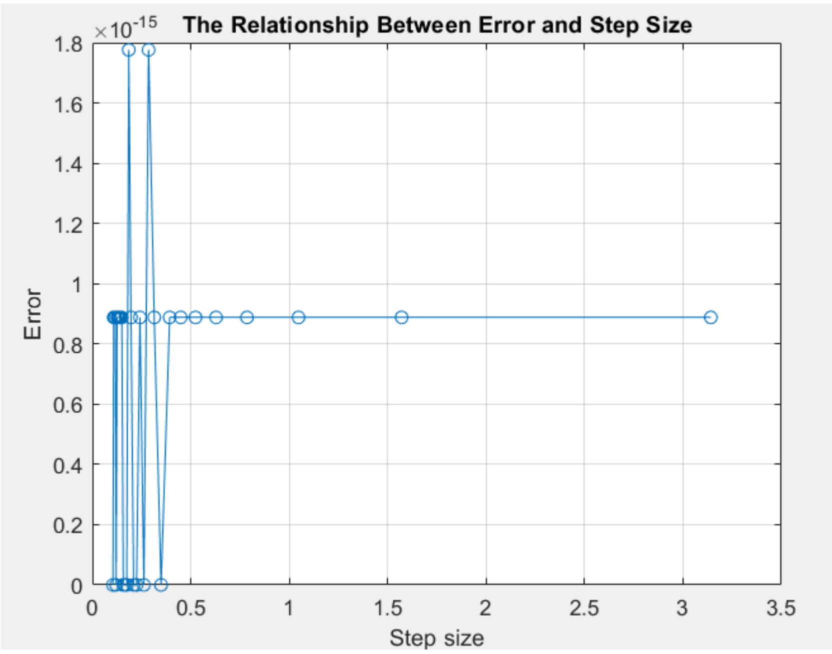
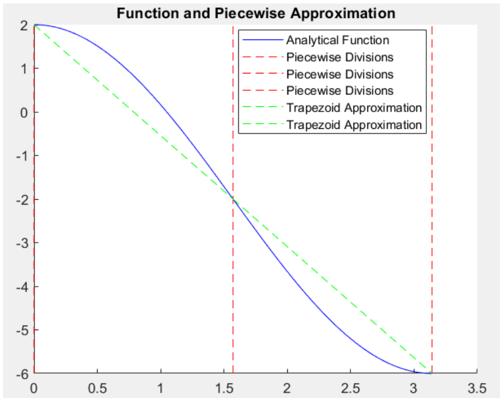
# FEM Hw1

應力所 碩一 R12543117
段翔齡

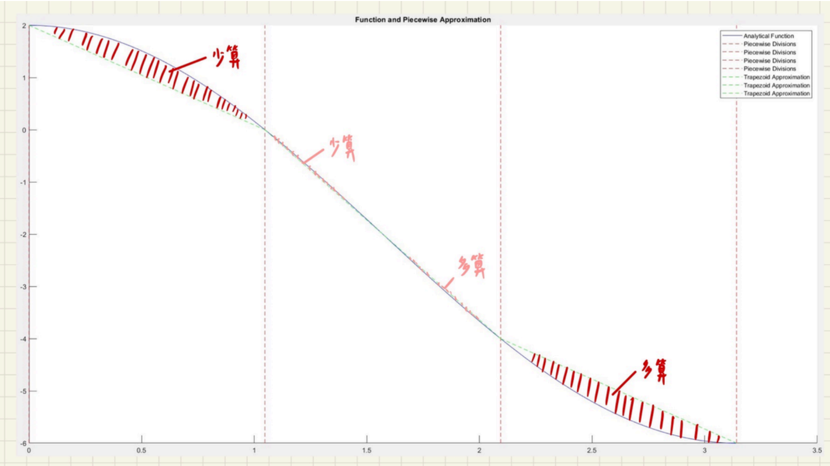
## Question 1:

1. Please apply the *trapezoidal rule* to compute the integral of the function $4\cos(x) - 2$ over the interval $[0, \pi]$. Compare your result with its analytical answer and discuss the error due to step size.

*Matlab Figures:*

| 步長對誤差的關係圖 | 切割兩塊梯形法與解析解誤差探討 |
|---|---|
|  | <br>Command Window<br>第1段面積差異: 0.85841<br>第2段面積差異: -0.85841<br>總解析面積: -6.2832<br>總梯形法面積: -6.2832<br>總面積差異: 8.8818e-16 |

由上圖左可以觀察到不管切割Step為1~30 (Step size為其倒數)，其誤差都非常小。近一步分析，由於給定三角函數範圍恰好在[0,pi]區間為對稱，當切割為2塊時(如圖右)，其解析解與梯形法所計算的解，其面積差剛好呈現互補關係，表示梯形法第一塊所少算的面積恰巧為梯形法第二塊多算之面積，故總面積誤差趨近於0。




Command Window

```
第1段面積差異: 0.32251
第2段面積差異: -4.4409e-16
第3段面積差異: -0.32251
總解析面積: -6.2832
總梯形法面積: -6.2832
總面積差異: -8.8818e-16
fx >>
```

如上圖所示為分割奇數塊，可以觀察到恰好左邊第一塊與右邊第一塊之面積差加起來恰好是0，故不論分割步長是細或粗，其結果皆相同。

# Question 2:

2. Consider the function $f(x) = 4x^3 + 7x^2 + 2x + 9$. Please apply at least two numerical differentiation schemes to calculate the first derivative of $f(x)$ in the interval [0, 1]. Plot the results and discuss its errors based on the exact $\frac{df(x)}{dx}$ due to step size.

Forward Difference Scheme : $f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$

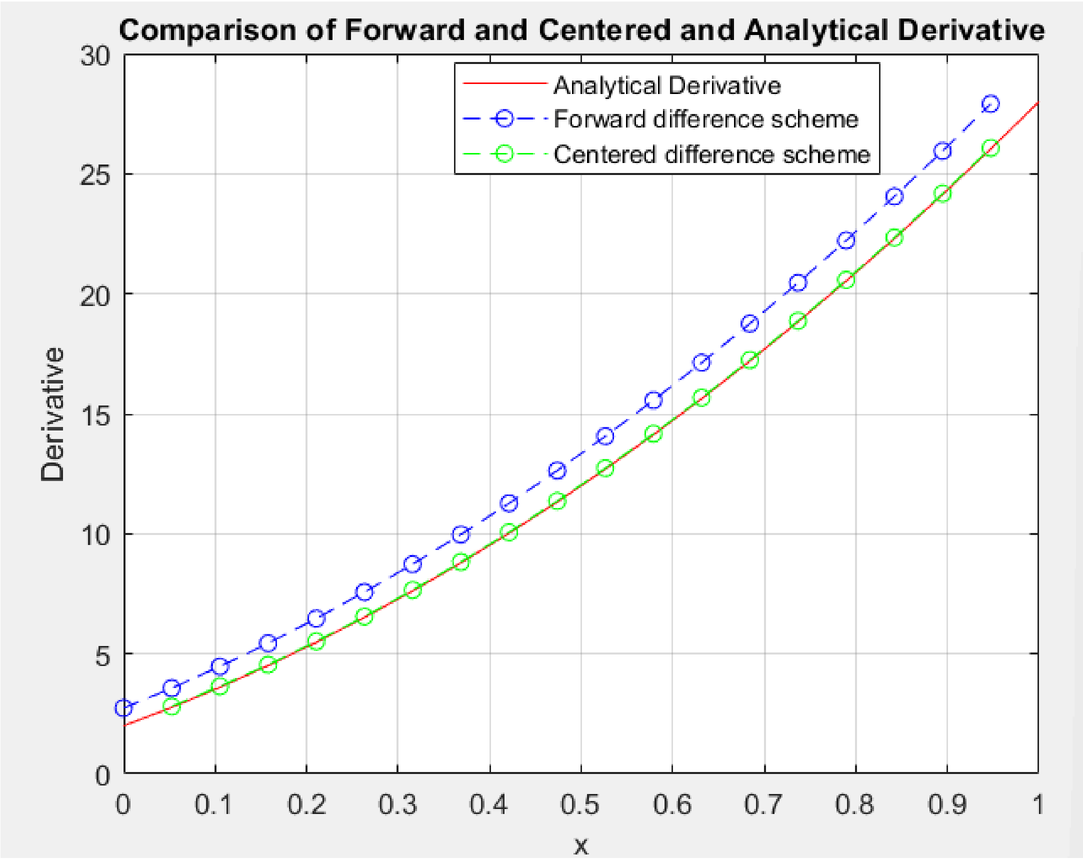
Centered Difference Scheme : $f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$

由以下圖形可以觀察到，步長delta x越小其誤差值越小，更接近於解析解，以及相較於前向差分法(Forward Difference Scheme)，中心差分法的誤差值更小，而且在步長delta x=0.1的情形下，其誤差值已經非常小。

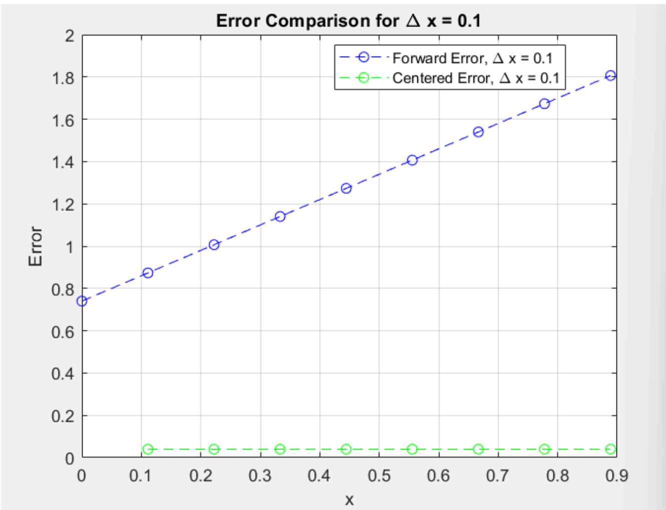
另一點，Matlab程式碼有另一項控制取樣點為，% x=linspace(0,1,20); 當取樣點越密集，也會得到更平滑的結果，相對也會增加消耗運算資源，有時需要取捨。以下圖皆採用取樣20個點，繪圖出來並計算誤差。(詳見Ref.程式碼)

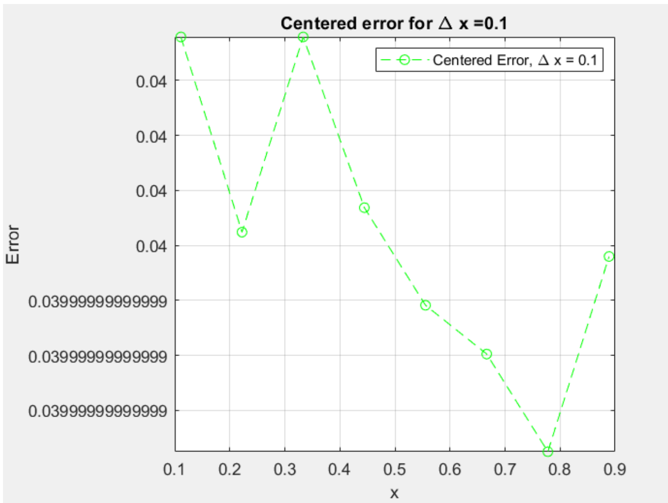
*Matlab Figures:*

## 步長delta x=0.1

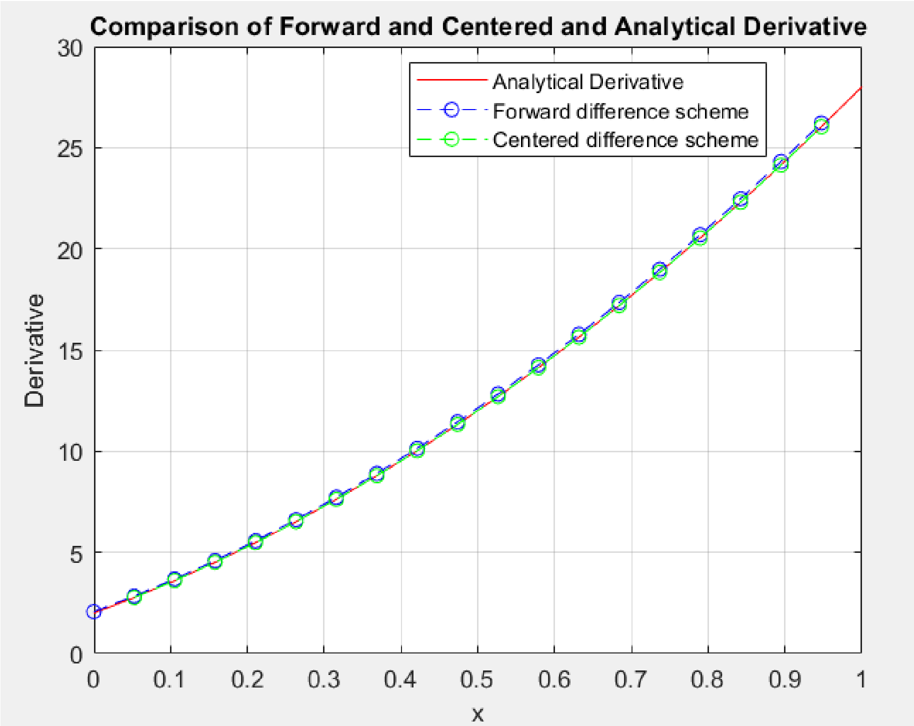



```
For delta_x = 0.1000:
    Total Forward Error: 11.460000
    Total Centered Error: 0.320000
```

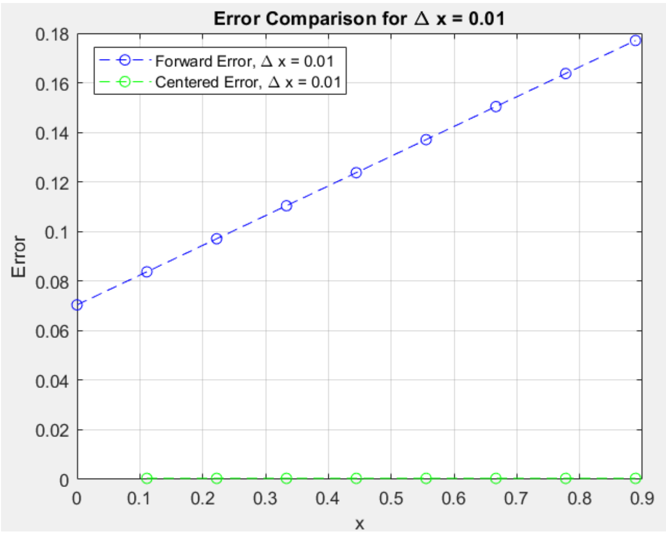


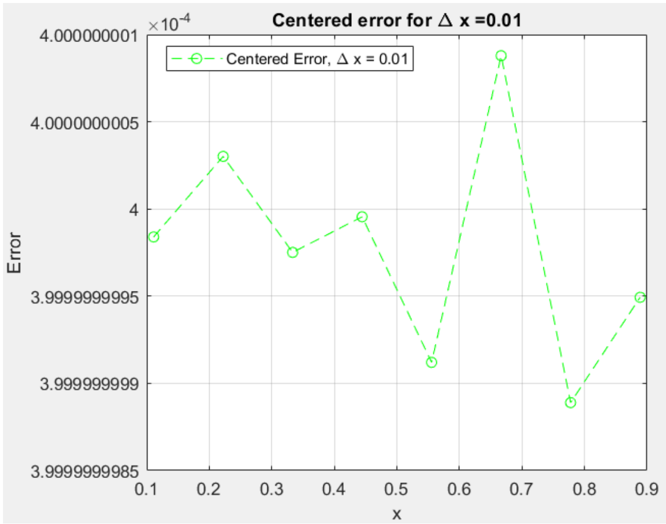



# 步長delta x=0.01

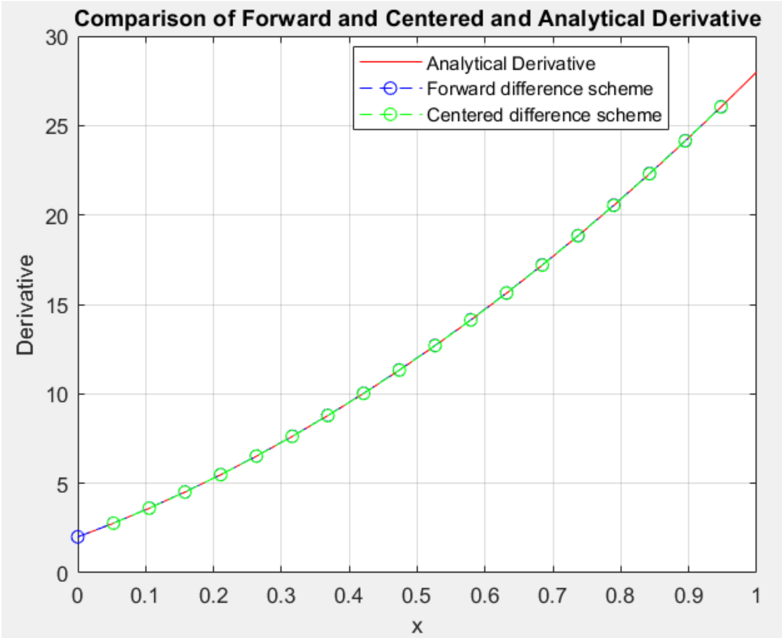



```
For delta_x = 0.0100:
    Total Forward Error: 1.113600
    Total Centered Error: 0.003200
```

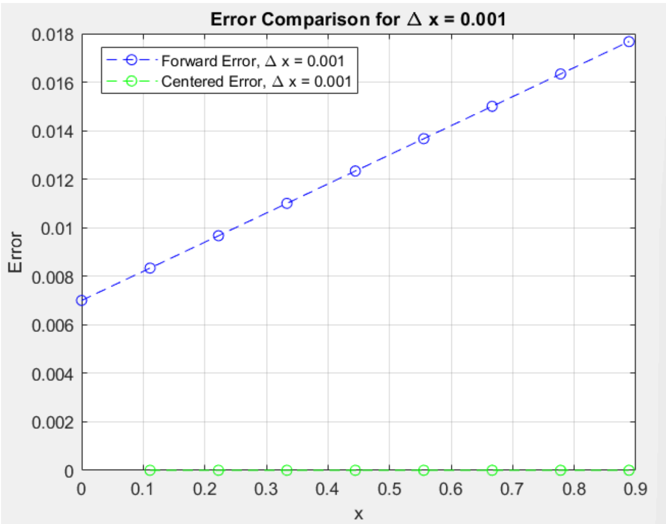


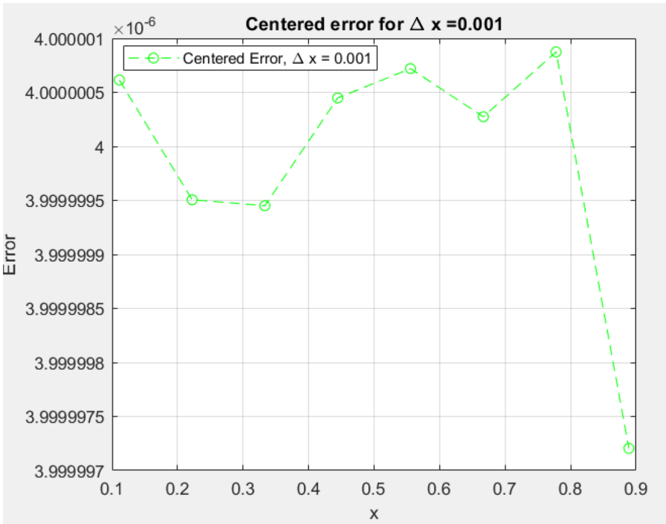



# 步長delta x=0.001




```
For delta_x = 0.0010:
    Total Forward Error: 0.111036
    Total Centered Error: 0.000032
```



# Question 3:

3. $\frac{dy}{dx} = y - 3x^2 + 1,\ y(0) = 0.5.$ Please use the *Euler method* to solve the following initial value problem on the interval [0, 0.1] with a step size $h = 0.005$. Then compare your results with the analytical answer.

**在計算解析解時，法一：手寫計算（使用工數所學進行化簡），法二:Matlab 程式進行運算(使用Matlab 函式 dsolve）**

$\frac{dy}{dx} = y - 3x^2 + 1,\ y(0) = 0.5.$

$\frac{dy}{dx} - y = -3x^2 + 1$

$P(x) = -1$

$Q(x) = -3x^2 + 1$

積分因子

$\mu(x) = e^{\int P(x)dx} = e^{-x}$

同乘積分因子

$e^{-x}\frac{dy}{dx} - ye^{-x} = e^{-x}(-3x^2+1)$

$\Rightarrow \frac{d}{dx}(e^{-x}y) = e^{-x}(-3x^2+1)$

对兩端積分

$e^{-x}y = \int e^{-x}(-3x^2+1)\,dx + c$

$\Rightarrow e^{-x}y = e^{-x}(3x^2+6x+5) + c$

$y = 3x^2+6x+5+ce^{x}$

$y(0) = 0.5$ 代入

$y = 3x^2+6x+5-\frac{9}{2}e^{x}$

$\int e^{-x}(-3x^2+1)\,dx$

$= -\int(-3x^2+1)\,d(e^{-x})$

$= -\left[e^{-x}(-3x^2+1) - \int e^{-x}\,d(-3x^2+1)\right]$

$= -\left[e^{-x}(-3x^2+1) - \int(-6x)e^{-x}dx\right]$

$= -\left[e^{-x}(-3x^2+1) - \int(6x)\,d(e^{-x})\right]$

$= e^{-x}(3x^2-1) + \left[e^{-x}(6x) - \int e^{-x}d(6x)\right]$

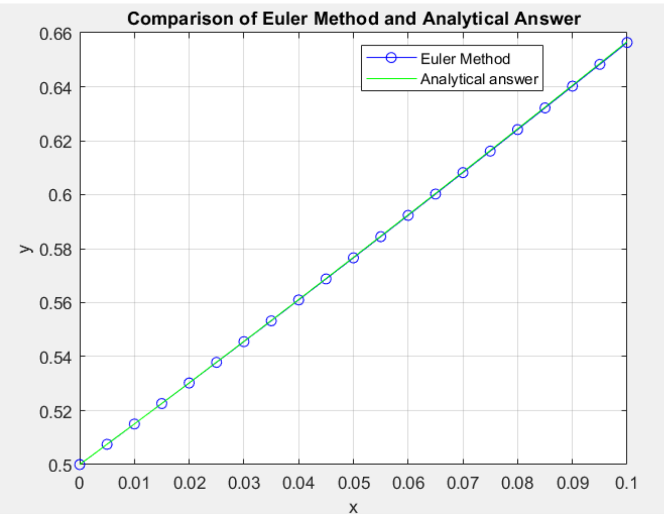
$= e^{-x}(3x^2-1) + e^{-x}(6x) + 6e^{-x}$

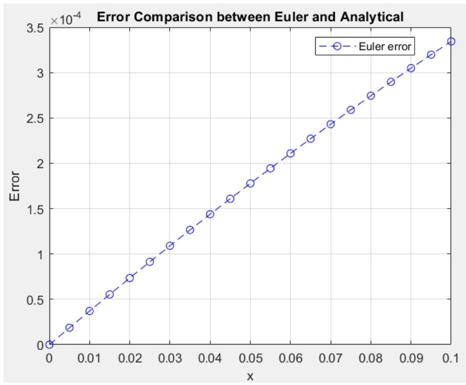
$= e^{-x}(3x^2+6x+5)$

```
%%
clear;
clc;

syms x y(x)
dydx=diff(y,x)==y-3*x^2+1;

ysol=dsolve(dydx);

disp('The general solution is:');
disp(ysol);

% 設定初始條件 y(0) = 1
ySol_with_condition = dsolve(dydx, y(0) == 0.5);

% 顯示帶初始條件的解
disp('The solution with the initial condition y(0) = 0.5 is:');
disp(ySol_with_condition);
```

## Matlab Figures:

| Euler 法與解析解作圖 | Error 誤差 |
| --- | --- |




```
Command Window
Total Error Sum: 0.003652
fx >>
```

**使用Euler 法且固定步長h=0.005，在取樣點20個點的情形下，其誤差已經非常低。。**

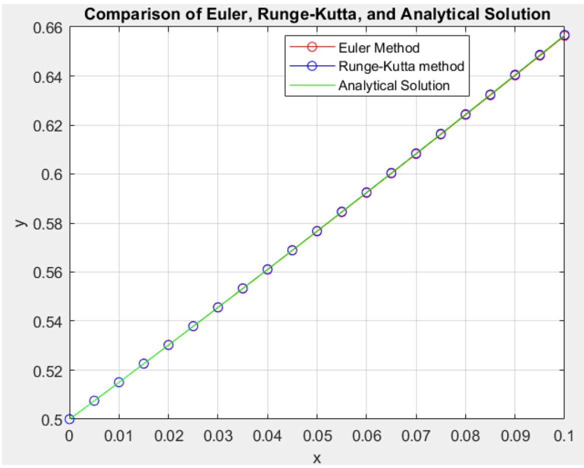
# Question 4:

4. $\frac{dy}{dx} = y - 3x^2 + 1,\ y(0) = 0.5$. Please use *the fourth-order Runge-Kutta method* to solve the following initial value problem on the interval [0, 0.1] with a step size $h = 0.005$. Then compare your results with the ones of Problem 3.

**在使用 fourth-order Runge-Kutta方法可以觀察到與解析解的誤差已經近似於0，進一步將Euler法、 Runge-Kutta方法的誤差抓出來，Euler法誤差的尺度為10(^-4)次方、 Runge-Kutta法誤差的尺度10(^-12)次方。**

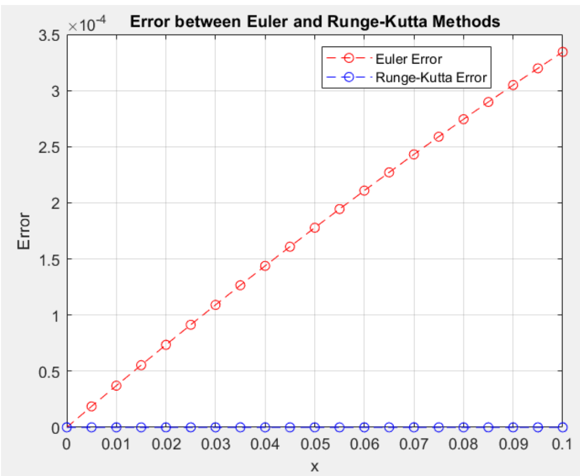
## Matlab Figures:

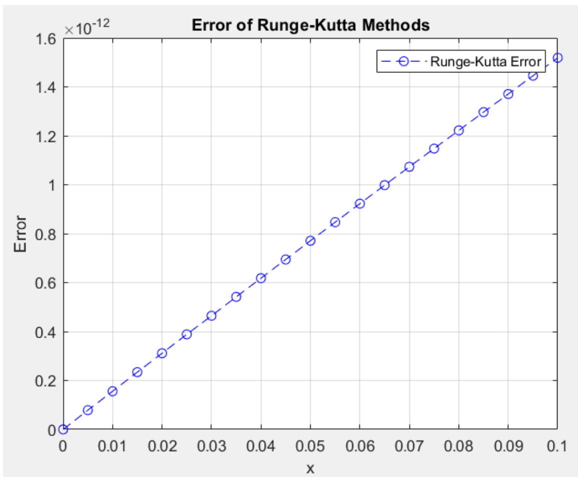
**Euler Method, Fourth-order Runge-Kutta, Analytical answer 作圖**




Command Window

```
Total Euler Method Error: 0.003652
Total Runge-Kutta Method Error: 0.000000
fx >>
```



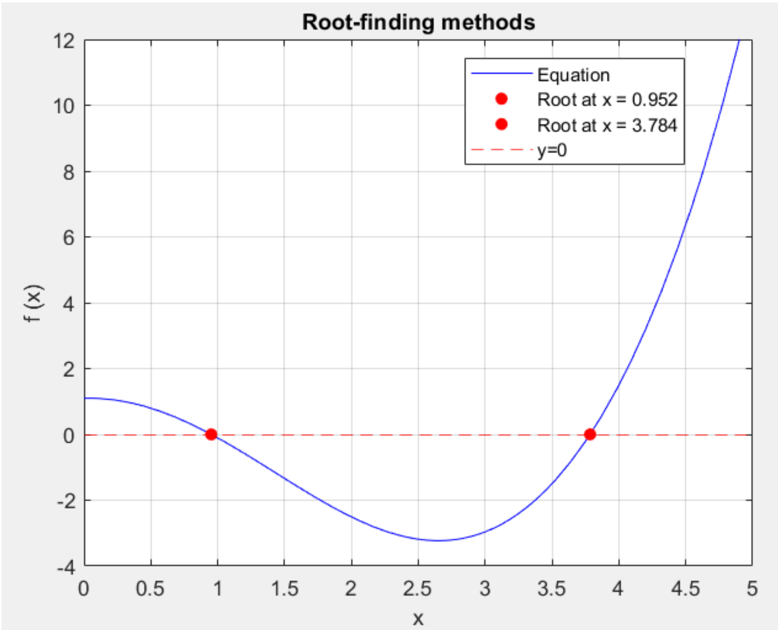
# Question 5:

5. Please find the real root of the equation $f(x) = 3\cos(x) + 0.1e^{x} - 2$ by using one of your favorite root-finding methods to locate the two roots of $f(x)$ that are nearest to $x = 2$.

**使用勘根定理需要先決定兩個猜測值，Matlab再根據猜測值附近進行勘根，故初始猜測值要適當給予，其代入方程式的誤差也幾近於0。**

## Matlab Figures:

| Root-finding methods 作圖 |
| --- |




Command Window

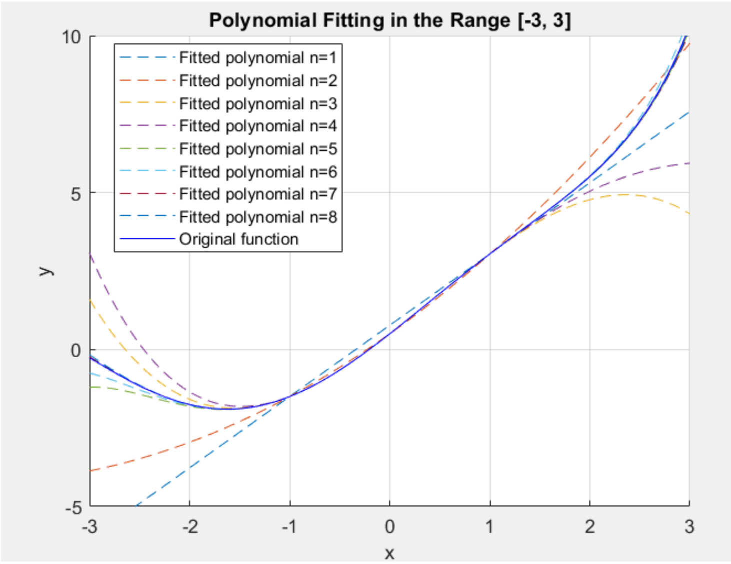
```
勘根定理之一根為: 0.951658,代入方程式:4.440892e-16
勘根定理另一根為: 3.784452,代入方程式:-4.440892e-16
fx >>
```

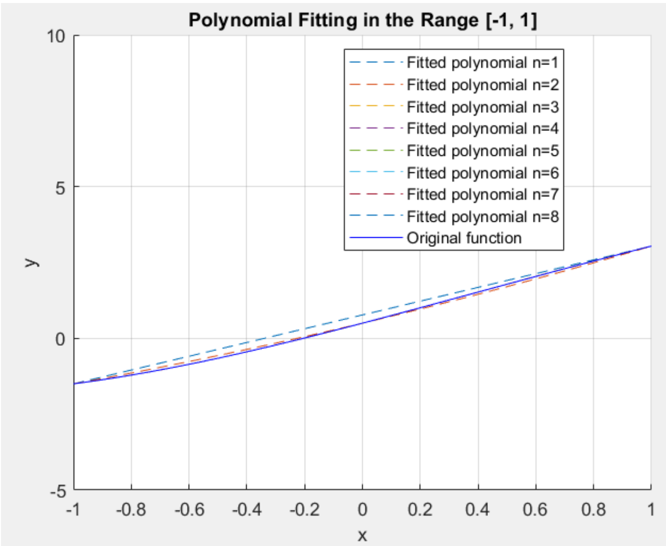
# Question 6:

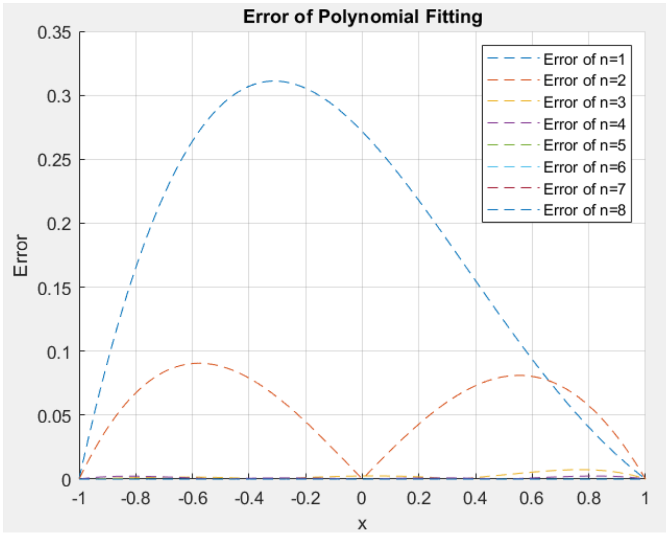
6. Consider the function $f(x) = 3\sin x + \frac{1}{2}e^{x}$. Please use polynomial interpolation with proper orders to estimate the values of the function based on points that are evenly spaced within the range of $-1$ to 1. In addition, analyze how the error is distributed for different numbers of interpolation points with various polynomial orders.

以下圖為在區間[-1 1]範圍內進行多項式差值法，可以觀察到多項式次數越多次與函式也會越擬合，其誤差隨著多項式項次增加而下降。

**Euler Method, fourth-order Runge-Kutta, analytical answer作圖**




Command Window

```
Total errors in the range [-1, 1]:
Total error for polynomial n=1: 183.755585
Total error for polynomial n=2: 55.608657
Total error for polynomial n=3: 2.353678
Total error for polynomial n=4: 1.005259
Total error for polynomial n=5: 0.017219
Total error for polynomial n=6: 0.003441
Total error for polynomial n=7: 0.000082
Total error for polynomial n=8: 0.000024
fx >>
```

# Reference Matlab Code:

## Question 1

### Section 1 畫出error對step圖

hw1_1.m | hw1_2.m | hw1_3.m | hw1_4.m | hw1_5.m | hw1_6.m

```matlab
clear;
clc;

f=@(x) 4*cos(x)-2;
% 積分範圍
a=0;
b=pi;

% 解析解
% q = integral(fun,xmin,xmax)
integral_fun=integral(f,a,b);

% 定義步數
max_n=30;
errors=zeros(1,max_n);
step_sizes=zeros(1,max_n);

for n_number=1:max_n
    %利用linspace 分點
    x=linspace(a,b,n_number+1);
    y=f(x);

    %初始化area
    area=0;

    %計算步長
    step_size=(b-a)/n_number;
    step_sizes(n_number) = step_size;

    for i=1:n_number
        area=area+(y(i)+y(i+1))*step_size/2;
    end
    errors(n_number)=abs(integral_fun-area);
end

figure;
plot(step_sizes,errors,'-o');
xlabel('Step size');
ylabel('Error');
title('The Relationship Between Error and Step Size');
grid on;
```

### Section 2 細部探討切割面積

```matlab
%%
clc;
clear;
%定義函數
f=@(x) 4*cos(x)-2;
f_integral = @(x) 4*sin(x) - 2*x; % 函數的解析積分

% 積分範圍
a=0;
b=pi;

%利用linspace 分點
n_piece=3;
x=linspace(a,b,n_piece+1);
y=f(x);

% 解析解用來作為對比
x_analytical=linspace(a,b,1000);
y_analytical=f(x_analytical);

%初始化area (梯形法)
area_trapz = zeros(1, n_piece); % 梯形法計算面積
area_exact = zeros(1, n_piece); % 解析法計算面積
step_size=(b-a)/n_piece;

for i=1:n_piece
    area_trapz(i)=(y(i)+y(i+1))*step_size/2;
    % 解析面積: 利用已知的積分函數計算該區間的積分
    area_exact(i) = f_integral(x(i+1)) - f_integral(x(i));
```

```matlab
    area_exact(i) = f_integral(x(i+1)) - f_integral(x(i));

    % 顯示每段面積的差異
    area_diff = area_exact(i) - area_trapz(i);
    disp(['第', num2str(i), '段面積差異: ', num2str(area_diff)]);
end

% 顯示總面積
total_area_trapz = sum(area_trapz);
total_area_exact = f_integral(b) - f_integral(a);
disp(['總解析面積: ', num2str(total_area_exact)]);
disp(['總梯形法面積: ', num2str(total_area_trapz)]);
disp(['總面積差異: ', num2str(total_area_exact - total_area_trapz)]);
figure;
hold on;

% 畫解析解曲線
plot(x_analytical,y_analytical,'b-','DisplayName','Analytical Function');

% 在每個分段位置畫虛線
for i = 1:length(x)
    % 畫垂直虛線
    plot([x(i), x(i)], [min(y_analytical), max(y_analytical)], 'r--', 'DisplayName', 'Piecewise Divisions');
end

% 繪製分段線條
for i = 1:n_piece
    plot([x(i), x(i+1)], [y(i), y(i+1)], 'g--', 'DisplayName', 'Trapezoid Approximation');
end

xlabel=('x');
ylabel=('y');
legend('show');
title('Function and Piecewise Approximation');
```

# Question 2

## Section 1畫兩種有限差分法與解析解

hw1_1.m hw1_2.m hw1_3.m hw1_4.m hw1_5.m hw1_6.m

```matlab
clear;
clc;

f=@(x) 4*x.^3+7*x.^2+2*x+9;
df=@(x) 12*x.^2+14*x+2;

x=linspace(0,1,20); %更密集的點更平滑的結果
delta_x=0.001; %影響精度

%初始化前向差分和中心差分結果
forward_diff=zeros(1,length(x)-1);  %少一個點，最後一點會超出邊界
centered_diff=zeros(1,length(x)-2); %少兩個點，第一點與最後一點會超出邊界

%Forward difference scheme
for i=1:length(x)-1 %最後一點會超出邊界
    forward_diff(i)=(f(x(i)+delta_x)-f(x(i)))/delta_x;
end

%Centered-difference scheme
for i=2:length(x)-1 %從第二個點開始，第一點與最後一點會超出邊界
     centered_diff(i-1)=(f(x(i)+delta_x)-f(x(i)-delta_x))/(2*delta_x);
end
%Analytical Derivative
analytical_derivative = df(x);

% 計算誤差
forward_error = abs(analytical_derivative(1:end-1) - forward_diff);
centered_error = abs(analytical_derivative(2:end-1) - centered_diff);

% 計算誤差總和
total_forward_error = sum(forward_error);
total_centered_error = sum(centered_error);

% 列印誤差總和
fprintf('Total Forward Error: %.6f\n', total_forward_error);
fprintf('Total Centered Error: %.6f\n', total_centered_error);

figure;
plot(x,analytical_derivative,'r-','DisplayName','Analytical Derivative');
hold on;

plot(x(1:end-1),forward_diff,'b--o','DisplayName','Forward difference scheme');
hold on;

plot(x(2:end-1),centered_diff,'g--o','DisplayName','Centered difference scheme');
xlabel('x');
ylabel('Derivative');
title('Comparison of Forward and Centered and Analytical Derivative');
legend show;
grid on;
```

## Section 2 畫不同delta_x步長之error

hw1_1.m hw1_2.m hw1_3.m hw1_4.m hw1_5.m hw1_6.m

```matlab
%%
clear;
clc;

f=@(x) 4*x.^3+7*x.^2+2*x+9;
df=@(x) 12*x.^2+14*x+2;

x=linspace(0,1,10); %更密集的點更平滑的結果
delta_x_values=[0.1 0.01 0.001 0.0001]; %影響精度

for d=1:length(delta_x_values)
    delta_x=delta_x_values(d);
    %初始化前向差分和中心差分結果
    forward_diff=zeros(1,length(x)-1);  %少一個點，最後一點會超出邊界
    centered_diff=zeros(1,length(x)-2); %少兩個點，第一點與最後一點會超出邊界

    %Forward difference scheme
    for i=1:length(x)-1 %最後一點會超出邊界
        forward_diff(i)=(f(x(i)+delta_x)-f(x(i)))/delta_x;
    end

    %Centered-difference scheme
    for i=2:length(x)-1 %從第二個點開始，第一點與最後一點會超出邊界
         centered_diff(i-1)=(f(x(i)+delta_x)-f(x(i)-delta_x))/(2*delta_x);
    end
    %Analytical Derivative
    analytical_derivative = df(x);

     % 計算誤差
    forward_error = abs(analytical_derivative(1:end-1) - forward_diff);
    centered_error = abs(analytical_derivative(2:end-1) - centered_diff);

    % 計算誤差總和
    total_forward_error = sum(forward_error);
    total_centered_error = sum(centered_error);

    % 使用 fprintf 列印誤差總和
    fprintf('For delta_x = %.4f:\n', delta_x);
    fprintf('    Total Forward Error: %.6f\n', total_forward_error);
    fprintf('    Total Centered Error: %.6f\n', total_centered_error);
    % 繪製誤差圖
    figure;
    plot(x(1:end-1), forward_error, 'b--o', 'DisplayName', ['Forward Error, \Delta x = ' num2str(delta_x)]);
    hold on;
    plot(x(2:end-1), centered_error, 'g--o', 'DisplayName', ['Centered Error, \Delta x = ' num2str(delta_x)]);
    xlabel('x');
    ylabel('Error');
    title(['Error Comparison for \Delta x = ', num2str(delta_x)]);
    legend show;
    grid on;
    figure;
    plot(x(2:end-1), centered_error, 'g--o', 'DisplayName', ['Centered Error, \Delta x = ' num2str(delta_x)]);
    title(['Centered error for \Delta x =', num2str(delta_x)]);
    xlabel('x');
    ylabel('Error');
    legend show;
```

# Question 3

## Euler 法

Editor - C:\Users\user\Desktop\碩一上\FEM_0915\FEM_0916\FEM\HW_1\hw1_3.m

hw1_1.m | hw1_2.m | hw1_3.m | hw1_4.m | hw1_5.m | hw1_6.m

```matlab
clear;
clc;
%微分方程dydx
dydx=@(x,y) y-3*x^2+1;
y_ana=@(x) (3*x.^2+6*x+5)-(9/2*exp(x)); %解析解 (利用手寫解出)

h=0.005; %步長
a=0;
b=0.1;
x=a:h:b;
step_n=length(x)-1; %步數

%初始化
y_Euler=zeros(size(x));
y_analytical=zeros(size(x));

%初始條件
y_Euler(1)=0.5;

%Euler 方法求解
for i=1:step_n
    dydx_Euler=dydx(x(i),y_Euler(i));
    y_Euler(i+1)=y_Euler(i)+h*dydx_Euler;
end

%Analytical answers
y_analytical=y_ana(x);

%計算誤差
Euler_error=abs(y_Euler-y_analytical);

% 計算誤差總和
total_error = sum(Euler_error);

% 使用fprintf輸出誤差總和
fprintf('Total Error Sum: %.6f\n', total_error);

figure;
plot(x,y_Euler,'bo-','DisplayName','Euler Method');
hold on;
plot(x,y_analytical,'g-','DisplayName','Analytical answer');
xlabel('x');
ylabel('y');
title('Comparison of Euler Method and Analytical Answer');
legend('show');
grid on;

figure;
plot(x,Euler_error,'b--o','DisplayName','Euler error');
hold on;

xlabel('x');
ylabel('Error');
title('Error Comparison between Euler and Analytical');
legend show;
grid on;
```

## Matlab 解微分方程

```matlab
%%
clear;
clc;

syms x y(x)
dydx=diff(y,x)==y-3*x^2+1;

ysol=dsolve(dydx);

disp('The general solution is:');
disp(ysol);

% 設定初始條件 y(0) = 1
ySol_with_condition = dsolve(dydx, y(0) == 0.5);

% 顯示帶初始條件的解
disp('The solution with the initial condition y(0) = 0.5 is:');
disp(ySol_with_condition);
```

# Question 4

## 比較Question 3 and 4 並作圖呈現

hw1_1.m | hw1_2.m | hw1_3.m | hw1_4.m | hw1_5.m | hw1_6.m

```matlab
clear;
clc;

dydx=@(x,y) y-3*x^2+1;
y_ana=@(x) (3*x.^2+6*x+5)-(9/2*exp(x)); %解析解 (利用手寫解出)

a=0;
b=0.1;
h=0.005;
x=a:h:b;
step=length(x);

%初始化
y_Euler=zeros(1,step);
y_Runge=zeros(1,step);

%初始條件
y_Euler(1)=0.5;
y_Runge(1)=0.5;


%Euler 方法求解
for i=1:step-1
    dydx_Euler=dydx(x(i),y_Euler(i));
    y_Euler(i+1)=y_Euler(i)+h*dydx_Euler;
end

% Runge-Kutta
for i=1:step-1
    k1=h*dydx(x(i),y_Runge(i));
    k2=h*dydx(x(i)+h/2,y_Runge(i)+k1/2);
    k3=h*dydx(x(i)+h/2,y_Runge(i)+k2/2);
    k4=h*dydx(x(i)+h,y_Runge(i)+k3);
    y_Runge(i+1)=y_Runge(i)+1/6*(k1+2*k2+2*k3+k4);
end

%Analytical answers
y_analytical=y_ana(x);

%計算誤差
Euler_error=abs(y_Euler-y_analytical);
Runge_Kutta_error=abs(y_Runge-y_analytical);

% 計算誤差總和
total_Euler_error = sum(Euler_error);
total_Runge_Kutta_error = sum(Runge_Kutta_error);

% 使用 fprintf 列印出誤差總和
fprintf('Total Euler Method Error: %.6f\n', total_Euler_error);
fprintf('Total Runge-Kutta Method Error: %.6f\n', total_Runge_Kutta_error);

figure;
plot(x, Runge_Kutta_error, 'b--o', 'DisplayName', 'Runge-Kutta Error');
xlabel('x');
ylabel('Error');
title('Error of Runge-Kutta Methods');
legend show;
grid on;

figure;
plot(x,y_Euler,'ro-','DisplayName','Euler Method');
hold on;
plot(x,y_Runge,'bo-','DisplayName','Runge-Kutta method');
hold on;
plot(x,y_analytical,'g-','DisplayName','Analytical Solution');
title('Comparison of Euler, Runge-Kutta, and Analytical Solution');
xlabel('x');
ylabel('y');
legend('show');
grid on;

figure;
plot(x, Euler_error, 'r--o', 'DisplayName', 'Euler Error');
hold on;
plot(x, Runge_Kutta_error, 'b--o', 'DisplayName', 'Runge-Kutta Error');
xlabel('x');
ylabel('Error');
title('Error between Euler and Runge-Kutta Methods');
legend show;
grid on;
```

# Question 5

## Root - finding methods

```
clear;
clc;

f=@(x) 3*cos(x)+0.1*exp(x)-2;

%找到兩個猜測值附近的根
x0_1=2;
root1=fzero(f,x0_1);

x0_2=4;
root2=fzero(f,x0_2);

fprintf('勘根定理之一根為: %.6f,代入方程式:%e\n',root1,f(root1));
fprintf('勘根定理另一根為: %.6f,代入方程式:%e\n',root2,f(root2));

figure;
fplot(f,[0 5],'b-','DisplayName','Equation');
hold on;
scatter(root1, f(root1), 'ro', 'filled', 'DisplayName', sprintf('Root at x = %.3f', root1));
scatter(root2, f(root2), 'ro', 'filled', 'DisplayName', sprintf('Root at x = %.3f', root2));
yline(0,'r--','DisplayName','y=0');

xlabel('x');
ylabel('f (x)');
xlim([0 5]);
ylim([-4 12]);
title('Root-finding methods');
hold on;
legend('show');
grid on;
```

# Question 6

## Polynomial interpolation

hw1_1.m | hw1_2.m | hw1_3.m | hw1_4.m | hw1_5.m | hw1_6.m

```matlab
clear;
clc;

% 定義函數
f = @(x) 2*sin(x) + (1/2)*exp(x);

x_fit=linspace(-1,1,1000);  %擬合範圍
x_plot=linspace(-3,3,1000); %繪製擬合多項式圖

y_fun=f(x_fit);
n_order=8; %擬合多項式次數

% 顏色選擇
colors = lines(n_order); % 生成 n 種顏色

% 創建圖形
figure;
hold on;

%儲存誤差
errors=zeros(n_order,length(x_fit));

for n=1:n_order
    x=linspace(-1,1,n+1); %使用n+1個點
    y=f(x);

    p=polyfit(x,y,n);
    y_fit=polyval(p,x_fit);

    error_fit=abs(y_fit-y_fun);
    errors(n, :) = error_fit; % 儲存誤差

    plot(x_fit,y_fit,'LineStyle','--','Color',colors(n,:),...
        'DisplayName',sprintf('Fitted polynomial n=%d', n));
end
% Plot original function
plot(x_fit,y_fun,'b-','DisplayName','Original function');
xlim([-1,1]);
ylim([-5,10]);
xlabel('x');
ylabel('y');
title('Polynomial Fitting in the Range [-1, 1]');
legend('show');
grid on;

% 創建誤差圖
figure;
hold on;

% 繪製每個擬合線的誤差
for n = 1:n_order
    plot(x_fit, errors(n, :), 'LineStyle', '--', 'Color', colors(n, :), ...
        'DisplayName', sprintf('Error of n=%d', n));
end

xlabel('x');
ylabel('Error');
title('Error of Polynomial Fitting');
legend('show');
grid on;

% 計算並列印誤差總和
total_errors = sum(errors, 2); % 每條擬合曲線的總誤差
disp('Total errors in the range [-1, 1]:');
for n = 1:n_order
    fprintf('Total error for polynomial n=%d: %.6f\n', n, total_errors(n));
end


% 創建圖形：在區間 [-3, 3] 上繪製擬合方程式
figure;
hold on;

for n=1:n_order
    x=linspace(-1,1,n+1); %使用n+1個點
    y=f(x);

    p=polyfit(x,y,n);
    y_fit=polyval(p,x_plot);

    plot(x_plot,y_fit,'LineStyle','--','Color',colors(n,:),...
        'DisplayName',sprintf('Fitted polynomial n=%d', n));
end

% 繪製原始函數
plot(x_plot, f(x_plot), 'b-', 'DisplayName', 'Original function');
xlim([-3, 3]);
ylim([-5, 10]);
xlabel('x');
ylabel('y');
title('Polynomial Fitting in the Range [-3, 3]');
legend('show');
grid on;
```