**What is Polynomial Curve Fitting?**

Polynomial functions are functions which involve non-negative integer powers of x. These are inclusive of functions such as quadratic functions, cubic functions and the rest of the power progressions. The process of polynomial curve fitting is the process of constructing a mathematical function of best fit, to a series of data points, in such a way that the curve is representative of the majority of the data points present. The key is to identify a “smooth” curve of best fit. This tutorial enables the reader to determine such a curve from pre-existing data. The software used to determine this outcome is MATLAB.

# Polynomial Curve Fitting

This example shows how to fit a polynomial curve to a set of data points using the polyfit function. You can use polyfit to find the coefficients of a polynomial that fits a set of data in a least-squares sense using the syntax

p = polyfit(x,y,n),

where:

* x and y are vectors containing the x and y coordinates of the data points
* n is the degree of the polynomial to fit

Create some x-y test data for five data points.

x = [1 2 3 4 5];

y = [5.5 43.1 128 290.7 498.4];

Use polyfit to find a third-degree polynomial that approximately fits the data.

p = polyfit(x,y,3)

p = 1×4

-0.1917 31.5821 -60.3262 35.3400

-0.2x^3 +31.6 x^2 -60.3 x +35.3

After you obtain the polynomial for the fit line using polyfit, you can use polyval to evaluate the polynomial at other points that might not have been included in the original data.

Compute the values of the polyfit estimate over a finer domain and plot the estimate over the real data values for comparison. Include an annotation of the equation for the fit line.

x2 = 1:1:5;

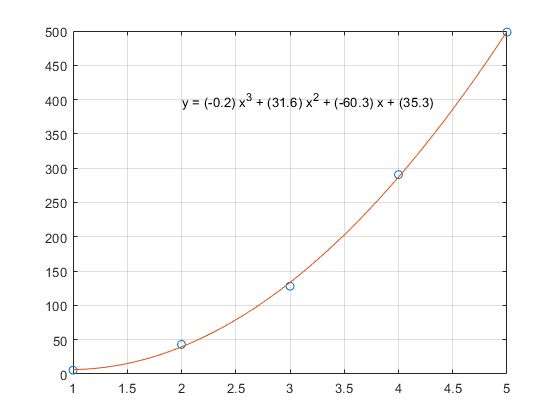
y2 = polyval(p,x2);

plot(x,y,'o',x2,y2)

grid on

s = sprintf('y = (%.1f) x^3 + (%.1f) x^2 + (%.1f) x + (%.1f)',p(1),p(2),p(3),p(4));

text(2,400,s)



Other variations in polyfit :

[[p](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-p),[S](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-S)] = polyfit([x](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-x),[y](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-y),[n](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-n)) also returns a structure S that can be used as an input to [polyval](https://www.mathworks.com/help/matlab/ref/polyval.html) to obtain error estimates.

[[p](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-p),[S](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-S),[mu](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-mu)] = polyfit([x](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-x),[y](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-y),[n](https://www.mathworks.com/help/matlab/ref/polyfit.html#bue6sxq-1-n)) also returns mu, which is a two-element vector with centering and scaling values. mu(1) is mean(x), and mu(2) is std(x). Using these values, polyfit centers x at zero and scales it to have unit standard deviation,

Example 2 :

## Objective: Fit Polynomial to Trigonometric Function

The question highlighted below is asking the user to generate a ten point curve using a sine function on a data set. The intervals for the points have been denoted. For clarity the code itself will be highlighted in blue, while the instructions will remain as normal.

Generate 10 points equally spaced along a sine curve in the interval [0,4\*pi].

x = linspace(0,4\*pi,10);

y = sin(x);

Use polyfit to fit a 7th-degree polynomial to the points.

p = polyfit(x,y,7);

Evaluate the polynomial on a finer grid and plot the results.

x1 = linspace(0,4\*pi);

y1 = polyval(p,x1);

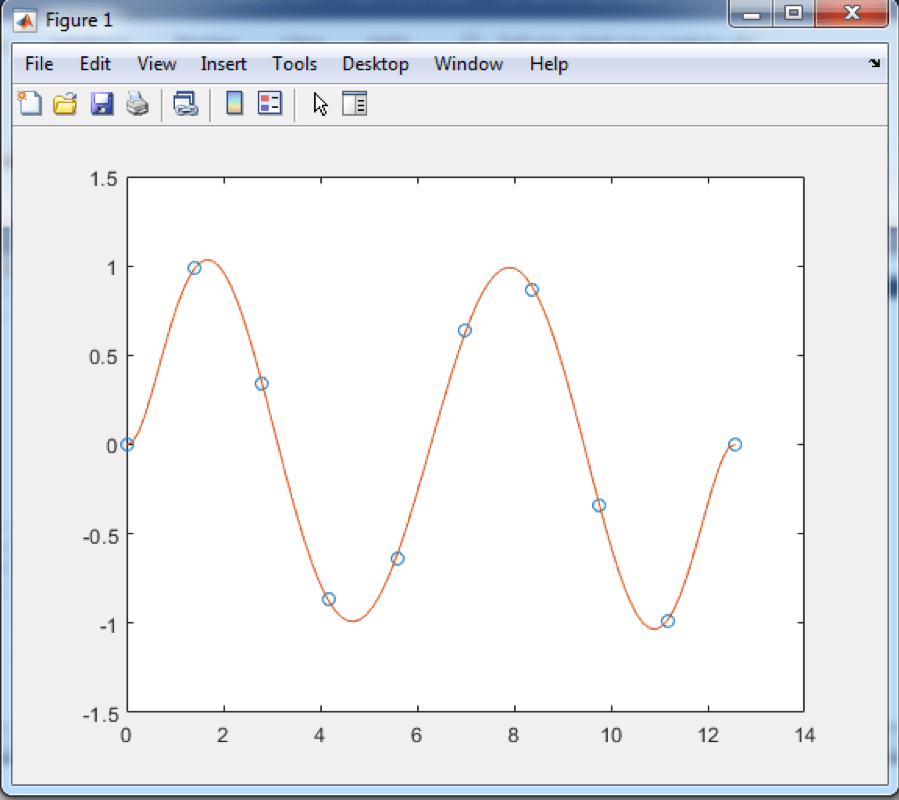
figure

plot(x,y,'o')

hold on

plot(x1,y1)

hold off



The result is a beautiful, sine curve as desired by the instructions given.

OBJECTIVE:

Create a vector of 5 equally spaced points in the interval [0,1],

and evaluate

https://sunglass.io/wp-content/uploads/2019/03/matlabpolyfitformula.png

at those points.

x = linspace(0,1,5);

y = 1./(1+x);

Fit a polynomial of degree 4 to the 5 points. In general, for n points, you can fit a polynomial of degree n-1 to exactly pass through the points.

p = polyfit(x,y,4);

Evaluate the original function and the polynomial fit on a finer grid of points between 0 and 2.

x1 = linspace(0,2);

y1 = 1./(1+x1);

f1 = polyval(p,x1);

Plot the function values and the polynomial fit in the wider interval [0,2], with the points used to obtain the polynomial fit highlighted as circles. The polynomial fit is good in the original [0,1] interval, but quickly diverges from the fitted function outside of that interval.

figure

plot(x,y,'o')

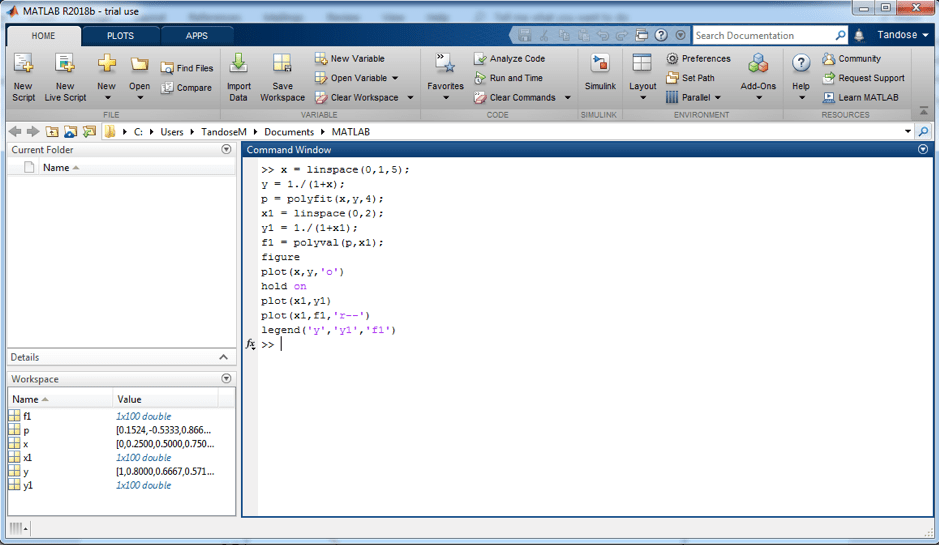
hold on

plot(x1,y1)

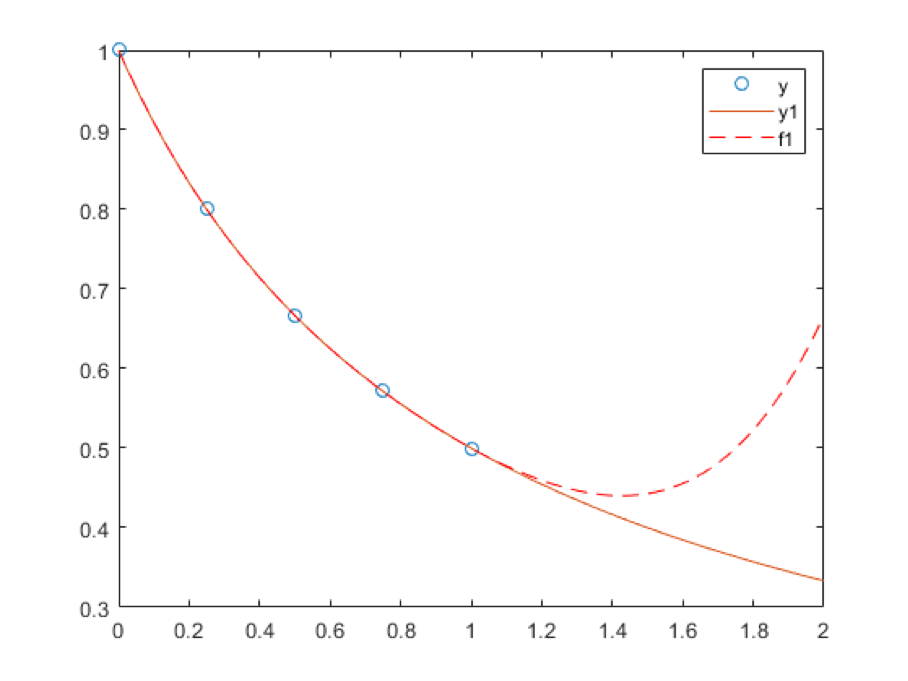
plot(x1,f1,'r--')

legend('y','y1','f1')

The following is the code entry into the MATLAB System:



The system output is shown below:



Tryouts :

Example 2 : replace the sin() with cos() - apply polyfit.

NON-LINEAR ROOTS

|  |  |
| --- | --- |
| **Linear Equations** | **Non-Linear Equations** |
| It forms a straight line or represents the equation for the straight line | It does not form a straight line but forms a curve. |
| It **has only one degree**. Or we can also define it as an equation having the maximum degree 1. | A nonlinear equation **has the degree as 2 or more than 2**, but not less than 2. |
| All these equations form a straight line in XY plane. These lines can be extended to any direction but in a straight form. | It forms a curve and if we increase the value of the degree, the curvature of the graph increases. |
| The general representation of linear equation is;  **y = mx +c**  Where x and y are the variables, m is the slope of the line and c is a constant value. | The general representation of nonlinear equations is;  **ax2 + by2 = c**  Where x and y are the variables and a,b and c are the constant values |
| **Examples:**   * 10x = 1 * 9y + x + 2 = 0 * 4y = 3x * 99x + 12 = 23 y | **Examples:**   * x2+y2 = 1 * x2+ 12xy + y2 = 0 * x2+x+2 = 25 |

**Note:**

The linear equation has only one variable usually and if any equation has two variables in it, then the equation is defined as a Linear equation in two variables. For example, 5x + 2 = 1 is Linear equation in one variable. But 5x + 2y = 1 is a Linear equation in two variables.

Let us see some examples based on these concepts.

Solved Examples

**Example:** Solve the linear equation 3x+9 = 2x + 18.

**Solution:** Given, 3x+9 = 2x + 18

⇒ 3x – 2x = 18 – 9

⇒ x = 9

**Example:** Solve the nonlinear equation x+2y = 1 and x = y.

**Solution:** Given, x+2y = 1

x = y

By putting the value of x in the first equation we get,

⇒ y + 2y = 1

⇒ 3y = 1

⇒ y = ⅓

∴ x = y = ⅓

# fzero

Root of nonlinear function

## Syntax

[x = fzero(fun,x0)](https://www.mathworks.com/help/matlab/ref/fzero.html#d123e474752)

[x = fzero(fun,x0,options)](https://www.mathworks.com/help/matlab/ref/fzero.html#d123e474780)

### Root Starting From One Point : ( sin (3))

fun = @sin; % function

x0 = 3; % initial point

x = fzero(fun,x0)

x = 3.1416

### Root Starting From an Interval

Find the zero of cosine between 1 and 2.

fun = @cos; % function

x0 = [1 2]; % initial interval

x = fzero(fun,x0)

x = 1.5708

Note that cos(1) and cos(2) differ in sign.

### Root of a Function Defined by a File

Find a zero of the function f(x) = x3 – 2x – 5.

First, write a file called f.m.

function y = f(x)

y = x.^3 - 2\*x - 5;

Save f.m on your MATLAB® path.

Find the zero of f(x) near 2.

fun = @f; % function

x0 = 2; % initial point

z = fzero(fun,x0)

output :

z =

2.0946

Since f(x) is a polynomial, you can find the same real zero, and a complex conjugate pair of zeros, using the roots command.

roots([1 0 -2 -5])

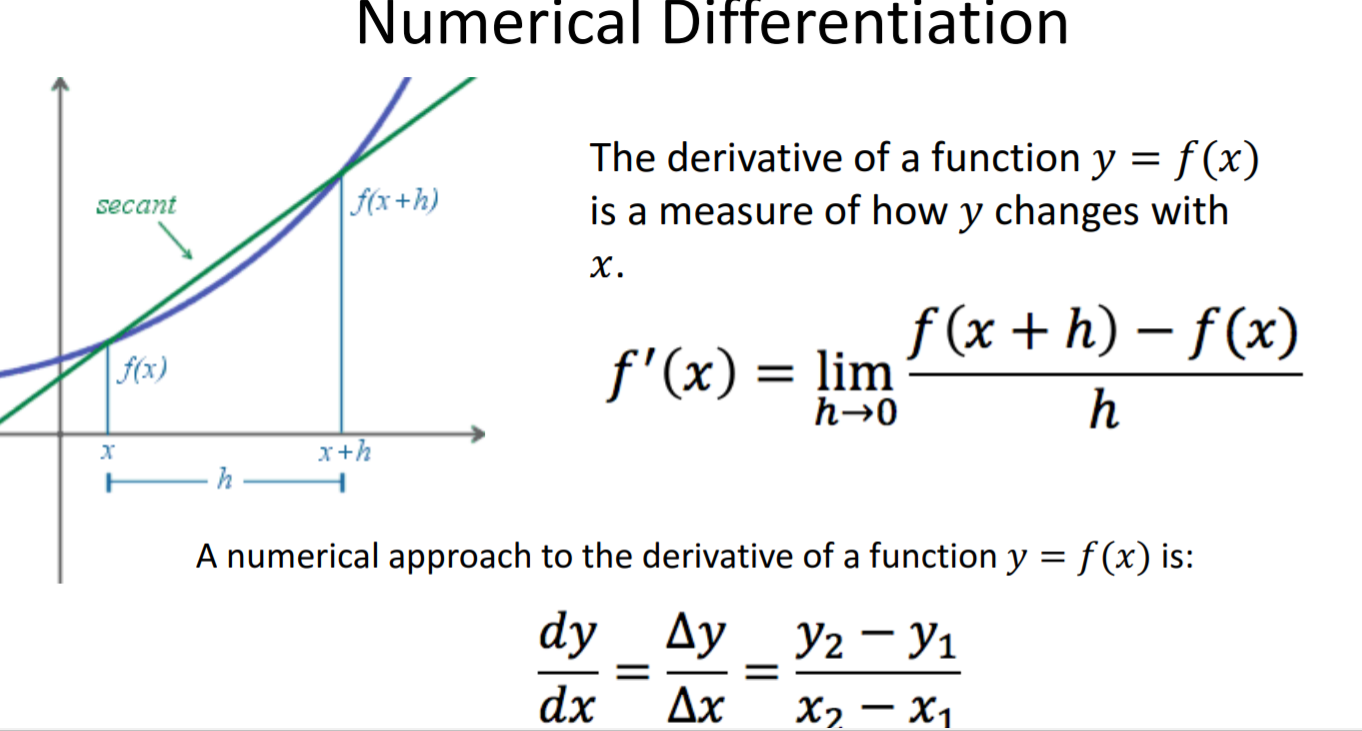
ans =

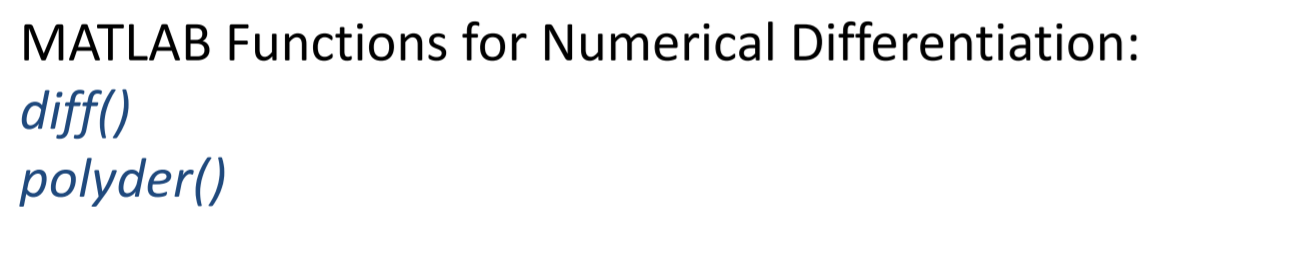
2.0946

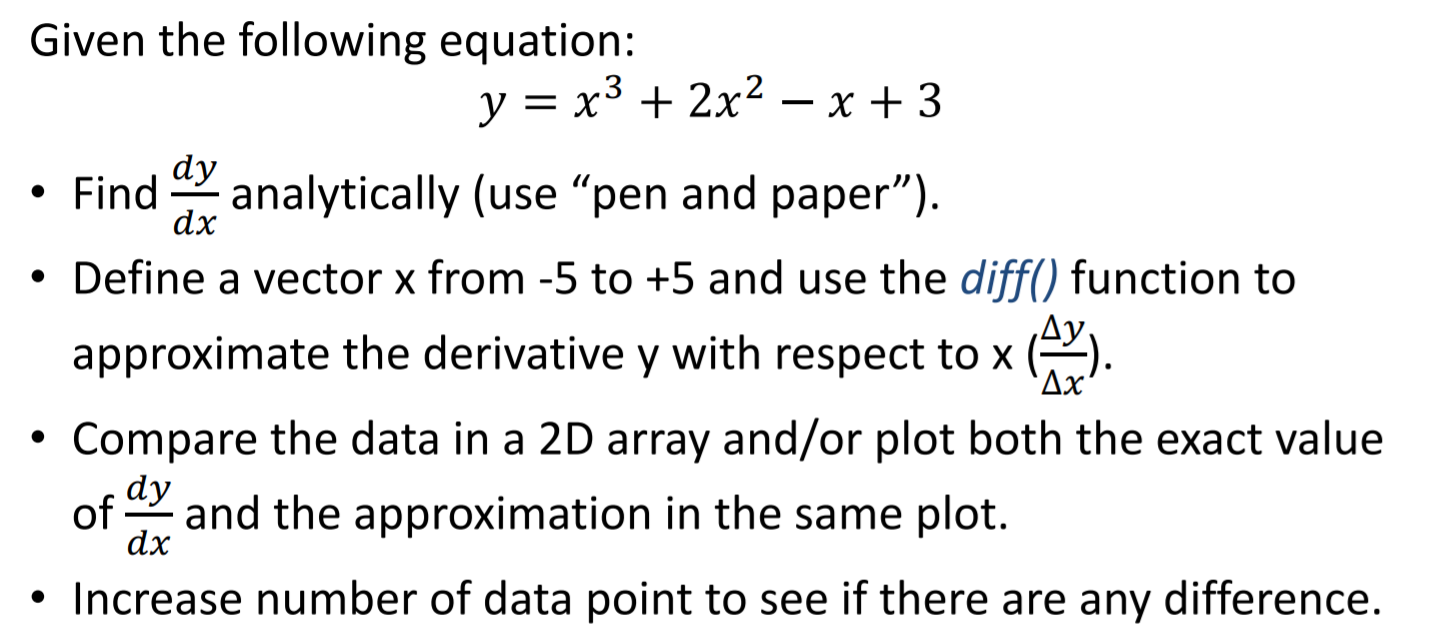
-1.0473 + 1.1359i

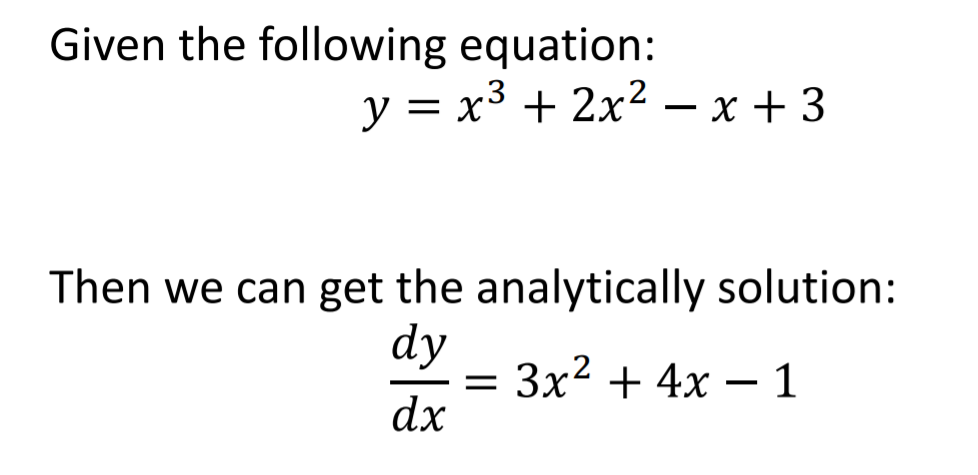
-1.0473 - 1.1359i

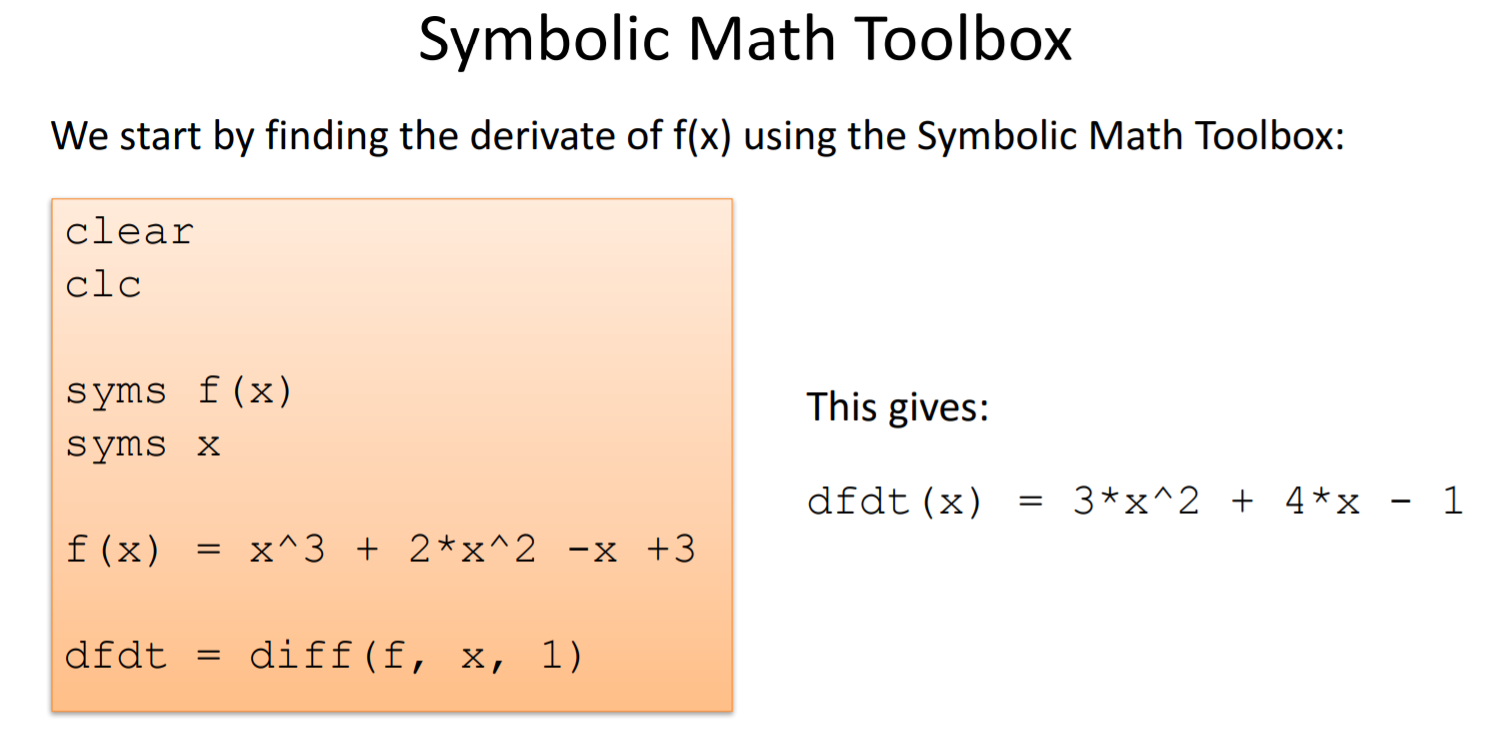
Numerical differentiation











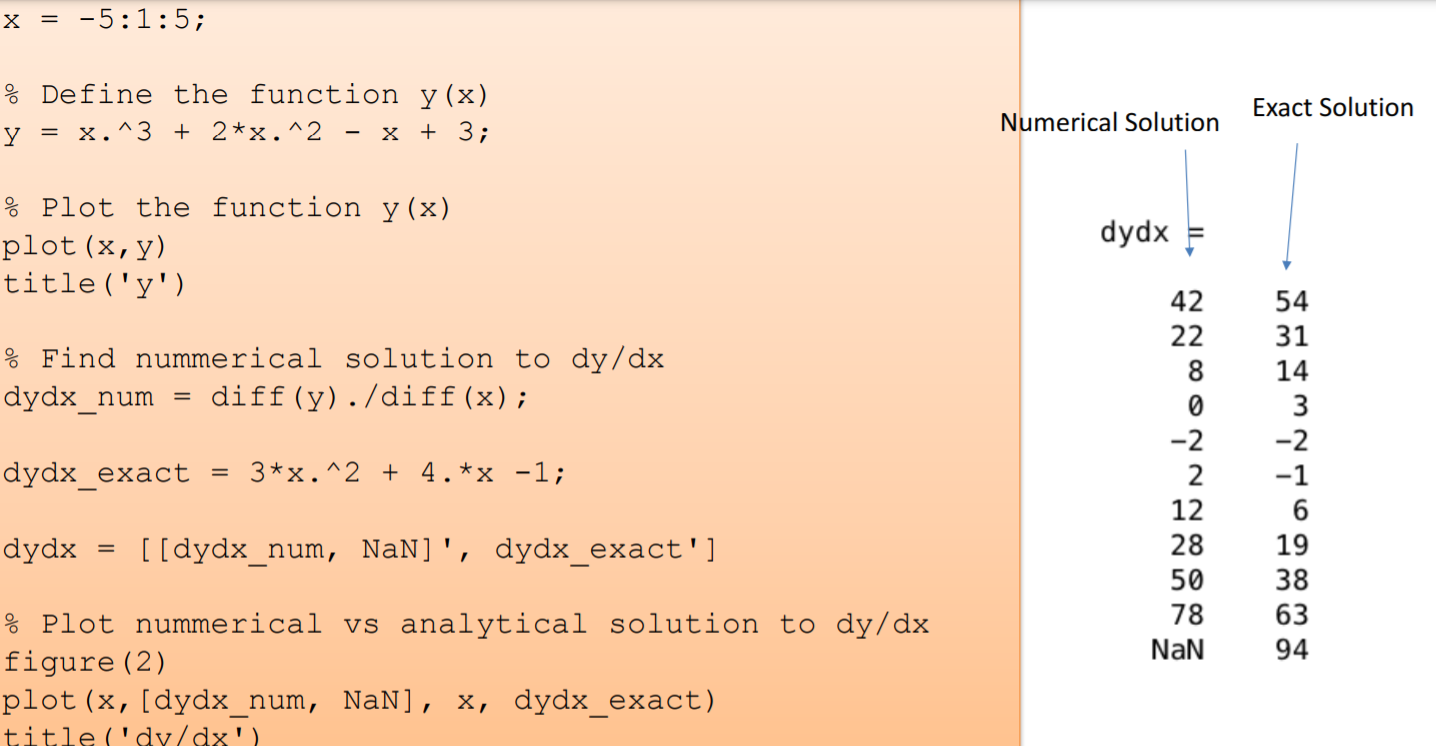
Create a vector, then compute the differences between the elements.

X = [1 1 2 3 5 8 13 21];

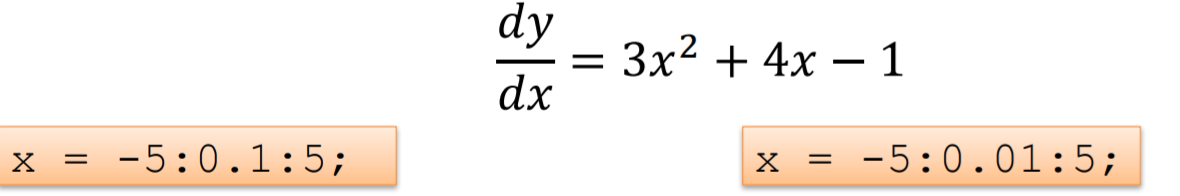
Y = diff(X)

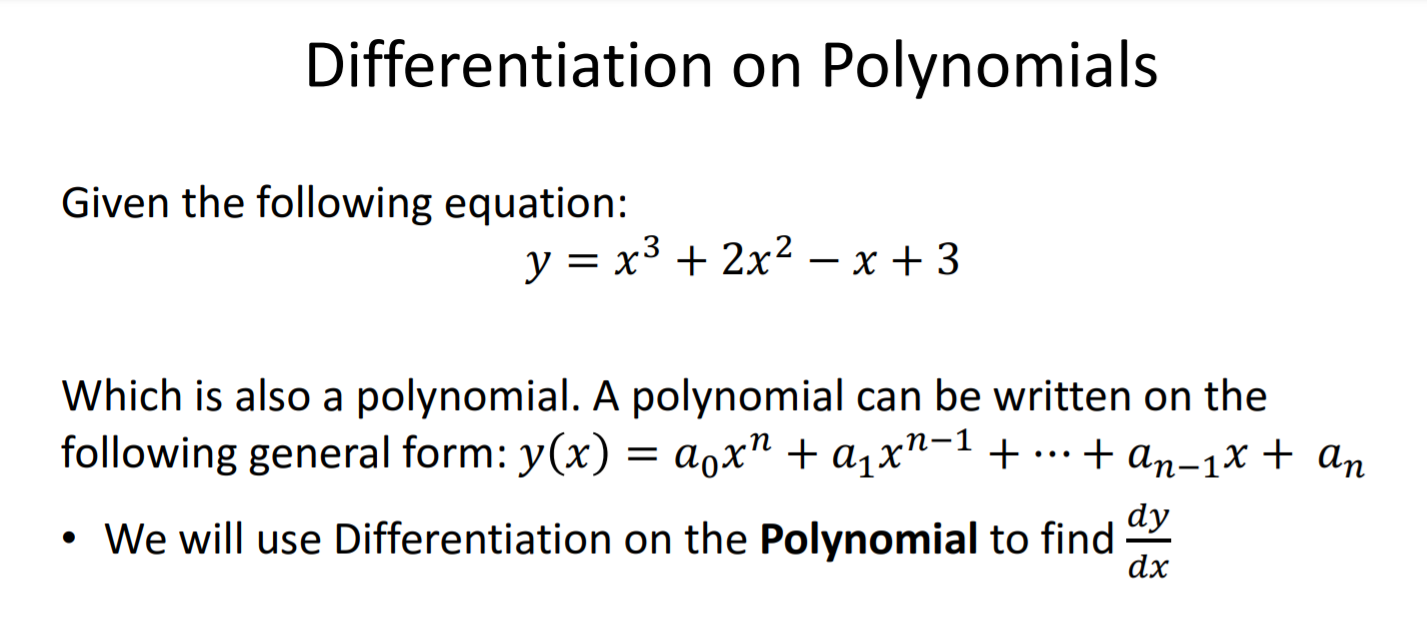
Y = 1×7

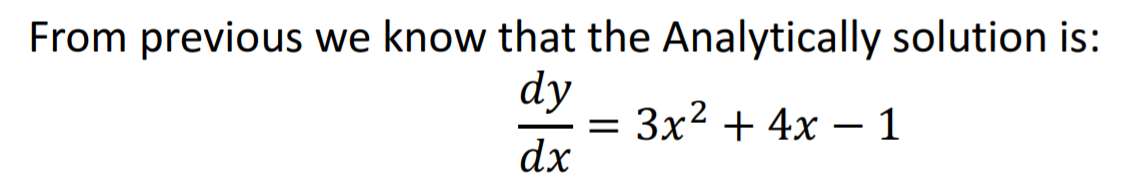
0 1 1 2 3 5 8

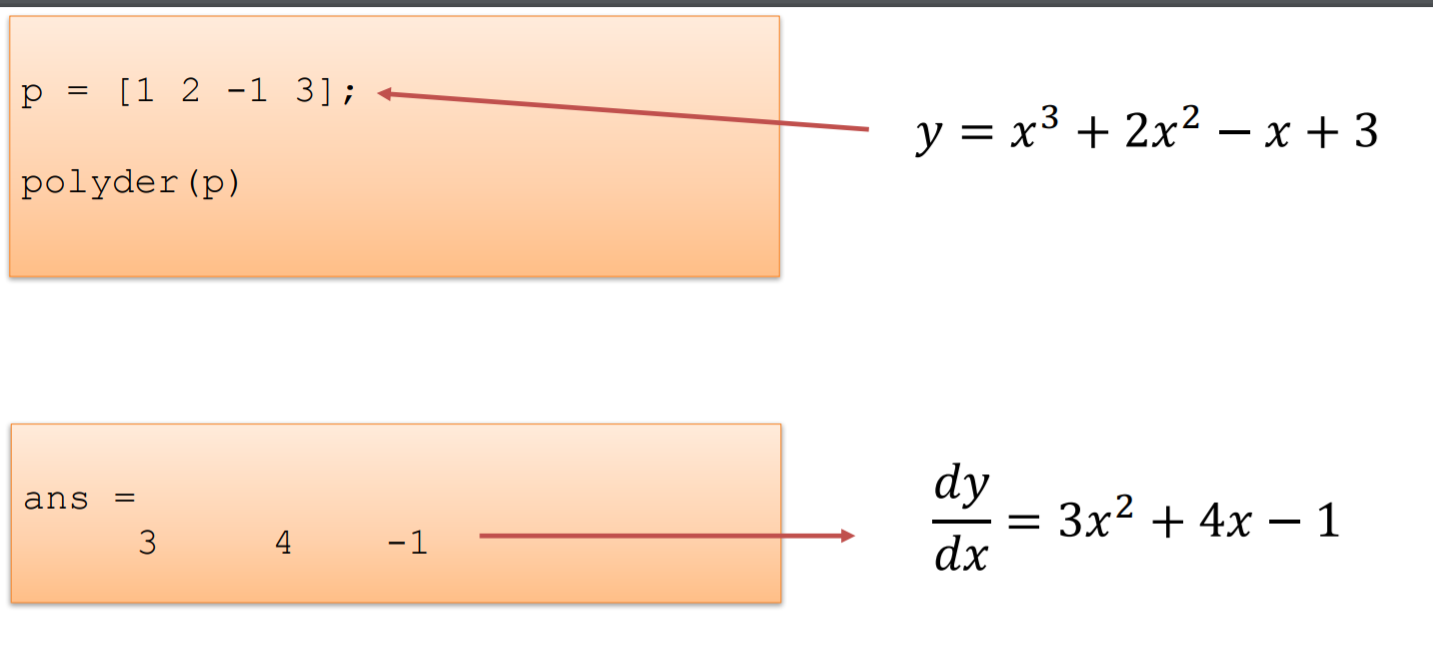


TRYOUTS:









# Runge-Kutta 2nd order method to solve Differential equations

Given the following inputs: 

1. An ordinary [differential equation](https://www.geeksforgeeks.org/second-order-linear-differential-equations/) that defines the value of **dy/dx** in the form **x** and **y**.
2. Initial value of y, i.e., **y(0)**.

The task is to find the value of unknown function y at a given point x, i.e. **y(x)**.

