1) Taylor series for Single Variable:

Syntax

```
taylor(f,var)
taylor(f,var,a)
```

approximates `f` with the Taylor series expansion of `f` at the point `var = a`.

Example 01

```
syms x
T1=taylor(exp(x)) %Maculurian Series
T2=taylor(sin(x))
T3=taylor(log(sin(x)),x,2) % Taylor series at x=2
T4=taylor(log(x),x,1)
```

Example 02

```
syms x
T1=taylor(exp(x)) %Maculurian Series
T2=taylor(sin(x))
T3=taylor(log(sin(x)),x,2) % Taylor series at x=2
T4=taylor(log(x),x,1)
```

Taylor series for Two-Variable:

```
syms x y
f = y*exp(x - 1) - x*log(y);
T = taylor (f, [x y], [1 1],'Order',3)
```

Plotting Taylor series for different order

```
syms x
f = sin(x);
T6 = taylor (f, x);
T8 = taylor (f, x,'Order',8);
T10 = taylor (f, x,'Order',10);
fplot ([T6 T8 T10 f])
xlim ([-4 4])
grid on
legend(O(x^6)', O(x^8)', O(x^{10})',
'sin(x)')
title('Taylor Series Expansion')
```

2) Newton-Raphson method:

Single Variable:

```
clc;
clear;
syms x
f(x)= x*sin(x)+cos(x);
df(x)= diff (f, x);
a=2; b=3;
if f(a)*f(b)<0
x0=(a+b)/2;
else
fprintf ("change interval value")
end
```

```
 %x0=input ("enter the initial approximation:");
for n=1: 5000
    xn= x0-(f(x0)/df(x0));
fprintf ('\n The required root at %d is: %f', n, xn);
if abs(f(xn)) <=0.000001
      break;
    else
        x0=xn;
    end
end
fplot(f);
xlim([-10 10])
```

3) Newton-Raphson method for Multivariable:

```
clc;
syms x y h k
f (x, y) =x^2-y^2-4;
g (x, y) =y^2+x^2-16;

dfx = diff (f, x);%derivative
dfy = diff(f,y);
dgx = diff(g,x);
dgy = diff(g,y);

 x0=input ("Enter the initial value of x:");
 y0=input ("Enter the initial value of y:");

 for i=1:1000
            EQ1=dfx (x0, y0) *h+dfy (x0, y0) *k+f (x0, y0);
            EQ2=dgx (x0, y0) *h+dgy (x0, y0) *k+g (x0, y0);

            S=solve ([EQ1==0, EQ2==0],[h,k]);
            xn=x0+S.h;
            yn=y0+S.k;
fprintf ("The required root at %d is %f \n %f\n",i,xn,yn);
    if abs (f(xn, yn))<=0.00001
            break;
            else
                x0=xn;
                y0=yn;
    end
 end
```

4) Euler's method

```
%Program to solve ordinary differential equations using Euler's method
clc;
close;
clear;
% Define the function f(x,y) that represents the differential equation
f = @(x,y) x + y;
% Define the initial conditions
x0 = 0; % initial x value
y0 = 1; % initial y value
xn = 1; % final x value
% Define the step size and the range of x values to approximate
h = 0.1; % step size
x = x0:h:xn; % range of x values
% Initialize the y vector with the initial value y0
y(1) = y0;
```

```
% Use Euler's method to approximate the solution
for i = 2:length(x)
    y(i) = y(i-1) + h * f(x(i-1), y(i-1));
    disp(y(i))
end
```

5) Modified Euler's method

```
%Program to solve ordinary differential equations using modified Euler's
method
clc;
clear;
close;
x0 = 0;
y0 = 1;
h = 0.1; % Step size
xn = 0.1; % Final value
% Define the function y' = f(x,y)
f = @(x,y) -y^2+ x ;
 % Implement Euler's modified method
x = x0:h:xn;
y = zeros(size(x));
y(1) = y0;
 for i = 2:length(x)
    % Predictor step
    y(i) = y(i-1) + h*f(x(i-1), y(i-1));
    fprintf('Required value by Euler''s method (Predictor method) %f \n :'
, y(i));
    %disp(y(i))
    x(i)=x(i-1)+h;
    y(i,1)=y(i);
    % Corrector step
    for j=2:20
    y(i,j) = y(i-1) + (h/2)*(f(x(i-1), y(i-1)) + f(x(i), y(i,j-1)));
   disp(y(i,j));
    if abs(y(i,j)-y(i,j-1))<0.00001
        y(i)=y(i,j);
        fprintf('Required value by Euler''s modified method (Corrector
method) %f \n :' , y(i));
        break;
    end
    end
end
```

6) Runge Kutta 4th order method

```
%Program to solve ordinary differential equations using RK 4th order method
clc;
clear;
f = @(x,y) x+y;
%Define the interval
x0=1;
y0=1;
h=0.2;
xn=2;
n=(xn-x0)/h;
x = x0:h:xn;
y = zeros(size(x));
y(1) = y0;
x(1)=x0;
```

```
 for i = 2:length(x)
    k1=h*f(x(i-1),y(i-1));
k2=h*f(x(i-1)+h/2,y(i-1)+k1/2);
k3=h*f(x(i-1)+h/2,y(i-1)+k2/2);
k4=h*f(x(i-1)+h,y(i-1)+k3);
      y(i)=y(i-1)+(k1+2*k2+2*k3+k4)/6;
      x(i)=x(i-1)+h;
    fprintf("The value of y(%f) is %f\n", x(i), y(i));
end
```