ACM 104 **November 7, 2023**

Problem Set 4

Problem 1. (10 points) Least Squares Solution

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

Putting A in reduced row echelon form:

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 1 & 5 & -1 \\ -3 & 1 & 1 \end{bmatrix} \xhookrightarrow{-r_1+r_3 \to r_3} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 3 & 0 \\ -3 & 1 & 1 \end{bmatrix} \xhookrightarrow{3r_1+r_4 \to r_4} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 3 & 0 \\ 0 & 7 & -2 \end{bmatrix} \xhookrightarrow{r_4+r_3 \to r_3} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 10 & -2 \\ 0 & 7 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 10 & -2 \\ 0 & 7 & -2 \end{bmatrix} \xhookrightarrow{5r_2+r_3 \to r_3} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 0 & 13 \\ 0 & 7 & -2 \end{bmatrix} \xhookrightarrow{r_3/13 \to r_3} \begin{bmatrix} 1 & 2 & -1 \\ 0 & -2 & 3 \\ 0 & 0 & 1 \\ 0 & 7 & -2 \end{bmatrix} \begin{array}{c} -3r_3 + r_2 \to r_2 \\ r_3 + r_1 \to r_1 \\ 2r_3 + r_4 \to r_4 \end{array} \begin{bmatrix} 1 & 2 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \\ 0 & 7 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \\ 0 & 7 & 0 \end{bmatrix} \xhookrightarrow{-r_2/2 \to r_2} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 7 & 0 \end{bmatrix} \begin{array}{c} -2r_2 + r_1 \to r_1 \\ -7r_2 + r_4 \to r_4 \end{array} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Because the reduced row echelon form of A has 3 pivots, then the columns of A are linearly independent. Hence, A has full rank. Therefore $K = A^T A$ is invertible and positive definite.

$$K = A^T A = \begin{bmatrix} 11 & 4 & -5 \\ 4 & 34 & -12 \\ -5 & -12 & 12 \end{bmatrix} \qquad A^T b = \begin{bmatrix} -18 \\ 28 \\ 17 \end{bmatrix}$$

Now solving the system $Kx^* = A^T b$ with matlab (refer to code attached). We get that the least square solution is:

$$x^* = \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix}$$

So we have:

$$Ax^* = \begin{bmatrix} 0 \\ 5 \\ 6 \\ 8 \end{bmatrix} = b \Rightarrow ||Ax - b||^2 = 0$$

Therefore the least squares error is zero.

Problem 2. (10 points) Interpolation for Integration

part (a)
From the question we have that $n = 1$. Thus we have

$$p_n(x) = f(x_0)L_1(x) + f(x_1)L_2(x)$$

$$p_n(x) = f(x_0) \cdot \left(\frac{x - x_1}{x_0 - x_1}\right) + f(x_1) \cdot \left(\frac{x - x_0}{x_1 - x_0}\right)$$

Integrating we have

$$\int_a^b p_n(x)\ dx = \frac{f(x_0)}{x_0 - x_1}\int_a^b x\ dx - \frac{f(x_0)}{x_0 - x_1}\int_a^b x_1\ dx + \frac{f(x_1)}{x_1 - x_0}\int_a^b x\ dx - \frac{f(x_1)}{x_1 - x_0}\int_a^b x_0\ dx$$

$$\int_a^b p_n(x)\ dx = \frac{1}{2}\frac{f(x_0)}{x_0 - x_1}(b^2 - a^2) - \frac{f(x_0)x_1}{x_0 - x_1}(b - a) + \frac{1}{2}\frac{f(x_1)}{x_1 - x_0}(b^2 - a^2) - \frac{f(x_1) \cdot x_0}{x_1 - x_0}(b - a)$$

Substituting for $x_0 = a$ and $x_1 = b$:

$$\int_a^b p_n(x)\ dx = \frac{1}{2}\frac{f(a)}{a - b}(b^2 - a^2) - \frac{f(a)b}{a - b}(b - a) + \frac{1}{2}\frac{f(b)}{b - a}(b^2 - a^2) - \frac{f(b) \cdot a}{b - a}(b - a)$$

$$\int_a^b p_n(x)\ dx = -\frac{1}{2}f(a) \cdot (b + a) + f(a) \cdot b + \frac{1}{2}f(b) \cdot (b + a) - f(b) \cdot a$$

$$\int_a^b p_n(x)\ dx = f(a)\left(\frac{-1}{2}(b + a) + b\right) + f(b)\left(\frac{1}{2}(b + a) - a\right) = \frac{1}{2} \cdot (b - a) \cdot (f(a) + f(b))$$

part (b)
From part a we have:

$$p_n(x) = f(x_0) \cdot \left(\frac{x - x_1}{x_0 - x_1}\right) + f(x_1) \cdot \left(\frac{x - x_0}{x_1 - x_0}\right)$$

Also from part A we have:

$$\int_a^b p_n(x)\ dx = \frac{1}{2}\frac{f(x_0)}{x_0 - x_1}(b^2 - a^2) - \frac{f(x_0)x_1}{x_0 - x_1}(b - a) + \frac{1}{2}\frac{f(x_1)}{x_1 - x_0}(b^2 - a^2) - \frac{f(x_1) \cdot x_0}{x_1 - x_0}(b - a)$$

Since $x_0 = a + \frac{1}{3}(b - a)$ and $x_1 = a + \frac{2}{3}(b - a)$. Then we have $x_0 - x_1 = -\frac{1}{3}(b - a)$ and $x_1 - x_0 = \frac{1}{3}(b - a)$.
Substituting for $x_0 - x_1$ and $x_1 - x_0$ we have:

$$\int_a^b p_n(x)\ dx = -\frac{3}{2}\frac{f(x_0)}{b - a}(b^2 - a^2) + 3\frac{f(x_0)x_1}{b - a}(b - a) + \frac{3}{2}\frac{f(x_1)}{b - a}(b^2 - a^2) - 3\frac{f(x_1) \cdot x_0}{b - a}(b - a)$$

$$\int_a^b p_n(x)\ dx = -\frac{3}{2}f(x_0)(b+a) + 3f(x_0) \cdot x_1 + \frac{3}{2}f(x_1)(b+a) - 3f(x_1) \cdot x_0 = f(x_0)\left(-\frac{3}{2}(b + a) + 3x_1\right) + f(x_1)\left(\frac{3}{2}(b + a) - 3x_0\right)$$

Substituting for x_0 and x_1 we have:

$$\int_a^b p_n(x)\ dx = f(x_0)\left(-\frac{3}{2}(b + a) + 3a + 2(b - a)\right) + f(x_1)\left(\frac{3}{2}(b + a) - 3a - (b - a)\right) = \frac{1}{2}(b-a)f(x_0) + \frac{1}{2}(b-a)f(x_1)$$

$$\int_a^b p_n(x)\ dx = \frac{1}{2}(b - a)\left(f(a + \frac{1}{3}(b - a)) + f(a + \frac{2}{3}(b - a))\right)$$

part (c)
First integral:
Actual value:

$$\int_0^1 e^x \, dx = e - 1 \approx 1.71828$$

Using the Trapezoid Rule:

$$\int_0^1 e^x \, dx \approx \frac{1}{2} \cdot (1-0) \cdot \left(e^0 + e^1\right) = \frac{1}{2}(e+1) \approx 1.85914$$

Using the Open Rule:

$$\int_0^1 e^x \, dx \approx \frac{1}{2} \cdot (1-0) \cdot \left(e^{1/3} + e^{2/3}\right) = \frac{1}{2}\left(e^{1/3} + e^{2/3}\right) \approx 1.67167$$

From the values above we see that the open rule yields a more accurate approximation for $\int_0^1 e^x \, dx$ than the trapezoid rule does.

Second integral:
Actual value:

$$\int_0^\pi sin(x) \, dx = 2.0$$

Using the Trapezoid Rule:

$$\int_0^\pi sin(x) \, dx \approx \frac{1}{2} \cdot (\pi - 0) \cdot (sin(0) + sin(\pi)) = 0$$

Using the Open Rule:

$$\int_0^\pi sin(x) \, dx \approx \frac{1}{2} \cdot (\pi - 0) \cdot (sin(\pi/3) + sin(2\pi/3)) = \frac{\sqrt{3}\pi}{2} \approx 2.7207$$

From the values above we see that the open rule yields a more accurate approximation for $\int_0^\pi sin(x) \, dx$ than the trapezoid rule does.

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

part (a)
Using the equations given in the problem we have:

$$\vec{r} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} - \begin{bmatrix} \beta_0^* 1 & \beta_1^* x_1^{(1)} & \beta_2^* x_2^{(1)} & \beta_3^* x_1^{(1)} x_2^{(1)} \\ \beta_0^* 1 & \beta_1^* x_1^{(2)} & \beta_2^* x_2^{(2)} & \beta_3^* x_1^{(2)} x_2^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_0^* 1 & \beta_1^* x_1^{(m)} & \beta_2^* x_2^{(m)} & \beta_3^* x_1^{(m)} x_2^{(m)} \end{bmatrix}$$

Rewriting we have:

$$\vec{r} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} - \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_1^{(1)} x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_1^{(2)} x_2^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_1^{(m)} x_2^{(m)} \end{bmatrix} \begin{bmatrix} \beta_0^* \\ \beta_1^* \\ \beta_2^* \\ \beta_3^* \end{bmatrix}$$

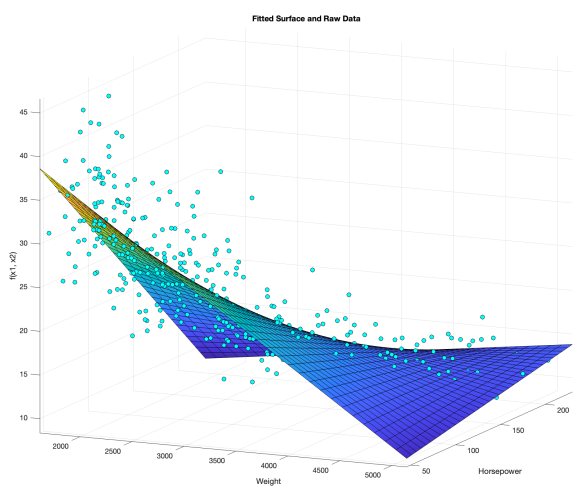
Let $\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$ and $A = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_1^{(1)} x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_1^{(2)} x_2^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_1^{(m)} x_2^{(m)} \end{bmatrix}$, and $\beta^* = \begin{bmatrix} \beta_0^* \\ \beta_1^* \\ \beta_2^* \\ \beta_3^* \end{bmatrix}$

In other words we are trying to solve the system $A\beta^* = \vec{y}$. Therefore the system of normal equations on β^* is the following:

$$A^T A \beta^* = A^T \vec{y}$$

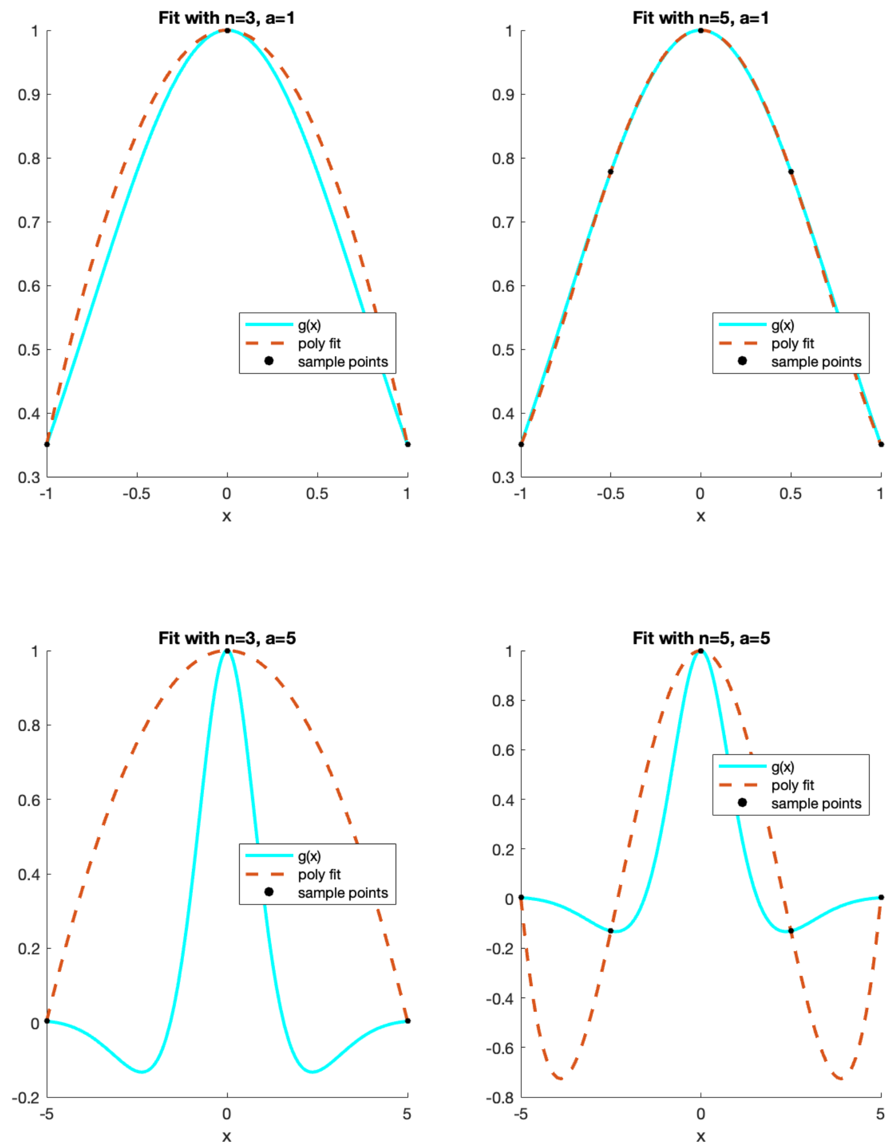
where $\vec{y}, A$, and β^* are as defined above.

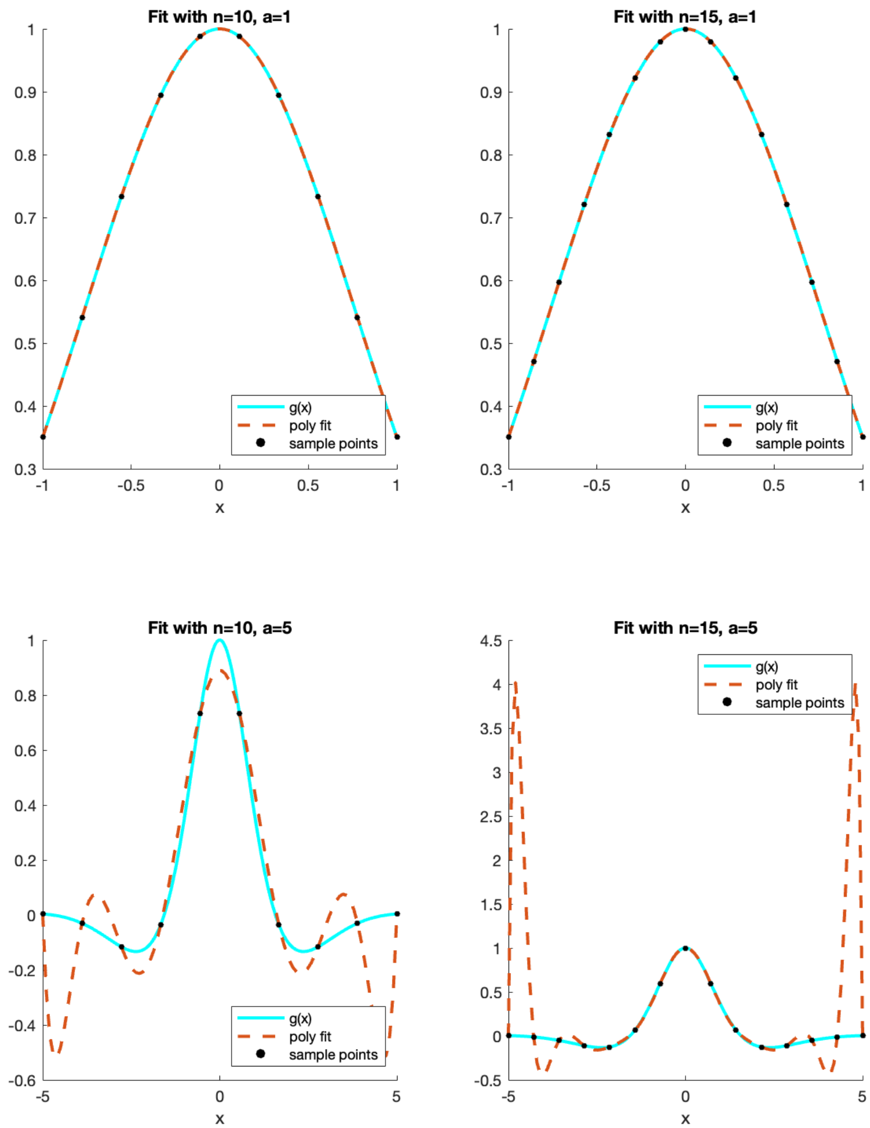
part (b)
See code attached. Plot is shown below.



Problem 4.(10 points) Polynomial Interpolation

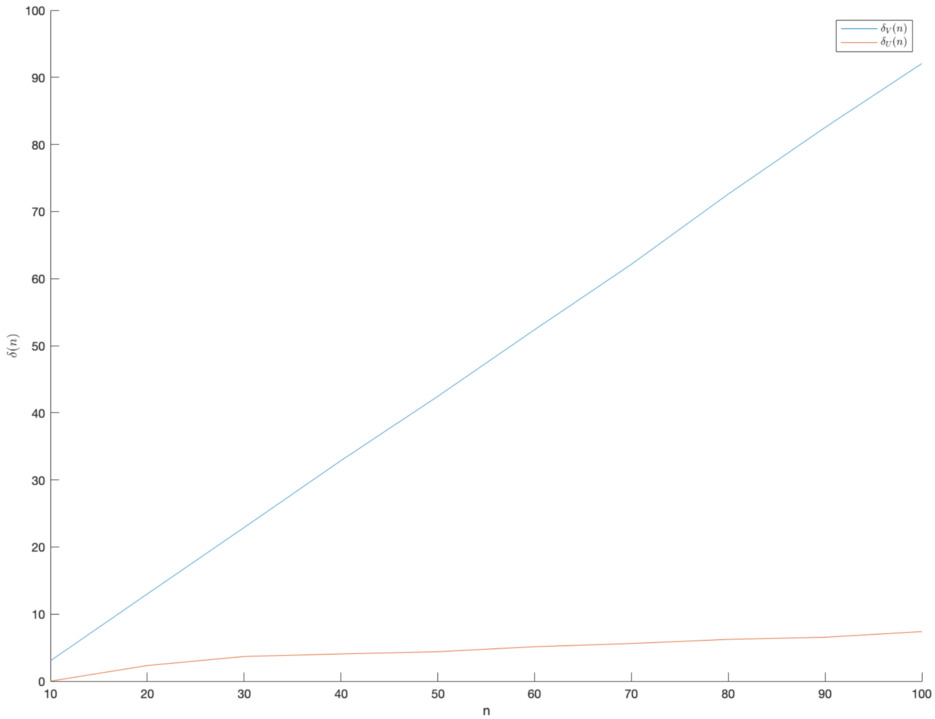
See code attached. Plots are shown below.




Fit with n=10, a=1
Fit with n=15, a=1
Fit with n=10, a=5
Fit with n=15, a=5
g(x)
poly fit
sample points
x

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

See code attached. Plot is shown below.



```
A = [1 2 -1; 0 -2 3; 1 5 -1; -3 1 1]
```

```
A = 4×3
     1     2    -1
     0    -2     3
     1     5    -1
    -3     1     1
```

```
b = [0; 5; 6; 8]
```

```
b = 4×1
     0
     5
     6
     8
```

```
rref(A)
```

```
ans = 4×3
     1     0     0
     0     1     0
     0     0     1
     0     0     0
```

```
K = A' * A
```

```
K = 3×3
    11     4    -5
     4    34   -12
    -5   -12    12
```

```
f = A' * b
```

```
f = 3×1
   -18
    28
    17
```

```
x = K \ f
```

```
x = 3×1
   -1.0000
    2.0000
    3.0000
```

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 $(\star)$.

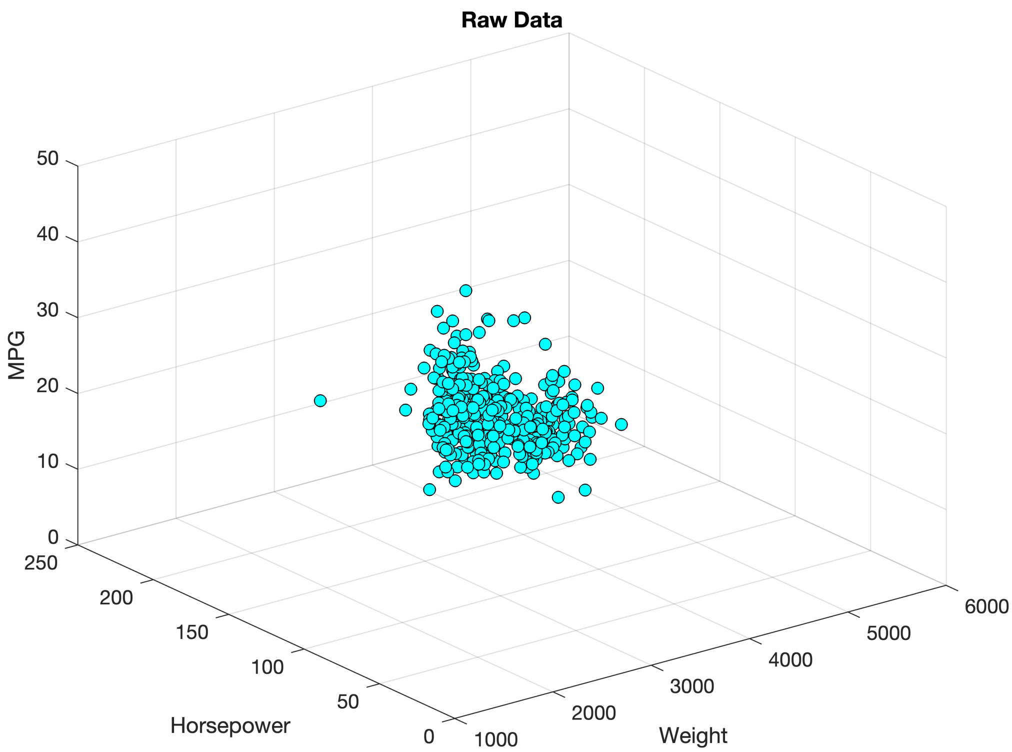
To start, let us load the dataset, define our variables, and perform some necessary cleanup as there exist **NaN** values. We do this for you :)

```
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');
```



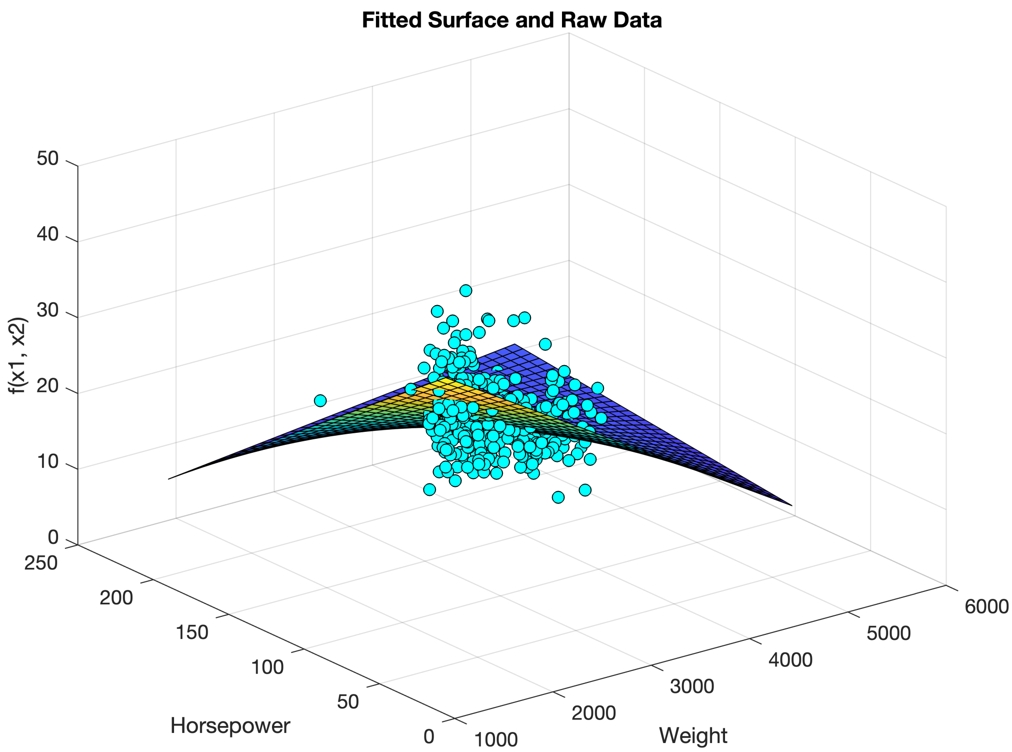
Finally, let us find the best surface to fit the data. Remember, in MATLAB, when solving a system, backslash is your best friend. Use **`scatter3()`** and **`fsurf()`** when plotting.

```
%{
Build the matrix A
Solve for beta
%}
% A is an m by 4 matrix
m = size(y, 1);
A = zeros(m , 4);
for i=1:m
    A(i, :) = [1 x1(i, 1) x2(i,1) x1(i, 1)*x2(i,1)];
end
```

```
K = A' * A;
f = A' * y;

% solving for beta
beta = K \ f;
y_pred = A * beta;

%{
Plotting
%}
figure;
scatter3(x1,x2,y,'MarkerEdgeColor','k','MarkerFaceColor','c');
hold on;
xlabel('Weight');
ylabel('Horsepower');
zlabel('f(x1, x2)');
[umin, ~] = min(x1);
[umax, ~] = max(x1);
[vmin, ~] = min(x2);
[vmax, ~] = max(x2);
func = @(x1,x2) [1, x1, x2, x1*x2]*beta;
fsurf(func, [umin umax vmin vmax])
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly vectorize
your function to return an output with the same size and shape as the input arguments.
```

```
hold off;
title('Fitted Surface and Raw Data');
```



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.

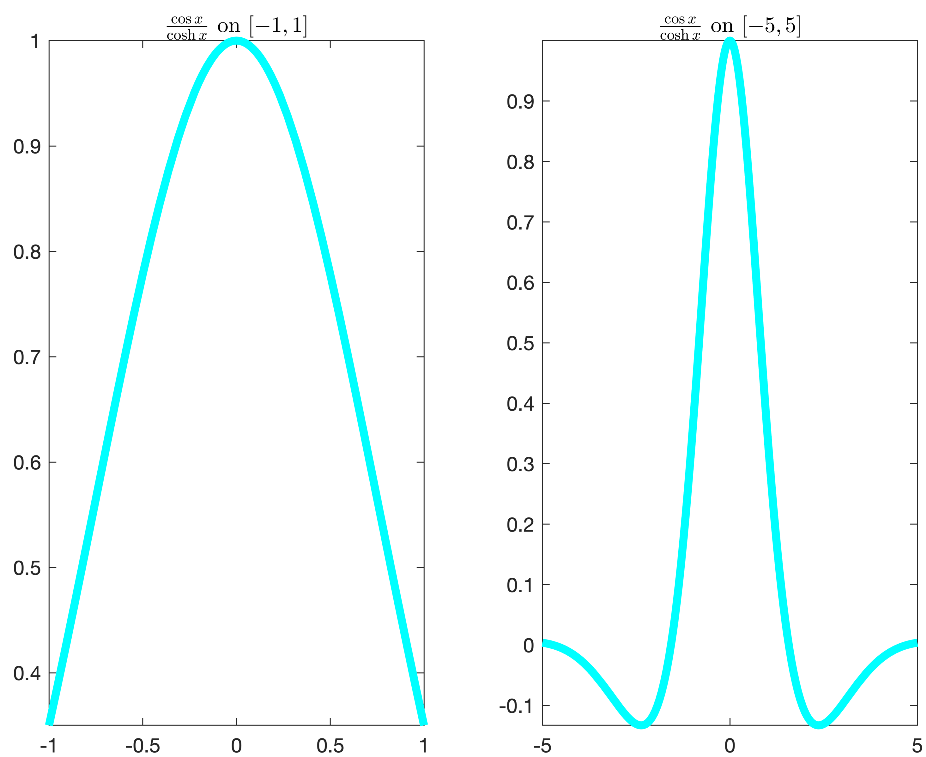
Useful MATLAB functions for this problem:

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

```

```

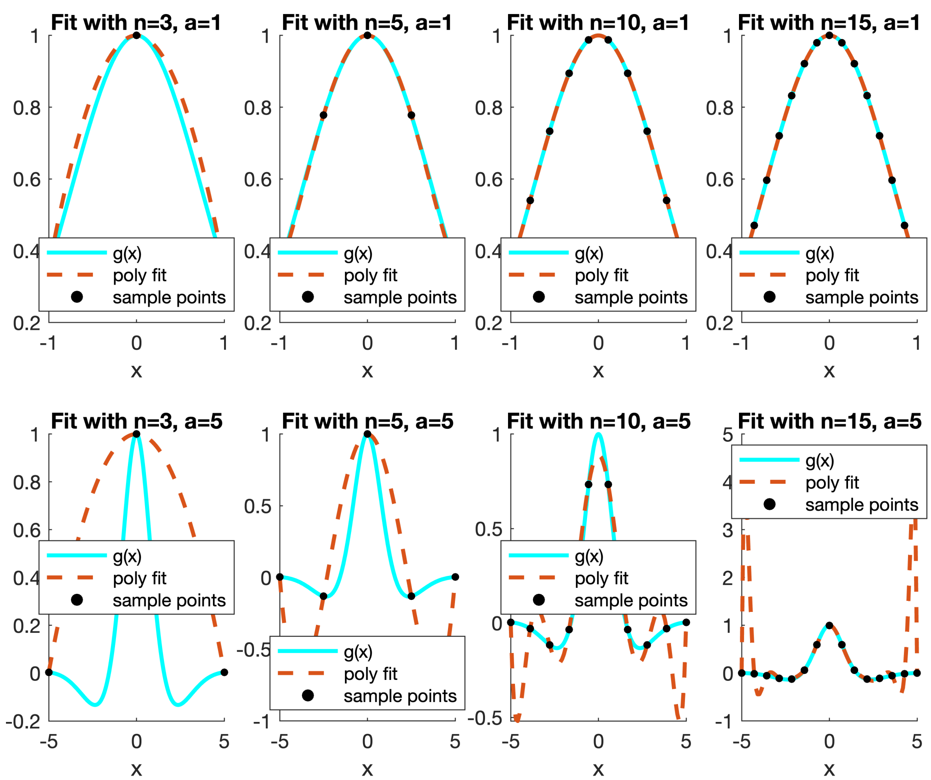
```
%{Let us start by doing some setup
%}
```

```
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
%}
plot_pos = 1;
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
    %}
    points = linspace(-ival, ival, degree + 1);
    g_eval = g(points);
    coeffs = polyfit(points, g_eval, degree);


    %{
    PLOTTING
    Plot g(x), the sampled points and interpolating polynomial
    Please use different colors and linestyles
    %}
    subplot(2,4, plot_pos);
    hold on
    fplot(g, [-ival, ival], "-c", "lineWidth", 2);
    title(compose("Fit with n=%d, a=%d", degree+1, ival));
    xlabel("x");
    more_points = linspace(-ival, ival, 100);
    plot(more_points, polyval(coeffs, more_points), "--", "lineWidth", 2);
    scatter(points, g_eval, 15, 'k', "filled");
    legend("g(x)", "poly fit", "sample points", "location", "best");
    plot_pos = plot_pos + 1;

  end
end
```



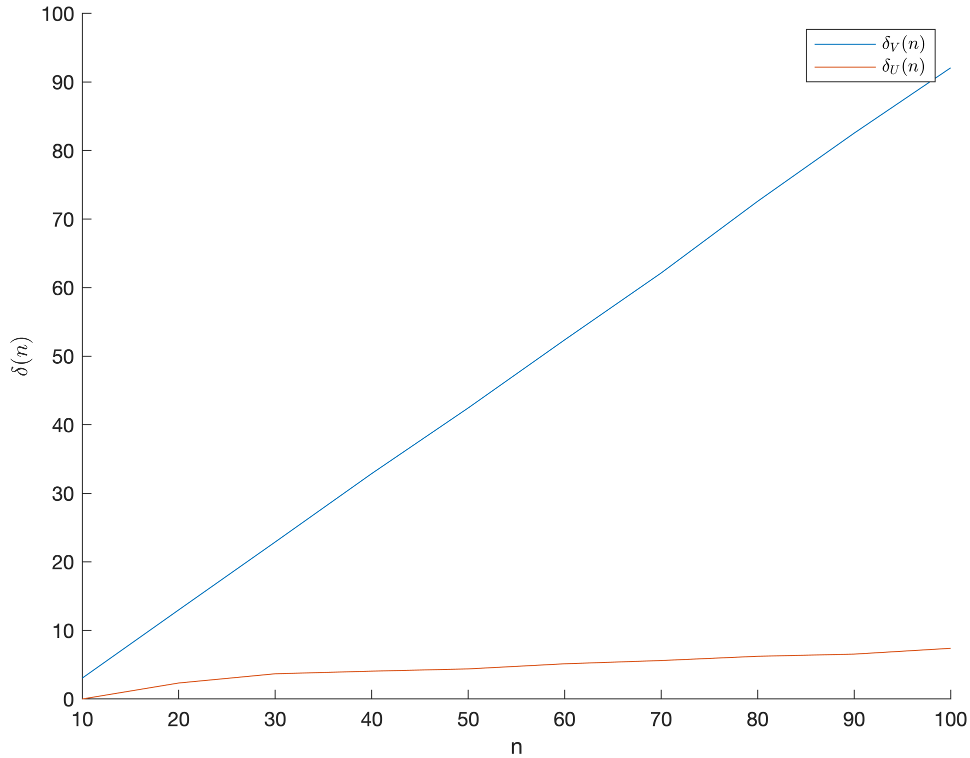
Problem 5 (10 points) Stability of the Gram-Schmidt Algorithm

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)
%}
clc; clear;
delta_v = zeros(1, 10);
delta_u = zeros(1, 10);
for n=10:10:100
    %step 1
    H = hilb(n);

    %step 2, Gram-Schmidt (unstable version)
    basis = zeros(n, n);
    basis(:,1) = H(:,1);
    for k = 2:n
        wk = H(:,k);
        sum = zeros(n, 1);
        for i = 1:(k-1)
            vi = basis(:,i);
            sum = sum + dot(wk, vi) * vi / (norm(vi) * norm(vi));
        end
        basis(:, k) = wk - sum;
    end
```

```matlab
    %step 3, Gram-Schmidt (unstable version)
    %normalize all vector
    for i = 1:n
        basis(:,i) = basis(:,i) / norm(basis(:,i));
    end

    A = eye(n) - basis' * basis;
    measure_unstable = norm(A, Inf);
    index = cast(n/10, "uint8");
    delta_v(1, index ) = measure_unstable;


    %step 2, Gram-Schmidt (stable version)
    basis = zeros(n, n);
    for j = 1:n
        basis(:, j) = H(:,j)/norm(H(:,j));
        for k = (j+1):n
            H(:,k) = H(:,k) - dot(H(:,k),basis(:, j)) * basis(:, j);
        end
    end

    %step 3, vectors are already normalized
    A = eye(n) - basis' * basis;
    measure_stable = norm(A, Inf);
    index = cast(n/10, "uint8");
    delta_u(1, index ) = measure_stable;

end
%{
Visualize the comparison as specified in (5)
%}
figure;
hold on
plot(10:10:100, delta_v);
plot(10:10:100, delta_u);
legend('$\delta_V (n)$','$\delta_U (n)$', 'Interpreter','latex');
xlabel("n");
ylabel("$\delta (n)$","Interpreter","latex");
```


$\delta_V(n)$
$\delta_U(n)$
100
90
80
70
60
50
40
30
20
10
0
$\delta(n)$
10
20
30
40
50
60
70
80
90
100
n