SAYEM-1901152

1.Generalized Differentiation code :

```
% Clear the command window and workspace
%clc;
clear all;

% Input section for curve fitting
x = input('Enter the input x = ');
y = input('Enter the input y = ');
N = input('Enter the order of curve (degree of polynomial) = ');

% Curve fitting using Vandermonde matrix
temp = length(x);
A = zeros(N, N);
B = zeros(N, 1);

for i = 1:N
    for j = 1:N
        A(i, j) = sum(x.^(i + j - 2));
    end
end

for t = 1:N
    B(t, 1) = sum((x.^(t - 1)) .* y);
end

U = A \ B;

% Display the fitted equation
display('The system equation is:');
fprintf('\n');
for g = N:-1:1
    fprintf('+(%.4fa^%d)', U(g), g - 1);
end
fprintf('\n');

% Plotting section
P = flip(U);
X = linspace(x(1), x(temp), 100);
Y = polyval(P, X);
plot(X, Y);
hold on
plot(x, y, '*');

% Differentiation section
f = @(x) polyval(P, x); % Convert polynomial coefficients into a function

h = input('Enter the value of h for differentiation: ');
a = input('Enter the value of x at which you want to differentiate: ');
```

```
% Differentiation using forward, backward, and central difference methods
df = (f(a + h) - f(a)) / h;
db = (f(a) - f(a - h)) / h;
dc = (f(a + h) - f(a - h)) / (2 * h);

% Output section
fprintf('The forward difference value = %.4f\n', df);
fprintf('The backward difference value = %.4f\n', db);
fprintf('The central difference value = %.4f\n', dc);
```

2.Generalized Integration code :

```
%clc;
clear all;

% Input section for curve fitting
x = input('Enter the x coordinates: ');
y = input('Enter the y coordinates: ');
N = input('Enter the number of fitting (degree of polynomial): ');

% Curve fitting using Vandermonde matrix
n = length(x);
A = zeros(N, N);
for i = 1:N
    for j = 1:N
        A(i, j) = sum(x.^(i + j - 2));
    end
end

B = zeros(N, 1);
for k = 1:N
    B(k) = sum((x.^(k - 1)) .* y);
end

coefficients = A \ B;

% Display the fitted equation
fitted_equation = '';
for g = N:-1:1
    fitted_equation = strcat(fitted_equation, sprintf('+(%.4fx^%d)',
coefficients(g), g - 1));
end
fprintf('Fitted equation: y = %s\n', fitted_equation(2:end));

% Plotting section
P = flip(coefficients);
X = linspace(x(1), x(n), 100);
Y = polyval(P, X);
plot(X, Y)
hold on
plot(x, y, '*')

% Integration section
f = @(x) polyval(P, x);
```

```
a = input('Enter the lower limit: ');
b = input('Enter the upper limit: ');
N = input('Enter the number of trapezoids for integration: ');

% Trapezoidal rule
h = (b - a) / N;
sum_trapz = 0;
for i = 1:N - 1
    sum_trapz = sum_trapz + f(a + i * h);
end
int_trapz = (h / 2) * (f(a) + 2 * sum_trapz + f(b));
fprintf('Result Using Trapezoidal Rule is: %.8f\n', int_trapz);

% Simpson's 1/3 rule
odd_sum_simp13 = 0;
even_sum_simp13 = 0;
for j = 1:2:N - 1
    odd_sum_simp13 = odd_sum_simp13 + f(a + j * h);
end
for k = 2:2:N - 2
    even_sum_simp13 = even_sum_simp13 + f(a + k * h);
end
int_simp13 = (h / 3) * (f(a) + 4 * odd_sum_simp13 + 2 * even_sum_simp13 +
f(b));
fprintf('Result Using Simpson 1/3 Rule is: %.8f\n', int_simp13);

% Simpson's 3/8 rule
sum_p1_simp38 = 0;
sum_p2_simp38 = 0;
for m = 1:3:N - 2
    sum_p1_simp38 = sum_p1_simp38 + f(a + m * h);
end
for m = 3:3:N - 1
    sum_p2_simp38 = sum_p2_simp38 + f(a + m * h);
end
int_simp38 = (3 * h / 8) * (f(a) + 3 * sum_p1_simp38 + 2 * sum_p2_simp38 +
f(b));
fprintf('Result Using Simpson 3/8 Rule is: %.8f\n', int_simp38);
```

3.Generalized bisection method Code

```
close all;
clear all;

fn = input('Enter the function (in terms of x): ');
a = input('Enter the value of a (positive): ');
b = input('Enter the value of b (negative): ');
e = input('Enter the tolerance level: ');
n = input('Enter the maximum iteration number: ');

if fn(a) * fn(b) < 0
    for i = 1:n
        c = (a + b) / 2;
        if abs(fn(a) - fn(c)) <= e || abs(fn(b) - fn(c)) <= e
            break;
        end

        if fn(a) * fn(c) < 0
            b = c;
        else
            a = c;
        end
    end

    fprintf('The root is %f\n', c);
else
    fprintf('No root found in the given interval.\n');
end
```

4.Generalize False position Method:

```
%clc
clear all
close all

fn = input('Enter the function (in terms of x): ');
a = input('Enter the value of a (positive): ');
b = input('Enter the value of b (negative): ');
e = input('Enter the tolerance level: ');
n = input('Enter the maximum iteration number: ');

if fn(a)*fn(b) < 0
    for i=1:n
    c=(b*fn(a)-a*fn(b))/(fn(a)-fn(b));
    if abs(fn(a)-fn(c))<=e || abs(fn(b)-fn(c))<= e
        break
    end
    if fn(a)*fn(c) < 0
        b=c ;
    else
        a=c ;
    end
    end
    fprintf('The root is %f',c);
else
    fprintf('Enter initial value again')
 end
```

5.Generalized newton Rapson Method :

```
% Input section
syms x;  % Symbolic variable
f_expr = input('Enter the function (in terms of x): ', 's');
```

```
f = matlabFunction(str2sym(f_expr));

% Calculate the derivative of the function using symbolic differentiation
df = diff(str2sym(f_expr));

x0 = input('Enter the initial guess: ');
e = input('Enter the tolerance level: ');
n = input('Enter the maximum iteration number: ');

% Algorithm (Newton-Raphson method)
if subs(df, x, x0) ~= 0
    for i = 1:n
        % Next step
        df_val = subs(df, x, x0);
        x1 = x0 - (f(x0) / df_val);
        fprintf('Iteration number %d : %0.4f\n', i, x1);

        if abs(x0 - x1) <= e
            break;
        end

        x0 = x1;
    end
else
    display('Can not solve with Newton-Raphson method');
end

fprintf('The root is %.4f\n', x1);
```

6.Generalized Second method code :

```
%clc
clear all ;
f= input('Enter the function ');
x0= input('Initial guess:');
x1= input('Second initial guess:');
e= input('Tolerance=');
n = input('Iteration number=');
fx0 = f(x0);
fx1= f(x1);
    for i= 1:n
    x2= x1- fx1 *(x1 - x0) /(fx1-fx0); % This is the main  function

    x0= x1 ;  % swap the value of x and f(x)
    fx0 = fx1 ;
    x1= x2 ;
     fx1= f(x1);

     if abs(fx1)<= e
    break
    end
    root= x1 ;
    end
fprintf('The root is %.4f \n Iteration Number= %d ', root, i)
```

7.Generelized code for Gauss seidal method:

```
%clc
clear all
a = input('Enter co-efficient matrix: ');
b = input('Enter constant vector: ');
c = input('Enter initial guess vector: ');
n = input('Enter maximum number of iteration: ');
e = input('Tolerance: ');
N = length(b);
x = zeros(N,1);
y = zeros(N,1);
for j=1:n
for i=1:N
 x(i) = (b(i)/a(i,i)) - (a(i,[1:i-1,i+1:N])*c([1:i-1,i+1:N]))/a(i,i);
 c(i) = x(i);
end
 x';
 if abs(y-x)<e
 break
 end
y = x;
end
fprintf('\n Total Iteration= %d \n Roots are p=%d \n q=%d \n r=%d
\n',j,x(1),x(2),x(3));
```

8.Generalized code for Curve fitting :

```
%clc;
clear  all;
x=input('Enter the x co-ordinates :');
y=input('Enter the y co-ordinates :');
N=input('Enter the number of fitting :');
n=length(x);
%A Matrix Calculation%
A=zeros(N,N);
for i=1:N
for j=1:N
A(i,j)=sum(x.^(i+j-2));
end
end
%B Matrix Calculation%
B=zeros(N,1);
for k=1:N
B(k)=sum((x.^(k-1)).*y);
end
```

```
%constant determination%
U=A\B;
%printing equation%
display('The System Equation is :')
for g=N:-1:1
fprintf('(+%.4fx^%d)',U(g),g-1)
end
%plotting portion%
%Data Point%
%polyval(matrix,datapoint)
P=flip(U)
X=linspace(x(1),x(n),100);
Y=polyval(P,X);
plot(X, Y)
hold on
plot(x, y,'*')
```