x_0) - f(x_1) + f(x_0) + 2f(x_1) - 2f(x_0) + f(x_0)}{2}$$

$$= \boxed{\frac{f(x_0) + f(x_1)}{2}(b-a)}$$

c) $\int_0^1 e^x\,dx$

Trapezoid rule: $\frac{e^1 + e^0}{2} \approx 1.859$

Open rule: $\frac{e^{1/3} + e^{2/3}}{2} \approx 1.672$

Actual value ≈ 1.718

Trapezoid % error : $\left|\dfrac{1.859 - 1.718}{1.718}\right| \times 100$

$\approx \boxed{8.2\ \%\ \text{error}}$

Open rule % error : $\left|\dfrac{1.672 - 1.718}{1.718}\right| \times 100$

$\approx \boxed{2.7\ \%\ \text{error}}$

$\int_0^{\pi} \sin x \, dx$

Trapezoid rule : $\dfrac{\sin \pi + \sin 0}{2} = 0$ Actual: $\int_0^{\pi} \sin x \, dx = 2$

Open rule : $\dfrac{\sin \frac{\pi}{3} + \sin \frac{2\pi}{3}}{2} \approx 0.866$

Trapezoid % error : $\boxed{100\%\ \text{error}}$

Open rule % error : $\left|\dfrac{0.866 - 2}{2}\right| \times 100$

$\approx \boxed{56.7\%\ \text{error}}$

3. a) $r_i = y^{(i)} - f(x_1^{(i)}, x_2^{(i)})$

$$= y^{(i)} - \begin{pmatrix} 1 & x_1^{(i)} & x_2^{(i)} & x_1^{(i)} x_2^{(i)} \end{pmatrix} \begin{pmatrix} B_0^* \\ B_1^* \\ B_2^* \\ B_3^* \end{pmatrix}$$

$$r = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} - \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_1^{(1)} x_2^{(1)} \\ \vdots & x_1^{(2)} & x_2^{(2)} & x_1^{(2)} x_2^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_1^{(m)} x_2^{(m)} \end{pmatrix} \begin{pmatrix} B_0^* \\ B_1^* \\ B_2^* \\ B_3^* \end{pmatrix}$$

The least squares solution B^* are the solutions to $A^T A B^* = A^T y$

ACM/IDS 104 - Problem Set 4 - MATLAB Problems

Before writing your MATLAB code, it is always good practice to get rid of any leftover variables and figures from previous scripts.

```
clc; clear; close all;
```

Problem 3 (10 points) Application of Least Squares to Data Fitting

The dataset `carbig`, a built-in MATLAB dataset, contains various characteristics for $m = 406$ automobiles from the 1970s and 1980s. Let x_1, x_2 and y be the `Weight`, `Horsepower` and `MPG (Miles Per Gallon)` respectively. Let us assume the following theoretical model for the data:

$$y = f(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 \quad (\star)$$

Let $\beta^* = \left(\beta_0^*, \beta_1^*, \beta_2^*, \beta_3^*\right)^T$ denote the best fit to the data, i.e. the vector that minimizes the Euclidean norm of the residual vector $r = (r_1, \cdots, r_m)^T$, where:

$$r_i = y^{(i)} - f\left(x_1^{(i)}, x_2^{(i)}\right)$$

Part (a) (5 points)

Derive the system of normal equations on β^*. Do this in your written-up solutions.

Part (b) (5 points)

Find β^* by solving the normal system numerically and plot the scatter plot of the data $\left\{\left(x_1^{(i)}, x_2^{(i)}, y^{(i)}\right)\right\}, i = 1 \cdots m$, together with the fitted surface $y = f(x_1, x_2)$ given by $(\star)$.

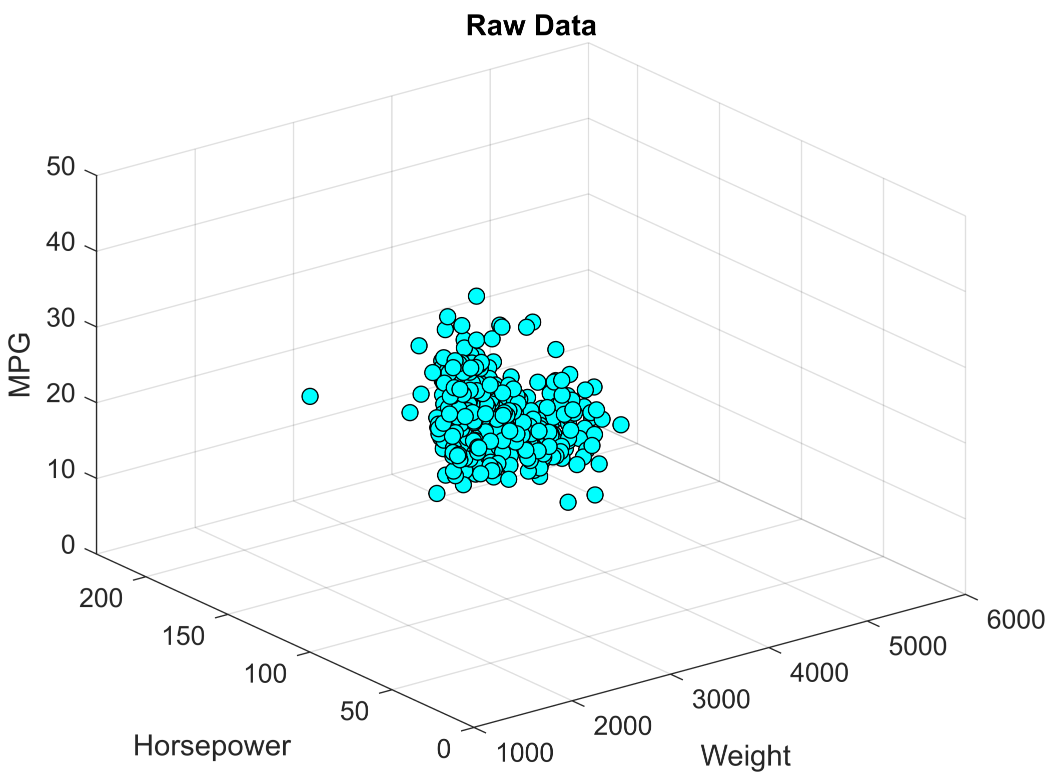
To start, let us load the dataset, define our variables, and perform some necessary cleanup as there exist `NaN` values. We do this for you :)

```
load carbig;
x1=Weight;
x2=Horsepower;
y=MPG;
clearvars -except x1 x2 y;
% Data cleaning
y=y(x1>0);
x2=x2(x1>0);
x1=x1(x1>0);

y=y(x2>0);
x1=x1(x2>0);
x2=x2(x2>0);

x1=x1(y>0);
x2=x2(y>0);
```

```
y=y(y>0);
```

Now, the dataset is ready for you to solve the normal system. Let us vizualize it. Drag the 3D plot around to get a better sense of the data.

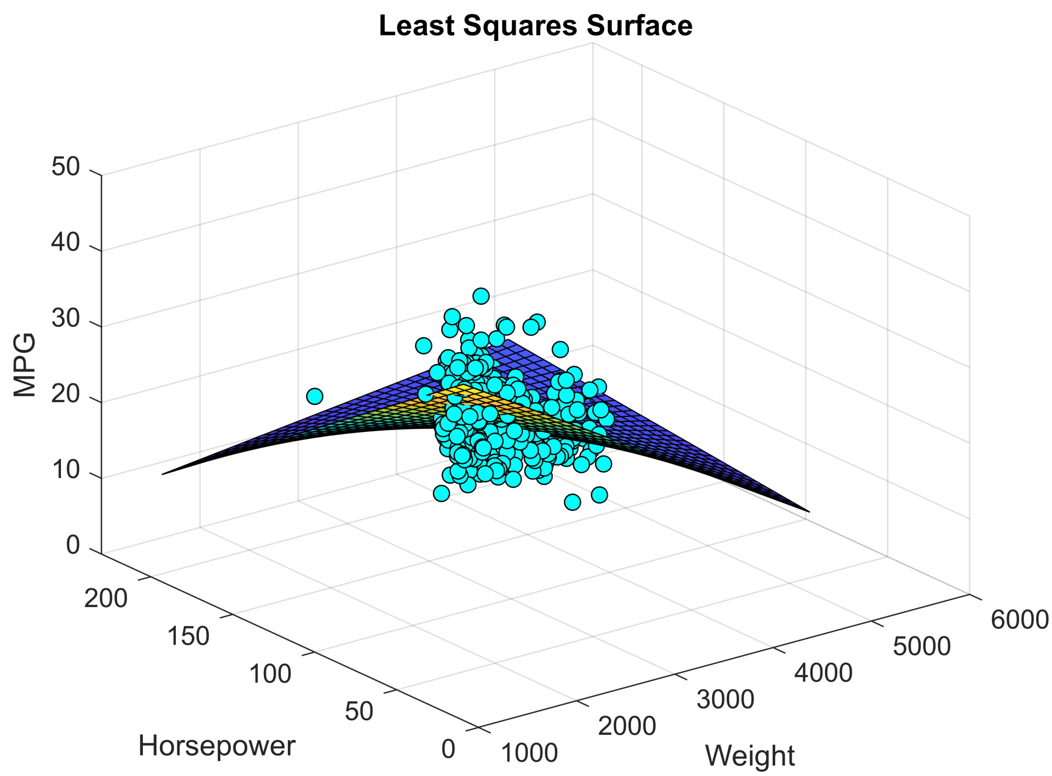
```
figure;
scatter3(x1,x2,y,'MarkerEdgeColor','k','MarkerFaceColor','c');
xlabel('Weight');
ylabel('Horsepower');
zlabel('MPG');
title('Raw Data');
```



Finally, let us find the best surface to fit the data. Remember, in MATLAB, when solving a system, backslash is your best friend. Use `scatter3()` and `fsurf()` when plotting.

```
%{
Build the matrix A
Solve for beta
A' * A * beta = A' * y
%}
A = [ones(length(y), 1), x1, x2, x1 .* x2];
beta = A\y;

%{
Plotting
%}
```

```
figure;
scatter3(x1,x2,y,'MarkerEdgeColor','k','MarkerFaceColor','c');
hold on
f = @(x, y) beta(1) + beta(2) * x + beta(3) * y + beta(4) * x .* y;
fsurf(f, [min(x1), max(x1), min(x2), max(x2)]);
hold off
xlabel('Weight');
ylabel('Horsepower');
zlabel('MPG');
title('Least Squares Surface')
```



Problem 4 (10 points) Polynomial Interpolation

Interpolating polynomials $p(x)$ are used for approximation of complex functions $g(x)$. Intuitively, the larger the degree of the interpolating polynomial, the more accurate the approximation $g(x) \approx p(x), x \in [a, b]$. Generally, this is not true (as you will see in this problem). High degree interpolating polynomials often behave badly, especially near the ends of the interval $[a, b]$. In practice, piecewise cubic splines are often used instead of high degree polynomials.

Suppose we want to approximate the function:

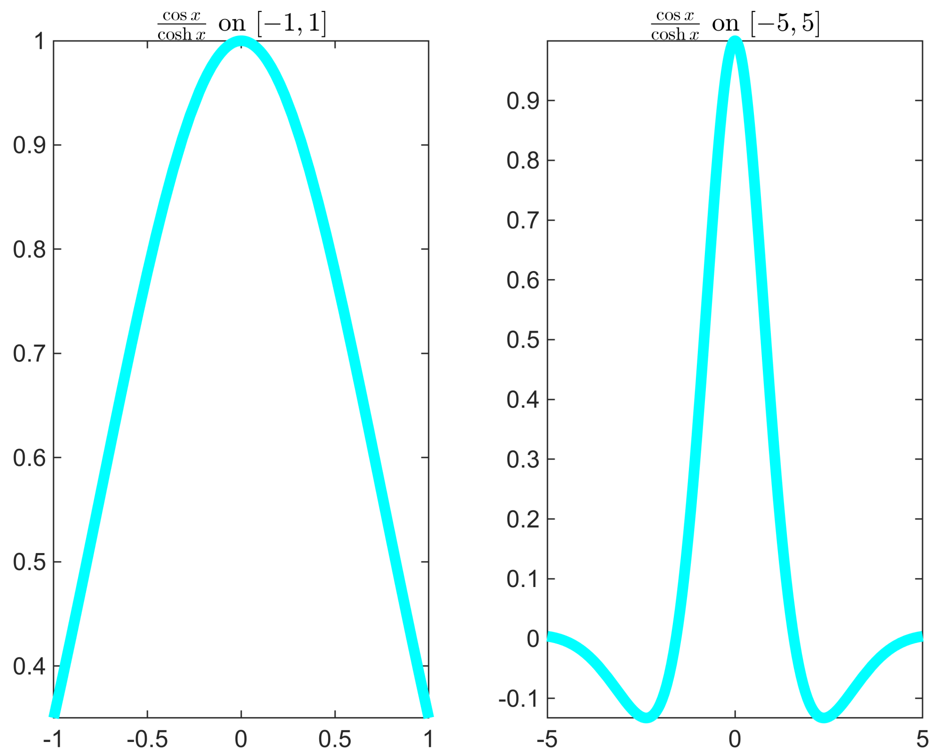
$$g(x) = \frac{\cos x}{\cosh x},\ x \in [-a, a]$$

using n points equally spaced between $-a$ and a. Find the interpolating polynomials and plot them versus the function $g(x)$ for $a = 1, 5$ and $n = 3, 5, 10, 15$. In your plot, show also the data points used for interpolation. The code should produce a single figure with 8 subplots.

Useful MATLAB functions for this problem:

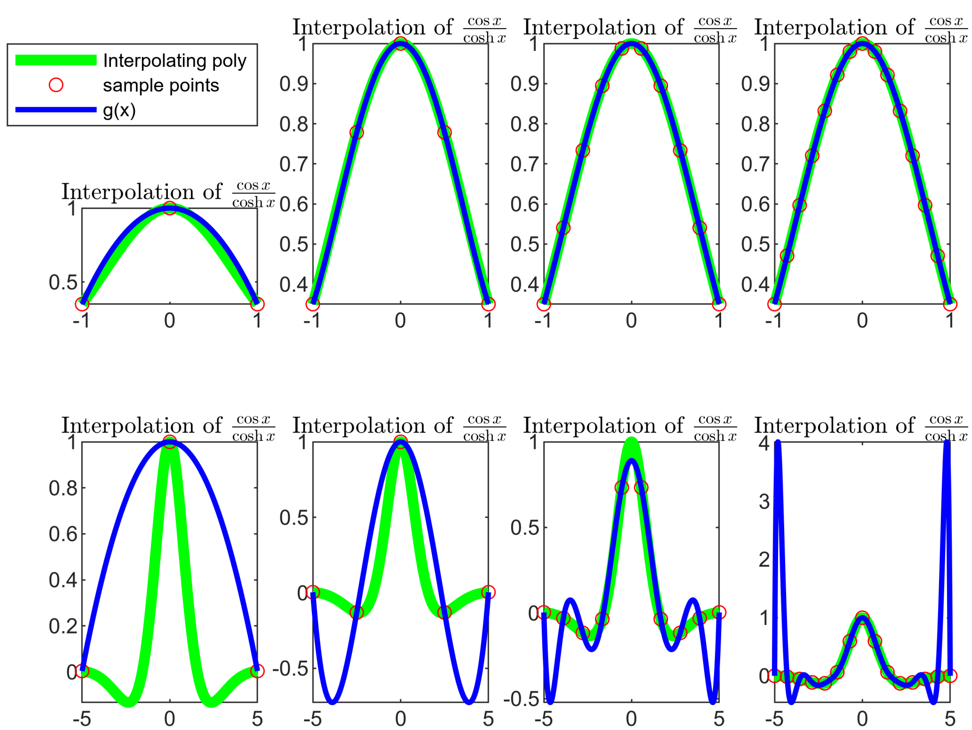
`linspace(), fplot(), polyval(), polyfit()`

```
%{
Let us start by doing some setup
%}
g = @(x) cos(x) ./ cosh(x); % you can define functions like this
a = [1, 5]; % setting interval values
n = [3, 5, 10, 15]; % setting the number of points
sub = 1; % setting the subplot index

%{
Let us see how g(x) looks like on both intervals
%}
figure;
subplot(1, 2, 1);
fplot(g, [-a(1), a(1)], "-c", "lineWidth", 4);
title("$\frac{\cos{x}}{\cosh{x}}$ on $[-1, 1]$","Interpreter","latex");
hold on
subplot(1, 2, 2);
fplot(g, [-a(2), a(2)], "-c", "lineWidth", 4);
title("$\frac{\cos{x}}{\cosh{x}}$ on $[-5, 5]$","Interpreter","latex");
```




```
%{
Complete the nested for-loops
%}
% figure('Position', [100, 100, 800, 800]);
figure;
for ival = a
    for degree = n-1
        %{
        Select degree+1 points in the interval
        Evaluate g(x) on these points
        Find the polynomial coefficients
        %}
        x_values = linspace(-ival, ival, degree + 1);
        y_values = g(x_values);
        coefficients = polyfit(x_values, y_values, degree);

        %{
        PLOTTING
        Plot g(x), the sampled points and interpolating polynomial
        Please use different colors and linestyles
        %}
        subplot(length(a), length(n), sub);
        sub = sub + 1;

        % Plot g(x)
```

```
        fplot(g, [-ival, ival], "-g", "lineWidth", 4);
        hold on;

        % Plot the sampled points
        plot(x_values, y_values, 'ro', 'MarkerSize', 5);

        % Plot the interpolating polynomial
        inter_x = linspace(-ival, ival, 1000); % Create a ton of x-values to give
illusion of cont. polynomial
        inter_y = polyval(coefficients, inter_x);
        plot(inter_x, inter_y, 'b', 'LineWidth', 2);

        title('Interpolation of $\frac{\cos{x}}{\cosh{x}}$', 'Interpreter',
'latex');
        if (sub == 2)
            legend('Interpolating poly', 'sample points', 'g(x)', 'Location',
'northoutside');
        end
    end
end
```



Problem 5 (10 points) Stability of the Gram-Schmidt Algorithm

The classical Gram-Schmidt algorithm is numerically unstable. This means that, when implemented on a computer, the round-off errors can cause the output vectors to be significantly non-orthogonal. To explore the issue, perform the following computations for each $n = 10, 20, 30, \cdots, 100$:

1. Create the Hilbert matrix H_n of size n (using `hilb(n)`) and consider the columns $h_1, \cdots, h_n$ as a basis of $\mathbb{R}^n$. The matrix H_n is non-singular, and thus its columns indeed form a basis, but it is very close to singular (i.e. its columns are close to being linearly dependent), and this leads to numerical problems.
2. Implement the basic Gram-Schmidt algorithm to construct an orthogonal basis $\{v_1, \cdots, v_n\}$ of $\mathbb{R}^n$ from $h_1, \cdots, h_n$. Please don't use any advanced built-in function for orthogonalization (such as `orthog()`) just basic matrix operations. At the end of the process normalize your vectors so that the basis is orthonormal.
3. If vectors $v_1, \cdots, v_n$ obtained in (2) are orthonormal, then $V = [v_1, \cdots, v_n]$ must be orthogonal. As a measure of orthogonality, compute the infinite norm $\delta_V(n) = ||I_n - V^T V||_\infty$, which is a matrix norm (use `norm(A, Inf)`). The closer $\delta_V(n)$ is to zero, the closer the columns of V are to being orthogonal.
4. Repeat (2) and (3) to construct an orthonormal basis $\{u_1, \cdots, u_n\}$ using the modified Gram-Schmidt algorithm (which is numerically more stable) described in lecture 9 (page 46), and compute $\delta_U(n)$.
5. To compare the basic and modified Gram-Schmidt algorithms, plot $\delta_V(n)$ and $\delta_U(n)$ versus n.

```
%{
Complete the following for-loop following (1)-(4)
%}
delta_v = zeros(10, 1);
delta_u = zeros(10, 1);
for n=10:10:100
    % Create the Hilbert matrix
    H = hilb(n);

    % ORIGINAL GRAM-SCHMIDT ALGO
    % Initialize an empty matrix for orthonormalized vectors
    V = zeros(n);
    V(:, 1) = H(:, 1) / norm(H(:, 1));
    for i = 2:n
        % Extract w_i, subtract projection of w_i onto current orthog. basis
        v_i = H(:, i);
        for j = 1:i-1
            v_j = V(:, j);
            v_i = v_i - (dot(H(:, j + 1), v_j) / power(norm(v_j), 2)) * v_j;
        end
        V(:, i) = v_i / norm(v_i);
    end
    % Calculate infinite norm
    infinite_norm1 = norm(eye(n) - V' * V, Inf);
    delta_v(n/10) = infinite_norm1;

    % MODIFIED GRAM-SCHMIDT ALGO
    U = zeros(n);
    for i = 1:n
        u_i = H(:, i) / norm(H(:, i));
        % Insert vec into orthonormal basis
```

```
        U(:, i) = u_i;
        % Modify w_(i+1) to w_n so they are perp. to u_i
        for j = i+1:n
            H(:, j) = H(:, j) - dot(H(:, j), u_i) * u_i;
        end
    end
    % Calculate infinite norm
    infinite_norm2 = norm(eye(n) - U' * U, Inf);
    delta_u(n/10) = infinite_norm2;
end

%{
Visualize the comparison as specified in (5)
%}
figure;
plot(10:10:100, delta_v, '-or');
grid on;
hold on;
plot(10:10:100, delta_u, '-*b');
hold off;
title('Gram-Schmidt Process Stability');
legend('delta_v', 'delta_u', 'Location', 'northwest');
ylabel('delta_n');
xlabel('n');
```