Eli Vlahos

Simulation Assignment #3

Ayman El-Hag

16/11/2020

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

[**Results for Question**](#_mb7mnh9rdwbt) **12**

[**Conclusion**](#_byxaxk99wept) **22**

# Introduction

The purpose of this assignment was to create a fitting algorithm that could replicate a polynomial, exponential or saturation model fit. It was also expected to display the test case results and determine which method was most appropriate for each case.

Code

Main function, used to call the various functions

%reads text file

A\_1 = dlmread('test1.txt');

%user chooses model

I = input("1. Polynomial \n2. Exponential \n3. Saturation \nSelect the function to fit your data:");

if I == 1

%take degree then call function

m = input("Degree of polynomial:");

Polynomial(A\_1, m);

elseif I == 2

%calls function

Exponential(A\_1);

elseif I == 3

%calls function

Saturation(A\_1);

end

Polynomial Function

function Polynomial(D, m)

arguments

%specifies the type of input

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

m double;

end

%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);

end

Exponential function

function Exponential(D)

arguments

%specifies the type of input

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

end

%copies an original matrix and deletes all instances of a y being 0

%(this causes error)

Original = D;

rowsToDelete = any(D(:,2) == 0,2);

D(rowsToDelete,:) = [];

%takes size and log

s = size(D);

s = s(1,1);

ln\_D = log(D);

%average and sum initializer

average\_y = 0;

average\_x = 0;

sum\_x = 0;

sum\_y = 0;

sum\_xy = 0;

sum\_xx = 0;

for i = 1:s

%takes sum of each element

sum\_x = sum\_x + D(i,1);

sum\_y = sum\_y + ln\_D(i,2);

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

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

end

%initialize coefficients

b = (s\*sum\_xy-sum\_x\*sum\_y)/(s\*sum\_xx-sum\_x^2);

a = exp((sum\_y-b\*sum\_x)/s);

%takes original or old size, and initializes the new\_Y, E and Standard

%vectors

old\_s = size(Original);

old\_s = old\_s(1,1);

%compares sizes to output warning

if s ~= old\_s

disp('Warning, one of the datapoints was invalid')

end

new\_Y = zeros(old\_s,1);

E = zeros(old\_s,1);

Standard = zeros(old\_s,1);

%Takes floating point preference, initializes symbolic equation,

%function and string that represents it

sympref('FloatingPointOutput',true);

syms x y

f(x,y) = a\*exp(x\*b);

str = append('Exponential, y = ',num2str(a),'\*e^{x\*(',num2str(b), ')}');

%initializes new\_Y, error and standard deviation vectors

for i = 1:old\_s

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

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

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

end

%initializes r\_2 and calls display function

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

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

end

Saturation function

function Saturation(D)

arguments

%specifies the type of input

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

end

%copies an original matrix and deletes all instances of a y or x being 0

%(this causes error)

Original = D;

rowsToDelete = any(D == 0,2);

D(rowsToDelete,:) = [];

%takes size and initializes averages and sums

s = size(D);

s = s(1,1);

average\_y = 0;

average\_x = 0;

sum\_x = 0;

sum\_y = 0;

sum\_xy = 0;

sum\_xx = 0;

%takes sum

for i = 1:s

sum\_x = sum\_x + 1/D(i,1);

sum\_y = sum\_y + 1/D(i,2);

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

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

end

% initializes coefficients

a\_0 = (s\*sum\_xy-sum\_x\*sum\_y)/(s\*sum\_xx-sum\_x^2);

a = 1/((sum\_y-a\_0\*sum\_x)/s);

b = a\_0\*a;

%takes original or old size, and initializes the new\_Y, E and Standard

%vectors

old\_s = size(Original);

old\_s = old\_s(1,1);

%compares sizes to output warning

if s ~= old\_s

disp('Warning, one of the datapoints was invalid')

end

new\_Y = zeros(old\_s,1);

E = zeros(old\_s,1);

Standard = zeros(old\_s,1);

sympref('FloatingPointOutput',true);

syms x y

f(x,y) = (a\*x)/(b+x);

str = append('Saturation, y = ',num2str(a),'\*x/(',num2str(b), '+x)');

%initializes new\_Y, error and standard deviation vectors

for i = 1:old\_s

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

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

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

end

%initializes r\_2 and calls display function

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

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

end

All functions called the display function with the necessary parameters. The parameters were the original matrix of data, D, the new Y values, the symbolic equation and the R^2 value.

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

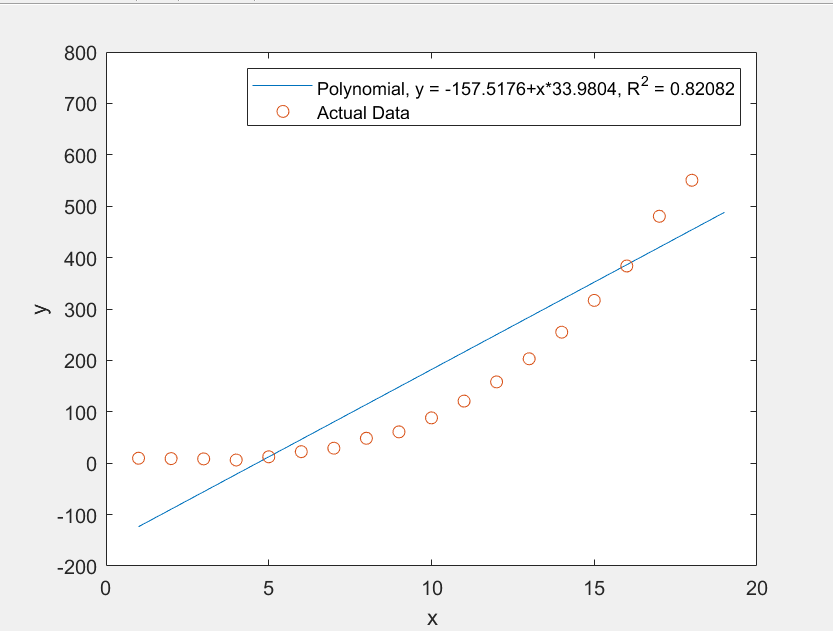
symbolic = append(symbolic,', R^2 = ', num2str(r\_2));

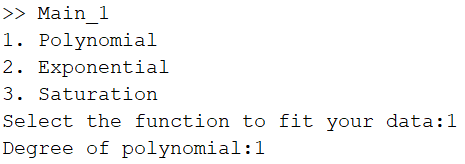
legend(symbolic, 'Actual Data');

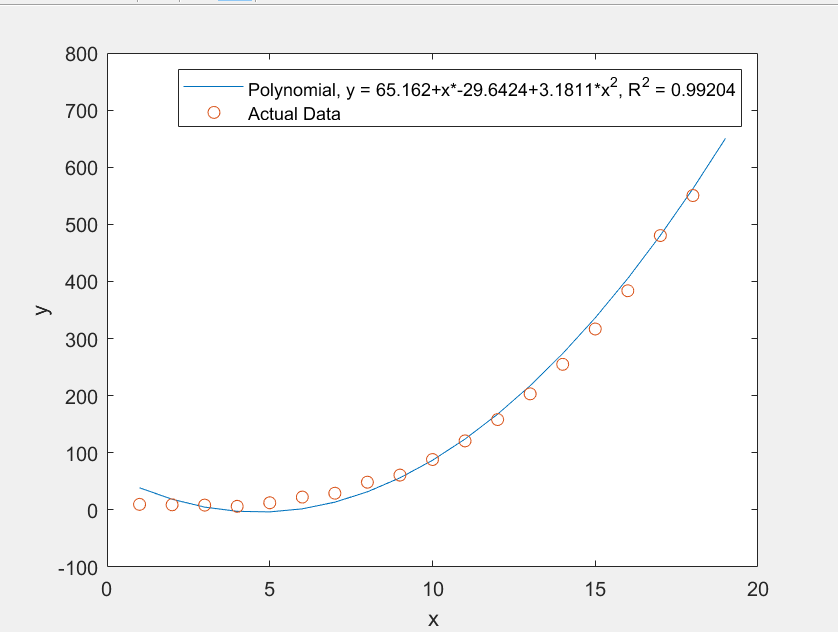
end

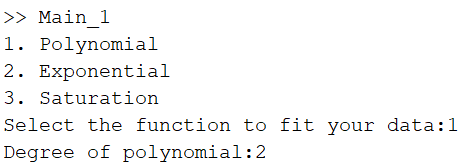
# Results for Question

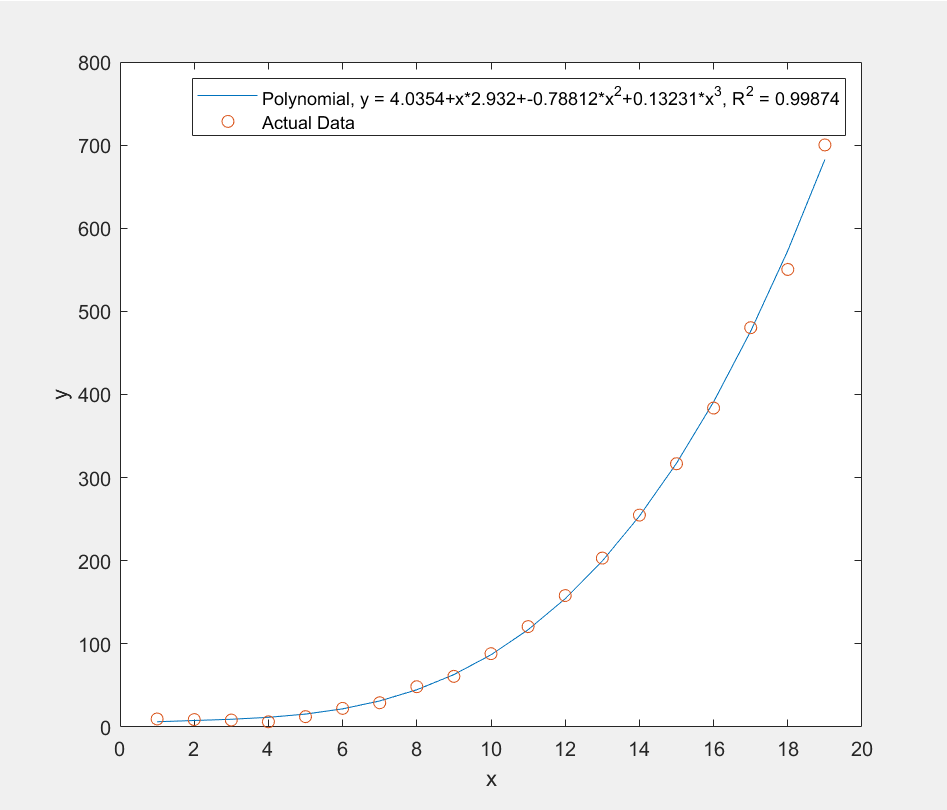
Here are the five graphs for the first text file

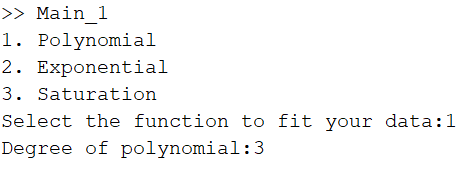


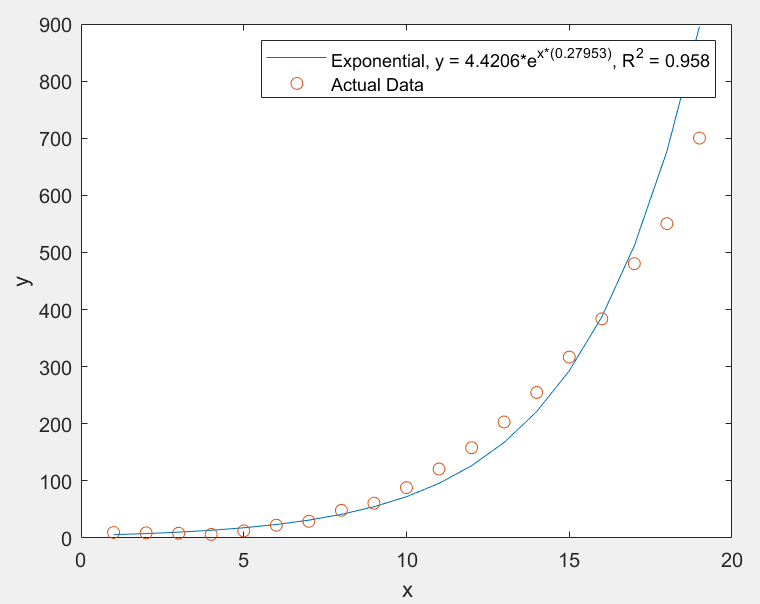


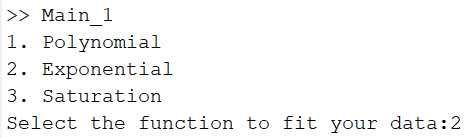


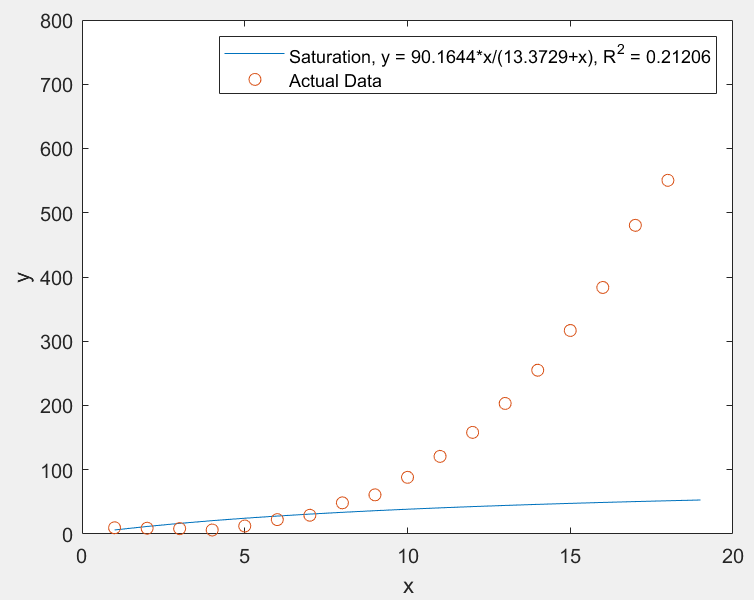


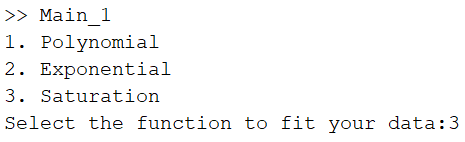






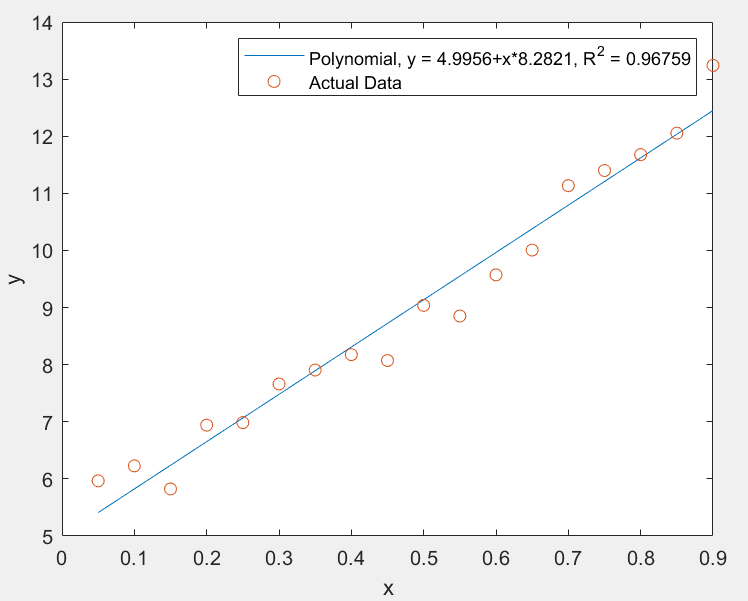


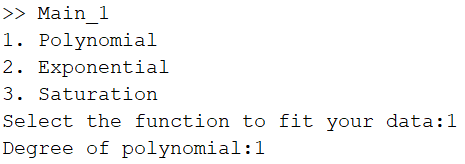


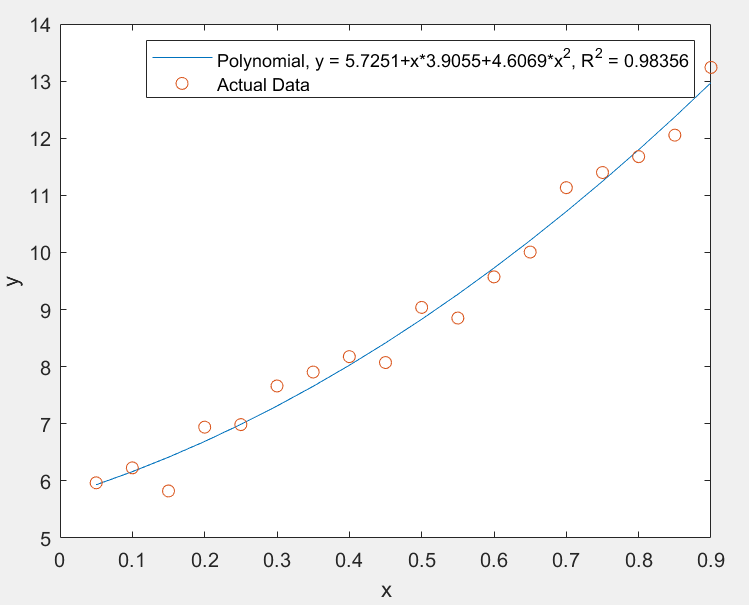


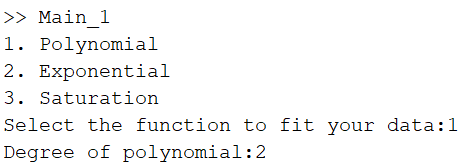
The best fit for the first set of data is a Polynomial to the degree of three. This can be seen as it has the highest R^2 value at 0.99874, closest to one.

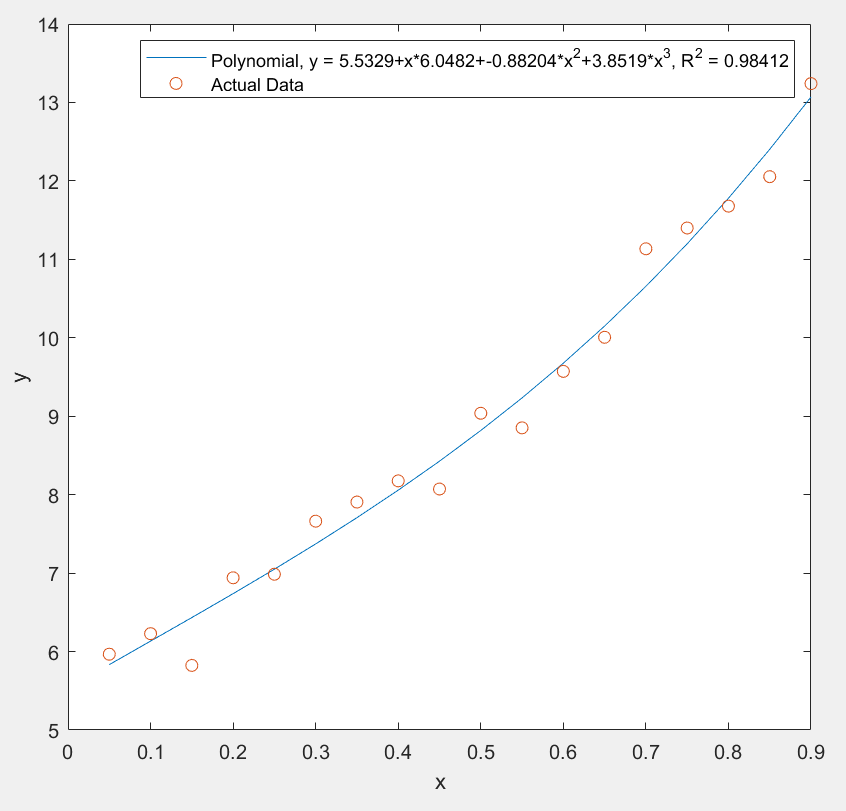
This is for the second text file

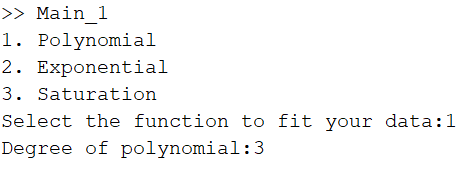


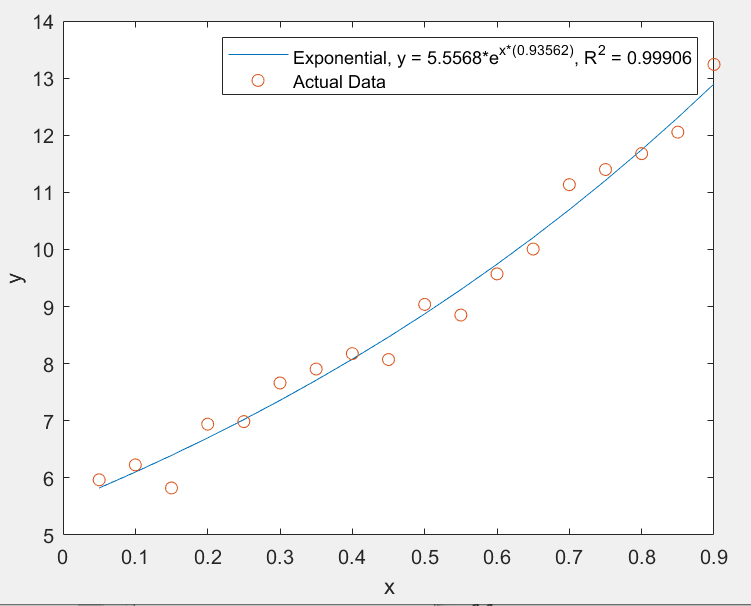


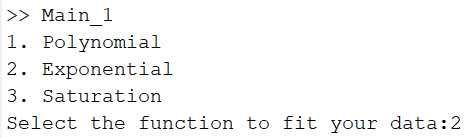


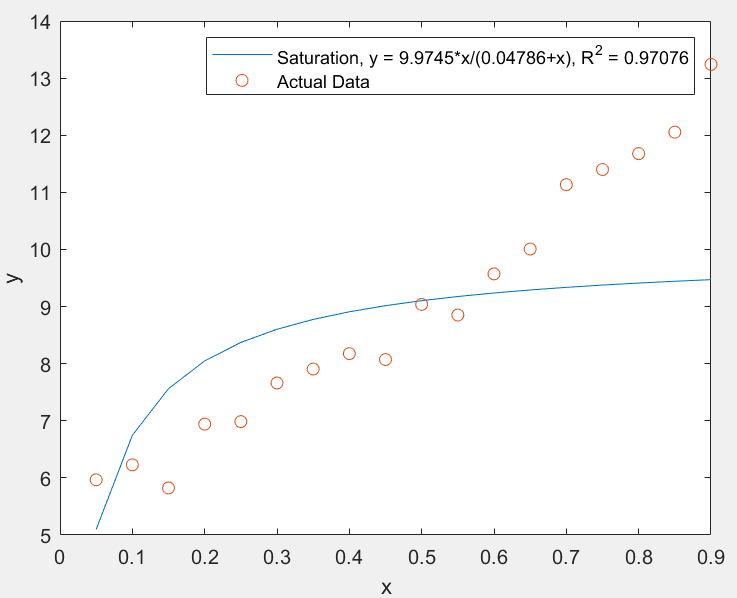


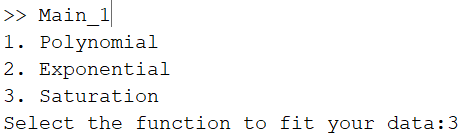












The best fit for the second set of data is an Exponential function. This can be seen as it has the highest R^2 value at 0.99906, closest to one.