Eli Vlahos

Simulation Assignment #4

Ayman El-Hag

05/12/2020

[**Introduction**](#_8eh6vw228l1v) **2**

[**Results for Question**](#_mb7mnh9rdwbt) **16**

[**Conclusions**](#_xy9mnr7l5byz) **19**

# Introduction

The purpose of the lab was to create a holistic program to approximate derivatives and integrals. This was done by using a polynomial regression of third order in cases where the dataset along was not sufficient. To test the results an example data set and points were used.

Code

Main file (comments are included in code.

%reads text file

A\_1 = dlmread('test\_1.txt');

%user chooses model

I = input("Derivative (1) or Integral? (2) ");

if I == 1

Derivative(A\_1);

elseif I == 2

Integral(A\_1);

end

Primary Derivative file, used to initially parse the input

function Derivative(D)

arguments

%specifies the type of input

D (:,2) {mustBeNumeric, mustBeFinite};

end

%input

p = input("At what point would you like to evaluate? ");

%size of matrix

s = size(D);

s = s(1,1);

%kept track to see if the changes were the same throughout the data.

%keep track of the minimum so it can be used later

change = -D(1,1)+D(2,1);

last\_change = change;

minimum = change;

bool = 1;

equal = 0;

%checked to see if the difference was constant and if not what the

%minimum was

for i = 1:s

if i < s

if change ~= last\_change

minimum = change;

bool = 0;

end

last\_change = change;

change = -D(i,1)+D(i+1,1);

end

if D(i,1) == p

equal = 1;

end

end

%if all conditions are true use CDD immediately, otherwise call

%function

if equal && bool

lower = find(D(:,1) == p-change);

higher = find(D(:,1) == p+change);

f\_x = find(D(:,1) == p);

derivative = (D(higher,2)-D(lower,2))/(2\*change)

else

derivative = Polynomial\_Derivative(D,p,change)

end

end

This was the polynomial derivative function, where a polynomial to approximate the values was created and then used

function d = Polynomial\_Derivative(D, p, change)

arguments

%specifies the type of input

D (:,2) {mustBeNumeric, mustBeFinite};

p double;

change double;

end

%using third degree polynomial

m = 3;

%initializing zero matrices for the equation A\*X=B

%X is the matrix for the coefficients, A and B various sums

A = zeros(m+1);

B = zeros(m+1,1);

%take size

s = size(D);

s = s(1,1);

%initialize average of y

average\_y = 0;

for i = 1:s

%take sum of y

average\_y = average\_y + D(i,2);

end

%divide to get mean

average\_y = average\_y/s;

%goes through each square in the A matrix to initialize all values

for i = 1:m+1

for j = 1:m+1

if j == 1 & i == 1

%for the 1,1 point, initialize to size

A(1,1) = s;

else

for n = 1:s

%take sum and square values relative to the position

%in the matrix

A(i,j) = A(i,j)+D(n,1)^(i+j-2);

end

end

end

end

%Initialize the B vector

for i = 1:m+1

for j = 1:s

%take the sum for the value, using the relevant exponent

B(i,1) = B(i,1) + (D(j,1))^(i-1)\*D(j,2);

end

end

%find solution for coefficients

%initialize vectors for error, standard deviation and new Y values

X = inv(A)\*B;

new\_Y = zeros(s,1);

E = zeros(s,1);

Standard = zeros(s,1);

%changing preference for symbolic equations

sympref('FloatingPointOutput',true);

syms x y

%checking to see if the equation is linear and a data point indicates

%no y intercept

bool = 1;

for i = 1:s

if (D(i,2) == 0 & D(i,1) == 0 & m == 1)

%if the equation is linear and possess a 0,0 point, initialize

%differently

bool = 0;

end

end

%if the value fits the above conditions, will follow a different

%procedure for what function it will be defined as

if bool == 0

%take sums necessary for linear function

sum\_xy = 0;

sum\_xx = 0;

for i = 1:s

sum\_xy = sum\_xy + D(i,1)\*D(i,2);

sum\_xx = sum\_xx + D(i,1)\*D(i,1);

end

%initializes new function

f(x,y) = x\*(sum\_xy/sum\_xx);

str = append('Polynomial, y = ',num2str(sum\_xy/sum\_xx));

else

%otherwise initialize function as expected

f(x,y) = X(1,1)+x\*X(2,1);

str = append('Polynomial, y = ',num2str(X(1,1)),'+x\*',num2str(X(2,1)));

%use a loop to add all appropriate parts to the function and the

%string that defines it

for j = 3:m+1

f(x,y) = f(x,y)+X(j,1)\*x^(j-1);

str = append(str,'+',num2str(X(j,1)),'\*x^',num2str(j-1));

end

end

%initializes new\_Y, error and standard deviation vectors

for i = 1:s

new\_Y(i,1) = f(D(i,1),D(i,2));

E(i,1) = (D(i,2)-new\_Y(i,1))^2;

Standard(i,1) = (D(i,2)-average\_y)^2;

end

%initializes r\_2 and calls display function

r\_2 = (sum(Standard)-sum(E))/sum(Standard)

Display(D, new\_Y, str, r\_2);

%compute new derivative

d = (f(p+change,p+change)-f(p-change,p-change))/(2\*change);

end

Primary Integral File

function Integral(D)

arguments

%specifies the type of input

D (:,2) {mustBeNumeric, mustBeFinite};

end

%asks what the values are

p\_1 = input("What will be the lower point? ");

p\_2 = input("What will be the upper point? ");

n = input("What number of segments do you want? ");

%find size of input

s = size(D);

s = s(1,1);

%rows where the lower and upper limit are found

sum = 0;

row\_a = find(D(:,1) == p\_1);

row\_b = find(D(:,1) == p\_2);

%creates linearly spaced vector

X\_new = linspace(p\_1,p\_2,n+1);

change = X\_new(2)-X\_new(1);

%uses a boolean value to check to see whether or not values within the

%matrix are equal

equal = 1;

if n <= s

for i = 1:n

if ~(abs((X\_new(i) - D(row\_a+i-1,1))) < 0.01)

equal = 0;

end

end

else

equal = 0;

end

% checks to see if input is compatible with data

if p\_1 >= D(1,1) && p\_2 <= D(s,1)

bool = 1;

else

bool = 0;

end

%if both conditions are true can proceed as normal, otherwise need

%polynomial regression

if equal && bool

f\_a = D(row\_a,2);

f\_b = D(row\_b,2);

for i = 1:n-1

sum = sum+D(row\_a+i,2);

end

integral = ((p\_2-p\_1)/(2\*n))\*(f\_a+2\*sum+f\_b)

elseif ~equal && bool

integral = Polynomial\_Integral(D,p\_1,p\_2,n)

else

%gives warning

disp("one of the p values is out of range")

end

end

Polynomial Integral File

function i = Polynomial\_Integral(D, p\_1, p\_2, number)

arguments

%specifies the type of input

D (:,2) {mustBeNumeric, mustBeFinite};

p\_1 double;

p\_2 double;

number double;

end

%using linear polynomial

m = 3;

%initializing zero matrices for the equation A\*X=B

%X is the matrix for the coefficients, A and B various sums

A = zeros(m+1);

B = zeros(m+1,1);

%take size

s = size(D);

s = s(1,1);

%initialize average of y

average\_y = 0;

for i = 1:s

%take sum of y

average\_y = average\_y + D(i,2);

end

%divide to get mean

average\_y = average\_y/s;

%goes through each square in the A matrix to initialize all values

for i = 1:m+1

for j = 1:m+1

if j == 1 & i == 1

%for the 1,1 point, initialize to size

A(1,1) = s;

else

for n = 1:s

%take sum and square values relative to the position

%in the matrix

A(i,j) = A(i,j)+D(n,1)^(i+j-2);

end

end

end

end

%Initialize the B vector

for i = 1:m+1

for j = 1:s

%take the sum for the value, using the relevant exponent

B(i,1) = B(i,1) + (D(j,1))^(i-1)\*D(j,2);

end

end

%find solution for coefficients

%initialize vectors for error, standard deviation and new Y values

X = inv(A)\*B;

new\_Y = zeros(s,1);

E = zeros(s,1);

Standard = zeros(s,1);

%changing preference for symbolic equations

sympref('FloatingPointOutput',true);

syms x y

%checking to see if the equation is linear and a data point indicates

%no y intercept

bool = 1;

for i = 1:s

if (D(i,2) == 0 & D(i,1) == 0 & m == 1)

%if the equation is linear and possess a 0,0 point, initialize

%differently

bool = 0;

end

end

%if the value fits the above conditions, will follow a different

%procedure for what function it will be defined as

if bool == 0

%take sums necessary for linear function

sum\_xy = 0;

sum\_xx = 0;

for i = 1:s

sum\_xy = sum\_xy + D(i,1)\*D(i,2);

sum\_xx = sum\_xx + D(i,1)\*D(i,1);

end

%initializes new function

f(x,y) = x\*(sum\_xy/sum\_xx);

str = append('Polynomial, y = ',num2str(sum\_xy/sum\_xx));

else

%otherwise initialize function as expected

f(x,y) = X(1,1)+x\*X(2,1);

str = append('Polynomial, y = ',num2str(X(1,1)),'+x\*',num2str(X(2,1)));

%use a loop to add all appropriate parts to the function and the

%string that defines it

for j = 3:m+1

f(x,y) = f(x,y)+X(j,1)\*x^(j-1);

str = append(str,'+',num2str(X(j,1)),'\*x^',num2str(j-1));

end

end

%initializes new\_Y, error and standard deviation vectors

for i = 1:s

new\_Y(i,1) = f(D(i,1),D(i,2));

E(i,1) = (D(i,2)-new\_Y(i,1))^2;

Standard(i,1) = (D(i,2)-average\_y)^2;

end

%initializes r\_2 and calls display function

r\_2 = (sum(Standard)-sum(E))/sum(Standard);

Display(D, new\_Y, str, r\_2);

%rows where the lower and upper limit are found

summ = 0;

%takes the upper and bottom parts of the function

f\_a = f(p\_1,p\_1);

f\_b = f(p\_2,p\_2);

%creates a linearly spaced vector, same thing as in first file

X\_new = linspace(p\_1,p\_2,number+1);

%recording the difference between the points

change = X\_new(2)-X\_new(1);

%running a loop to add together each component

for i = 1:number-1

new\_y = f(p\_1+i\*change,p\_1+i\*change);

summ = summ+new\_y;

end

%integral result

i = ((p\_2-p\_1)/(2\*number))\*(f\_a+2\*summ+f\_b);

end

Display function: used to display the polynomial

function Display(D, new\_Y, symbolic, r\_2)

%preference of floating point values

sympref('FloatingPointOutput',true);

%initializes data, labels and plots it

x = D(:,1);

y = new\_Y;

z = D(:,2);

figure;

plot(x,y,x,z, 'o');

xlabel('x');

ylabel('y');

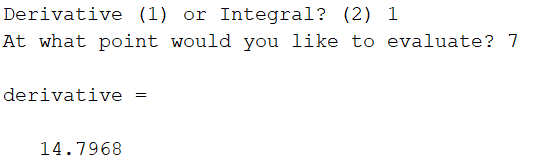
%finalizes legend

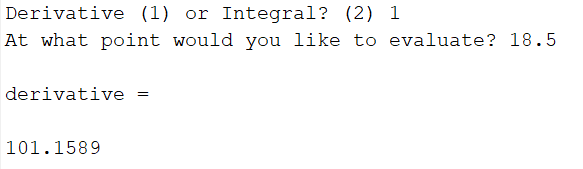
symbolic = append(symbolic,', R^2 = ', num2str(r\_2));

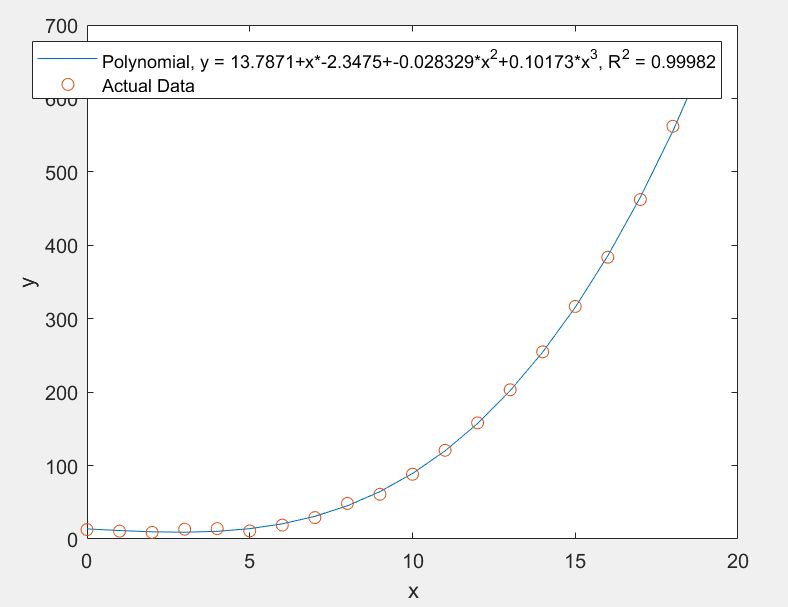
legend(symbolic, 'Actual Data');

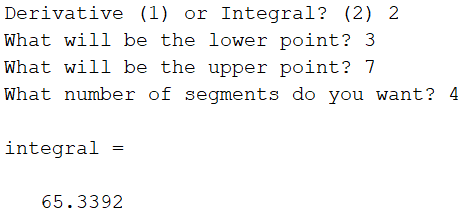
end

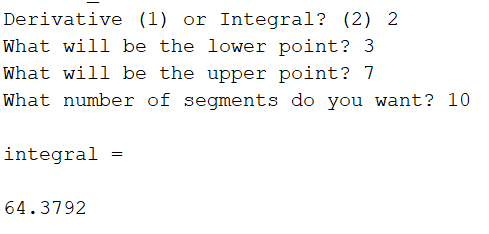
# Results for Question

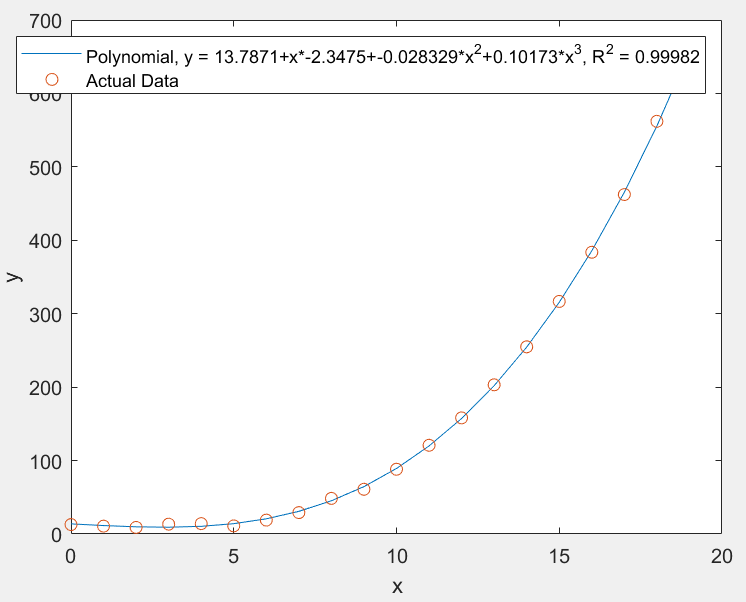












# Conclusions

Using a third order polynomial for regression was a success with the data set used. The test results were similar to the results when done by hand. The program is able to handle data accurately and do adequate approximations for derivation and integration.