Contents

ROOTS OF EQUATIONS

THE MATLAB CODE OF THE NEWTON'S METHOD

Q1:

Write a MATLAB implementation that applies the Newton's method to and an approximate solution of the nonlinear system

$$\begin{cases} \frac{1}{5}\ e^{\ 2Ax} \quad B\sin(xy) \ = 0.4325 \\ \frac{1}{5}\left(x^2y + B\cos(x)\right) = 0.0643 \end{cases}$$

by taking the termination criteria e= 10^{20} % and gives the root as output*:*

Solution:

The MATLAB code is as below.

```
function NewtonsMethod

clear all
clc

syms x y

x0 = input('Give the value of x0!: ');
y0 = input('Give the value of y0!: ');

TC = 10^(-20);

A = 1;
B = 1;

f1 = @(x,y)((1/5)*(exp(-2*A*x)-B*sin(x*y))-0.4325);
f2 = @(x,y)((1/5)*((x^2*y)+B*cos(x))-0.0643);

dxf1 = diff(f1, x); %df1/dx
dyf1 = diff(f1, y); %df1/dy
dxf2 = diff(f2, x); %df2/dx
```

```
dyf2 = diff(f2, y); %df2/dy

J = [dxf1,dyf1;dxf2,dyf2]; %J


invJ = matlabFunction(inv(J));%inverse function of J

x(1) = x0;

y(1) = y0;

i = 1;

E(1) = 100;

while (E>TC);

    S = eval([x(i); y(i)] + (feval(invJ,x(i),y(i)) *
[feval(f1,x(i),y(i)) ;  feval(f2,x(i),y(i))]));

    x(i+1) = S(1,1);
    y(i+1) = S(2,1);
    E(i+1) = eval(max(abs(((x(i+1)-x(i))/x(i+1))*100),
abs((((y(i+1)-y(i))/y(i+1)))*100)));

    i = i+1;

end
disp('  ')
disp('After calculations;')
Error = E(i-1)
x = eval(x(i-1))
y = eval(y(i-1))


end
```

THE OUTPUT OF MATLAB CODE NEWTON'S METHOD

```
Give the value of x0!: 2
Give the value of y0!: 3

After calculations;

Error =

  303.0018


x =

  449.8367


y =

   -0.0254
```

NUMERICAL ANALYSIS OF THE NEWTON'S METHOD

For different x0 and y0 values, there are different number of iterations and roots.

The method is very expensive - It needs the function evaluation and then the derivative evaluation.

The advantage of the method is its order of convergence is quadratic.

ALGORITHM FOR NEWTON'S METHOD

1. Create an m-file and name the function.
2. Introduce eqn (function)
3. Declare derivative function.
4. Ask user to enter X0,TC.

5. Declare error
6. In a while loop,
7. -find xsub(i+1)
8. -find error
9. –increment
10. Give the output

THE MATLAB CODE OF THE POLYNOMIAL REGRESSION

Q2:

Write a MATLAB implementation that takes the number of data points n (which is not less than 3) as input; creates arrays for x and y values of the data points and asks user to enter these values; constructs the best polynomial P3 (x) and gives it as the output.

Solution:

```
function Q2

clear all
clc

syms A B C

n = input('How many data points do you have?: ');
k = 1;
for i = 1:n
    fprintf('\n');
    fprintf('    Give data for point %i \n',i);
    fprintf('\n');
    x(k) = input(' Give x value!: ');
    y(k) = input(' Give y value!: ');
    fprintf('\n');


    xy(k) = x(k)*y(k);
    x2(k) = x(k)^2;
    x2y(k) = x2(k)*y(k);
    x3(k) = x(k)^3;
    x4(k) = x(k)^4;
```

```
    k = k+1;

end

sx = sum(x);
sy = sum(y);
sxy = sum(xy);
sx2 = sum(x2);
sx2y = sum(x2y);
sx3 = sum(x3);
sx4 = sum(x4);

eqn1 = A*sx4+B*sx3+C*sx2 == sx2y;
eqn2 = A*sx3+B*sx2+C*sx == sxy;
eqn3 = A*sx2+B*sx+C == sy;

[inp,outp]                                                    =
equationsToMatrix([eqn1,eqn2,eqn3],[A,B,C]);

unks = linsolve(inp,outp);

A = unks(1);
B = unks(2);
C = unks(3);

F = @(x) A*(x^2) + B*(x) + C;

syms x
disp(F(x))
```

THE OUTPUT OF THE MATLAB CODE POLYNOMIAL REGRESSION

```
How many data points do you have?: 3

    Give data for point 1

 Give x value!: 1
 Give y value!: 3


    Give data for point 2

 Give x value!: 4
 Give y value!: 6


    Give data for point 3

 Give x value!: 7
 Give y value!: 9

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND =  7.436822e-20.
> In Q2 (line 39)

F =

    @(x)A*(x^3)+B*(x^2)+C*(x)+D

43/16 - (5*x^2)/4 - (5*x^3)/128 - (21*x)/8
```

ALGORITHM FOR POLYNOMIAL REGRESSION

1. Name the implementation.

2. Ask user to enter how many data point numbers would be entered

3. In a for loop , code ask the x and y values as much as the data points

Also code calculates the;

The sum of x values
The sum of y values
The sum of x^2 values
The sum of x^3 values
The sum of x^4 values
The sum of x^5 values
The sum of x^6 values
*The sum of x*y values*
*The sum of (x^2)*y values*
*The sum of (x^3)*y values*

4. Then setting the augmented matrix.

5. Then setting the resultant matrix.

6. Taking the inverse of the augmented matrix and multiplying it with resultant matrix and finally giving the output.

$$[A; B; C; D] = [\text{augmented matrix}]^{-1} * [\text{resultant matrix}]$$

After these steps we get the constants and we put them into the best polynomial equation.

THE MATLAB CODE OF THE SIMSPSONS 3/8 RULE

Q3:

Write a MATLAB implementation that applies the Simspsons 3/8 rule to and an approximation *I* for

$$I = \int_0 x^2 e^{x_2} dx:$$

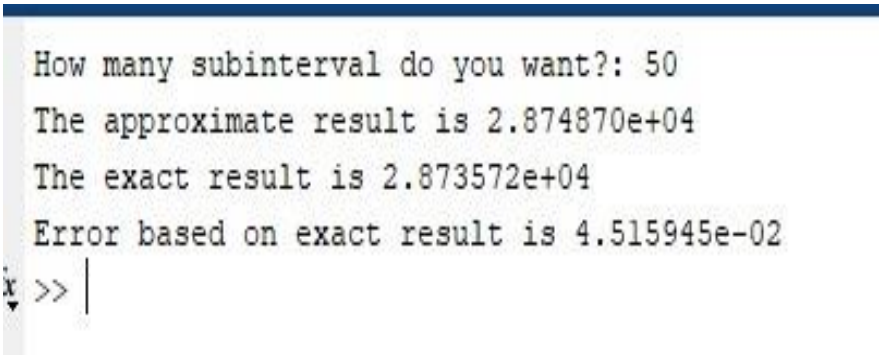
Give the error as the output.

Solution:

```
function Q3

clear all
```

```
clc
syms x;
N = input('How many subinterval do you want?: ');

a = 0;
b = pi;
f = @(x) (x.^2).*exp(x.^2);

h = (b-a)/(N);

T = a:h:b;

I = (3*h/8) * (f(T(1)) - 3*sum(f(T(3:3:end-1))) +
3*sum(f(T(2:1:end-1))) + 2*sum(f(T(3:3:end-1)))+
f(T(end)));

Exact = integral(f,0,pi);

Error = abs((Exact-I)/Exact)*100;

fprintf('The approximate result is %d\nThe exact
result is %d\nError based on exact result is %d\n', I,
Exact, Error)
```

THE OUTPUT OF THE MATLAB CODE SIMSPSONS 3/8 RULE



ALGORITHM FOR METHOD SIMSPSONS 3/8 RULE

Step 1. Input a, b and the number of intervals.
Step 2. Compute the value of h using this formula: h = (b - a) / n
Step 3. Calculate the value of sum using this formula: sum = f(a) + f(b)
Step 4. If n is an odd number, then
sum = sum + 2 * y(a + i * h)
Else
sum = sum + 3 * y(a + i * h)
Step 5. Calculate the Simpson's integration value using this formula: sum* 3 * h / 8

THE MATLAB CODE OF THE HIGH ACCURACY DIFFERENTIATION

Q4:

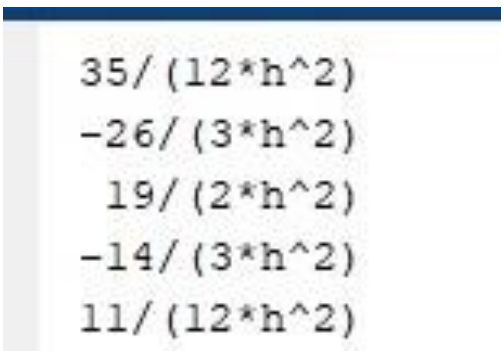
Obtain a fourth order of accuracy approximation formula for y''(0): You can make use of MATLAB software to and the unknown coefficients.

Solution:

```
function Q4
clear
clc
syms h
```

```
Augmented = (zeros(5,5,'sym'));
for k = 1:5

Augmented(1,k) = 1;

...
end

for i = 2:5

for j = 2:5

Augmented (i,j)=((j-1)^(i-1))./(factorial (i-
1)).*(h.^(i-1));

...

end

end

Resultant = (zeros(5,1));
Resultant(3,1) = 1;

Coefficients = inv(Augmented)*Resultant;

disp(Coefficients);

end
```

THE OUTPUT OF MATLAB CODE HIGH ACCURACY DIFFERENTIATION



NUMERICAL ANALYSIS OF THE HIGH ACCURACY DIFFERENTIATION

In this numerical differentiation part we have f'' (0); so we need to use forward mesh points. After this decision, we set the equation and expand the four term by using Taylor series expansion. Then in the light of the the accuracy formula we get five equations and five unknowns. To find unknowns out we use MATLAB code and collect the constants. Then finally we obtain third order of accuracy approximation formula for $y^{II}(0)$ with putting the known constants in the main equation.

ALGORITHM FOR HIGH –ACCURACY DIFFERENTIATION

1. Name the implementation.
2. Setting the *(5x5)* zeros matrix *(augmented matrix)* for filling h values.
3. Using for loop filled the first row of augmented matrix with ''1''.
4. Then again using for loop in a for loop we set the equation for filling other rows of the augmented matrix.
5. Creating the coefficients matrix *(5x1)* with zeros command.
6. To find this unknowns; take the inverse of the augmented matrix then multiply with the resultant matrix.
7. *[Coefficients]= [Augmented^{-1}]*[Resultant]*
8. Give the output matrix.

THE MATLAB CODE OF THE ORDINARY DIFFERENTIAL EQUATIONS

Q5:

Consider the second order differential equation with Neumann boundary conditions

$$y^{00}(t) + y^{0}(t) + e^{Dt}y(t) = e^{Dt} + 2e^{2t} + te^{Dt} + 1;$$
$$0 < t < T; y_0(0) = 1; y_0(T) = 2e^{2T} + 1; \quad (P)$$

where $D = A + B$. The exact solution of this problem is $y(t) = e^{2t} + t$:

a) Construct a second order of accuracy difference scheme for the numerical solution of problem (P).

b) Give the matrix realization.

c) Write a MATLAB implementation that finds the numerical solution of problem (P) by the scheme constructed in part (a): The implementation should take the number of subintervals N as an input. It should and the exact values at the mesh points and compare it with the approximate values derived by the difference scheme. For the error use the formula $E = \max_{0 \le k \le N} |y_k - y(t_k)|$:

Make the following table:

	$N = 10$	$N = 20$	$N = 40$	$N = 80$
Error ($T = 1$)				
Error ($T = 2$)				
Error ($T = 3$)				

In the output you should check if the error decrease by the factor 1/4 approximately when N is doubled.

Solution:

```
function Q5

clear
clc

N =input('Enter the number of subinterval "N" : ');

T = input( 'Enter the value of "T" : ' );
h= T/N;

Augmented = zeros(N+1,N+1);
Augmented(1,1:3) = [-3,4,-1];
Augmented(N+1,N+1) = 1;

Resultant = zeros(N+1,1);
Resultant(1,1) = -2*h;
Resultant(N+1,1) = exp(-2*T)+T;


for  i = 2:N


Augmented(i,i-1) = ((1/(h^2))-(1/(2*h)));
Augmented(i,1) = (-2/(h^2))+(exp(6*i*h));
Augmented(i,i+1) = (1/(h^2))+(1/(2*h));
Augmented(i,1)              =             (((exp(6*i*h)+2)*exp(-
2*i*h)))+(i*h*exp(6*i*h))+1;

end

Solution = zeros(N+1,1);

for  i = 0:N

Solution(i+1,1) = exp(-2*i*h)+(i*h);

end

Coefficient = inv(Augmented)*Resultant;

error = max(abs(Solution-Coefficient));

fprintf('Maximum Error : ');
disp(error);

end
```

THE OUTPUT OF MATLAB CODE ORDINARY DIFFERENTIAL EQUATIONS

```
Enter the number of subinterval "N" : 30
Enter the value of "T" : 1
Maximum Error :     4.6526
```

Error (T = 1)	0.0982	0.1112	0.0673	0.0408
Error (T = 2)	0.2244	0.1296	0.2681	0.0354
Error (T = 3)	0.2980	0.1481	0.0765	0.0519

NUMERICAL ANALYSIS OF THE ORDINARY DIFFERENTIAL EQUATIONS

If the error decrease by the factor 1/4 approximately when N is doubled logically since it is second-order differential equation.

ALGORITHM FOR ORDINARY DIFFERENTIAL EQUATION WITH NEUMANN BOUNDARY CONDITIONS

Algorithm of the Neumann Boundary Conditions as follows:

- Name the implementation
- Ask user to enter N and T values
- Creating Augmented matrix and Resultant matrix using Neumann Boundary Conditions.

- In a 'for' loop, compute and add the coefficients for Augmented matrix and Resultant matrix.
- Find the exact solution.
- Using inverse of matrix rule obtain Coefficient matrix.
- Find error using formula which is given in the question.
- Give the error as output.

CONCLUSION

In this project, how the numerical methods make the life easy for engineers, mathematician, physicist etc. Thanks to this project and our lecturer Mr. Sazaklıoğlu we learned and practice the numerical abilities and methods at MATLAB. We will use the skills gained from this project at the other numerical courses and throughout our engineering life.

REFERENCES

[1] Numerical-Methods-for-Engineers-6th-Edition

[2] Courses Notes