VSIT Vidyalankar School of Information Technology
NAAC Accredited College

LAB MANUAL

SUBJECT:_NUMERICAL AND STATISTICAL METHODS

YEAR : FIRST YEAR SEMESTER :_II

ACADEMIC YEAR : 2018-2019

Practical : 1

1. Title:

a) Program to solve algebraic and transcendental equation by bisection method.

b) Program to solve algebraic and transcendental equation by false position method.

2. Prior Concepts:

1. Algorithm of Bisection method
2. Algorithm of Regula-Falsi method

3. New Concepts:

1. Functions and syntax used for the program
2. deff('y=f(x)', ['y=x^3-4*x-9']) : to define a function
3. printf("format string", list of variables): to format the output
4. loop structure: while(condition)

```
                    .
                    .
                   end
```

4. Objectives:

1. Use programming techniques to solve algebraic and transcendental equations.
2. To display the answer in a tabulated form to get desired accuracy of the root.

5. Procedure:

ALGORITHM (Bisection Method):

1. Consider $f(x) = 0$ and find values of a and b such that $f(a)f(b) < 0$
2. Set$x_0 = \frac{a+b}{2}$
3. Find $f(x_0)$
4. If $f(a)f(x_0) < 0$ then root lies in (a, x_0) set $b = x_0$
5. If $f(a)f(x_0) > 0$ then root lies in (x_0, b) set $a = x_0$
6. Continue the process till the required accuracy.

ALGORITHM (False-Position Method)

1. Consider $f(x) = 0$ and find values of a and b such that $f(a)f(b) < 0$
2. Set $x_0 = \frac{af(b)-bf(a)}{f(b)-f(a)}$

3. Find $f(x_0)$
4. If $f(a)f(x_0) < 0$ then root lies in (a, x_0) set $b = x_0$
5. If $f(a)f(x_0) > 0$ then root lies in (x_0, b) set $a = x_0$
6. Continue the process till the required accuracy.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Algebraic and Transcendental Equations.

9. Questions:

(a) Scilab Program to solve algebraic and transcendental equation by bisection method.

1. $f(x) = x^3 - 4x - 9, a = 3, b = 4$
2. $f(x) = x^3 - x - 1, a = 1, b = 2$

PROGRAM:

```
function bisection(x1, x2, f)
 printf("Bisection Program");
 if (f(x1)*f(x2)>0) then
    error('no root possible ')
    abort;
 end;
 if(f(x2)< 0) then
   t=x1;
   x1=x2;
   x2=t;
 end
```

```
  i=0;
  printf("\n\tx1\tx2\tx3\tf(x1)\t\tf(x2)\tf(x3)");
  while(i<10)
   x3=(x1+x2)/2;
   printf("\n\t %.3f \t %.3f \t %.3f \t %.3f \t %.3f \t %.3f",x1,x2,x3,f(x1),f(x2),f(x3));


   if (f(x3)<0) then
      x1=x3;
   else
      x2=x3;
   end
   i=i+1;
  end
endfunction;
```

<u>command</u>

➔ deff('y=f(x)', ['y=x^3-4*x-9'])

➔ bisection(2,3,f)

<u>OUTPUT:</u>

<u>Bisection Program</u>

```
x1      x2      x3      f(x1)   f(x2)   f(x3)
2.000   3.000   2.500   -9.000  6.000   -3.375
2.500   3.000   2.750   -3.375  6.000   0.797
2.500   2.750   2.625   -3.375  0.797   -1.412
2.625   2.750   2.688   -1.412  0.797   -0.339
2.688   2.750   2.719   -0.339  0.797   0.221
2.688   2.719   2.703   -0.339  0.221   -0.061
2.703   2.719   2.711   -0.061  0.221   0.079
2.703   2.711   2.707   -0.061  0.079   0.009
```

```
2.703   2.707   2.705   -0.061  0.009   -0.026

2.705   2.707   2.706   -0.026  0.009   -0.009
```

The approximate root of the equation is 2.706

(b) Scilab Program to solve algebraic and transcendental equation by false position method.

1. $f(x) = x^3 - 4x - 9, a = 3, b = 4$
2. $f(x) = x^3 - x - 1, a = 1, b = 2$

PROGRAM:

```
function regulafalsi(x1, x2, f)

  printf("Regula Falsi Program");

  if (f(x1)*f(x2)>0) then
      disp('no root possible ')
      abort;
  end;

  if(f(x2)< 0) then
     t=x1;
     x1=x2;
     x2=t
  end

  i=0;

 printf("\n\tx1\tx2\tx3\tf(x1)\t\tf(x2)\tf(x3)");

  while(i<10)
    x3=(x1*f(x2) - x2*f(x1)) / (f(x2)-f(x1));

printf("\n\t %.3f \t %.3f \t %.3f \t %.3f \t %.3f \t %.3f",x1,x2,x3,f(x1),f(x2),f(x3));

   if (f(x3)<0) then
      x1=x3;
   else
      x2=x3;

end;
      i=i+1;
  end

endfunction;
```

command

```
➔ deff('y=f(x)', 'y=x^3-x-1')

➔ regulafalsi (1,2,f)
```

OUTPUT:

```
Regula-Falsi Program

      x1     x2     x3     f(x1)  f(x2)  f(x3)
      1.000  2.000  1.167  -1.000 5.000  -0.579
      1.167  2.000  1.253  -0.579 5.000  -0.285
      1.253  2.000  1.293  -0.285 5.000  -0.130
      1.293  2.000  1.311  -0.130 5.000  -0.057
      1.311  2.000  1.319  -0.057 5.000  -0.024
      1.319  2.000  1.322  -0.024 5.000  -0.010
      1.322  2.000  1.324  -0.010 5.000  -0.004
      1.324  2.000  1.324  -0.004 5.000  -0.002
      1.324  2.000  1.325  -0.002 5.000  -0.001
      1.325  2.000  1.325  -0.001 5.000  -0.000
 The approximate root of the equation is 1.325
```

Practical : 2

1. **Title:**
   a) Program to solve algebraic and transcendental equation by Newton Raphson method.
   b) Program to solve algebraic and transcendental equation by Secant method.

2. **Prior Concepts:**
   1. Algorithm of Newton Raphson method
   2. Algorithm of Secant method

3. **New Concepts:**
1. Functions and syntax used for the program
2. deff('y=f(x)', ['y=---------']) : to define a function
3. printf("format string", list of variables): to format the output
4. loop structure: while(condition)

```
                .
                .
              end
```

5. **Objectives:**
1. Use programming techniques to solve algebraic and transcendental equations.
2. To display the answer in a tabulated form to get desired accuracy of the root.

6. **Procedure:**

<u>ALGORITHM (Newton Raphson Method)</u>

1. Consider $f(x) = 0$ and find$f'(x)$
2. Let initial approximation bex_0
3. Find $f(x_0)$ and $f'(x_0)$
4. The next approximation is$x_1 = x_0 - \frac{f(x_0)}{f\prime(x_0)}$
5. Find $f(x_1)$ and $f'(x_1)$ to get x_2.
6. Continue the process till the required accuracy.

<u>FORMULA (Secant Method)</u>

$$\boldsymbol{x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}}$$

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Algebraic and Transcendental Equations.

9. Questions:

(a) Scilab Program to solve algebraic and transcendental equation by Newton Raphson Method

1. $f(x) = xe^x - 1,\ x_0 = 1$
2. $f(x) = x^3 - x - 1,\ x_0 = 1$

PROGRAM:

```
function newtonraphson(xn, f, fd)
disp("Newton Raphson Method");
printf("\n\ti\txi\tf(xi)\tfd(xi)\tx(i+1)");
i=0;

while(i<10)
 xnp=xn-(f(xn)/fd(xn));
  i=i+1;
printf("\n\t%d\t%.3f\t%.3f\t%.3f\t%.3f",i,xn,f(xn),fd(xn),xnp);
  xn=xnp;
end
printf("\n\nThe approximate root of the equation is:%.3f",xnp);
endfunction

deff('y=f(x)','y=x*exp(x)-1')
deff('y=fd(x)','y=x*exp(x)+exp(x)')
newtonraphson(1,f,fd)
```

OUTPUT:

```
Newton Raphson Method

        i       xi      f(xi)   fd(xi)  x(i+1)
        1       1.000   1.718   5.437   0.684
        2       0.684   0.355   3.337   0.577
        3       0.577   0.029   2.810   0.567
        4       0.567   0.000   2.764   0.567
        5       0.567   0.000   2.763   0.567
```

```
6       0.567   0.000   2.763   0.567
7       0.567   0.000   2.763   0.567
8       0.567   0.000   2.763   0.567
9       0.567   0.000   2.763   0.567
10      0.567   0.000   2.763   0.567
```

The approximate root of the equation is: 0.567

(b) Scilab Program to solve algebraic and transcendental equation by Secant Method

(a) $f(x) = x^3 - 2x - 5,\ x_n = 1, x_m = 2$

PROGRAM:

```
function secant(xn, xm, f)
disp("Secant Method");
printf("\n\ti\txi-1\t\txi\t\tf(xi-1)\t\tf(xi)\t\tx(i+1)");
i=0;
while(i<5)
xnm=xm-((f(xm)*(xn-xm))/(f(xn)-f(xm)));
printf("\n\t%d\t%f\t%f\t%f\t%f\t%f",i,xn,xm,f(xn),f(xm),xnm);
xn=xm;
xm=xnm;
i=i+1;
end
printf("\n\nThe approximate root of the equation is:%f",xnm);
endfunction
deff('y=f(x)','y=x^3-2*x-5')
secant(1,2,f)
```

OUTPUT:

Secant Method

```
i       xi-1        xi          f(xi-1)     f(xi)       x(i+1)
0       1.000000    2.000000    -6.000000   -1.000000   2.200000
1       2.000000    2.200000    -1.000000   1.248000    2.088968
2       2.200000    2.088968    1.248000    -0.062124   2.094233
3       2.088968    2.094233    -0.062124   -0.003555   2.094552
4       2.094233    2.094552    -0.003555   0.000011    2.094551

The approximate root of the equation is: 2.094551
```

Practical: 3

1. **Title:**

a) Program for Newton's forward interpolation.

b) Program for Newton's backward interpolation.

c) Program for Lagrange's interpolation.

2. Prior Concepts:

1. Formula for Newton's Forward Interpolation

   For a given set of values $(x_i, y_i),\ i = 0,1,2, \dots n$ where x_i's are at equal intervals and h is the interval of differencing i.e. $x_i = x_0 + ih$ then

   $$y(x) = y_0 + P\Delta y_0 + \frac{P(P-1)}{2!}\Delta^2 y_0 + \cdots + \frac{P(P-1)(P-2)\dots(p-n+1)}{n!}\Delta^n y_0$$

   Where,

   $P = \frac{x-x_0}{h}$

   x → missing term

   x_0 → first value in the table

   h → interval of differencing

2. Formula for Newton's Backward Interpolation

   For a given set of values $(x_i, y_i),\ i = 0,1,2, \dots n$ where x_i's are at equal intervals and h is the interval of differencing i.e. $x_i = x_0 + ih$ then

   $$y(x) = y_n + P\nabla y_n + \frac{P(P+1)}{2!}\nabla^2 y_n + \cdots. \qquad + \frac{P(P+1)(P+2)\dots(p+n-1)}{n!}\nabla^n y_n$$

   Where,

   $P = \frac{x-x_n}{h}$

   x → missing term

   x_n → last value in the table

   h → interval of differencing

3. Formula for Lagrange's Interpolation

   Let $y = f(x)$ be a function which takes the values $f(x_0), f(x_1), \dots, f(x_n)$ corresponds to $x_0, x_1, x_2, \dots x_n$ where values of x are not necessarily at equal intervals. Then Lagrange's interpolation formula is

   $$f(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} \times f(x_0) \qquad + \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} \times f(x_1) + \cdots$$

   $$+ \frac{(x-x_0)(x-x_1)(x-x_2)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\dots(x_n-x_{n-1})} \times f(x_n)$$

3. New Concepts:

1. Problems based on Newton's Forward Interpolation

2. Problems based on Newton’s backward interpolation
3. Problems based on Lagrange’s interpolation.
4. Loop to prepare the forward and backward difference table

4. Objectives:

1. Students should be able to solve all sums based on interpolation.

5. Procedure:

1. Define a function.
2. Use of for loop to prepare table
3. Use of for loop to implement formula.
4. Calculation of final answer.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Students should be able to solve all sums based on interpolation.

9. Questions:

Newton’s forward interpolation

1. Write a scilab program to find f(8) from the following data.

x	5	10	15	20
y	50	70	100	145

PROGRAM:

```
function newtonsforward(x, y, xg)
 n=length(x);
    if(length(x)<>length(y))   then
      disp("Not Possible");
      abort;
  end
h=x(2)-x(1);
p=(xg-x(1))/h;
```

```
  for i=1:n-1
      d(i,1)=y(i+1)-y(i);
  end

  for j=2:n-1
      for i=1:n-j
          d(i,j)=d(i+1,j-1)-d(i,j-1);
      end
  end
  disp(x,"The Values of x");
  disp(y,"The Values of y");
  disp(d,"Forward Difference Table");

term(1)=p;
  for i=2:n-1
      term(i)=term(i-1)*(p+1-i);
  end

t=0;
  for i=1:n-1
      t=t+((term(i)*d(1,i))/factorial(i));
  end
ye=y(1)+t;
  disp(ye,"The estimated value of y is");
endfunction
x=[5,10,15,20];
y=[50,70,100,145];
xg=8;
newtonsforward(x,y,xg)
```

OUTPUT:

```
The Values of x
     5.    10.    15.    20.
  The Values of y
     50.    70.    100.    145.
  Forward Difference Table


    20.    10.    5.
    30.    15.    0.
    45.    0.     0.
```

```
 The estimated value of y is

    61.08
```

Newton's backward interpolation

1. Using Newton's backward interpolation formula find y when $x = 3.6$.

x	0	1	2	3	4
y	7	17	45	103	203

PROGRAM:

```
function newtonsbackward(x, y, xg)
 n=length(x);
    if(length(x)<>length(y))   then
      disp("Not Possible");
      abort;
    end
h=x(2)-x(1);
p=(xg-x(n))/h;

  for i=1:n-1
      d(i,1)=y(i+1)-y(i);
  end

for j=2:n-1
      for i=1:n-j
          d(i,j)=d(i+1,j-1)-d(i,j-1);
      end
  end
  disp(x,"The Values of x");
  disp(y,"The Values of y");
  disp(d,"Backward Difference Table");

term(1)=p;
  for i=2:n-1
      term(i)=term(i-1)*(p+(i-1));
  end
```

```
t=0;
  for i=1:n-1
      t=t+((term(i)*d(n-i,i))/factorial(i));
  end
ye=y(n)+t;
  disp(ye,"The estimated value of y is");
endfunction
x=[0,1,2,3,4];
y=[7,17,45,103,203];
xg=3.6;
newtonsbackward(x, y,xg)
```

OUTPUT:

```
The Values of x
     0.    1.    2.    3.    4.
  The Values of y
     7.    17.    45.    103.    203.
  Backward Difference Table
     10.     18.    12.    0.
    28.     30.    12.    0.
    58.     42.    0.     0.
    100.    0.     0.     0.

 The estimated value of y is
     157.192
```

Lagrange's Interpolation

1. Using Lagrange's interpolation formula find $f(3)$ for the following data.

x	1	2	5	6
$f(x)$	10	15	22	35

PROGRAM:

```
function lagrange(x, y, xg)
    n=length(x);
    ye=0;
   for i=1:n
        num=1;
        den=1;
        for j=1:n
```

```
            if (i<>j)   then
                num=num*(xg-x(j));
                den=den*(x(i)-x(j));
            end
        end
        ye=ye+((num/den)*y(i));
    end
    disp(ye,"The estimated value of y is");
endfunction

x=[1,2,5,6];
y=[10,15,22,35];


lagrange(x,y,3)
```

OUTPUT:

```
The estimated value of y is


    16.
```

Practical : 4

1. Title:

Solving linear system of equations by iterative methods

a) Program for solving linear system of equations using Gauss Jordan method.

b) Program for solving linear system of equations using Gauss Seidel method.

2. Prior Concepts:

1. Equations based on Gauss Jorden Method.
2. Equations based on Gauss Seidel Iteration Method.

3. New Concepts:

1. Problems based on Gauss Jorden Method.
2. Problems based on Gauss Seidel Method.

4. Objectives:

1. Students should be able to solve all sums based on linear system of equations.

5. Procedure:

1. Define a function.
2. Use in build functions to find determinant and augmented matrix.
3. Find reduced form of matrix

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Linear Equations.

9. Questions:

Gauss Jordan Method:

<u>**PROGRAM:**</u>

1. Write a scilab program to solve the given system of equations by Gauss-Jordan method.

$$3x + 5y - 4z = 22$$

$$2x - 3y + z = 3$$

$$-x + 4y + 6z = 19$$

Program:

```
function gaussjordan(A, B)
   d=det(A);
   if d==0 then
      disp("Matrix is non-singular. Solution does not exist");
      abort;
   end
   A_aug=[A B];
   disp(A_aug,"The augmented matrix [A:B]=");
   AB_ech=rref(A_aug);
   disp(AB_ech,"Reduced form of [A:B]=");
   disp("The required solution is:")
   printf("\nx=%g",AB_ech(1,4));
   printf("\ny=%g",AB_ech(2,4));
   printf("\nz=%g",AB_ech(3,4));
endfunction
A=[3,5,-4;2,-3,1;-1,4,6];
B=[22;3;19];
gaussjordan(A,B);
```

OUTPUT:

```
gaussjordan(A,B);

 The augmented matrix [A:B]=


   3.    5.  - 4.    22.
   2.  - 3.    1.    3.
 - 1.    4.    6.    19.

 Reduced form of [A:B]=


   1.    0.    0.    5.
   0.    1.    0.    3.
   0.    0.    1.    2.

 The required solution is:
x=5
y=3
z=2
```

Gauss-Siedal method:

PROGRAM:

1. Write a scilab program to solve given system of equations using Gauss-Siedal method.

$$2x - y + z = 5$$
$$x + 3y - 2z = 7$$
$$x + 2y + 3z = 10$$

Program:

```
x(1)=0;
y(1)=0;
z(1)=0;

for i=2:10
   x(i)=(5+y(i-1)-z(i-1))/2;
   y(i)=(7-x(i)+2*z(i-1))/3;
   z(i)=(10-x(i)-2*y(i))/3;
end

printf("\n The approximate value of x is %d", round(x(10)));
printf("\n The approximate value of y is %d", round(y(10)));
printf("\n The approximate value of z is %d", round(z(10)));
```

OUTPUT:

```
The approximate value of x is 3
 The approximate value of y is 2
 The approximate value of z is 1
```

Practical : 5

1.Title:

Program of Numerical Integration.

2. Prior Concepts:

1. Formula of Trapezoidal rule
2. Formula of Simpson’s 1/3rd
3. Formula of Simpson’s 3/8
4. Y= f(x), how to calculate x and y
5. Values of $y_{0,}$ $y_{1,}$ y_2, y_3, y_4, y_5 ……

3. New Concepts:

1. Problems based on Trapezoidal rule
2. Problems based on Simpson’s 1/3rd
3. Problems based on Simpson’s 3/8.

4. Objectives:

1. Students should be able to solve sums based on integration.

5. Procedure:

1. Define a function.
2. Use of for loop
3. Define the formula:
4. Calling a function by passing parameters.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on differentiation and Integration.

9. Questions:

1. Write a scilab program to evaluate $\int_2^8 x^3 dx$ by using Trapezoidal Rule.

Program:

```
function trapezoidal(a, b, n, f)
h=(b-a)/n;
```

```
for i=a:b
x=(a:h:b);
y=f(x);
end
disp(x,"The values of x are:");
disp(y,"The values of y are:");
m=length(y);
s=y(1)+y(m);
for i=2:m-1

s=s+2*(y(i));
end
s=(h/2)*s;
disp(s);
endfunction;
deff("y=f(x)","y=x.^3")
trapezoidal(2,8,6,f)
```

O/P –
1035

2. Write a scilab program to evaluate $\int_2^8 x^3 dx$ by using Simpson's 1/3rd Rule.

Program:

```
function simpson1by3(a, b, n, f)
h=(b-a)/n;
x=(a:h:b);
y=f(x);
printf("display x");
disp(x);
printf("display y");
disp(y);
m=length(y);
s=y(1)+y(m);
for i=2:m-1
if(modulo(i,2)==0)
s=s+4*(y(i));
else
s=s+2*(y(i));
end
end;
s=(h/3)*s;
disp(s);
endfunction;
deff("y=f(x)","y=x.^3")
simpson1by3(2,8,6,f)
```

O/p-
1020

3. Write a scilab program to evaluate $\int_2^8 x^3 dx$ by using Simpson's 3/8th Rule.

Program:

```
function simpson3by8(a,b,n,f)
h=(b-a)/n;
x=(a:h:b);
y=f(x);
printf("display x");
disp(x);
printf("display y");
disp(y);
p=(3*h)/8;
m=length(y);
s=y(1)+y(m);
for i=2:m-1
if(modulo(i-1,3)==0)
s=s+2*(y(i));
else
s=s+3*(y(i));
end
end;
s=p*s;
disp(s);
endfunction;
deff('y=f(x)','y=x.^3')
simpson3by8(2,8,6,f)
```

O/p –

```
1020
```

Practical : 6

1.Title:

Programs on differential Equations.

2. Prior Concepts:

1. Formula of Euler's method
2. Formula of Euler's modified method
3. Formula of Runge –Kutta 2nd order.
4. Formula of Runge-Kutta 4th order.
5. Values of k_1, k_2, k_3, k_4.

3. New Concepts:

1. Problems based on Euler's method
2. Problems based on Euler's modified method
3. Problems based on Runge-Kutta 2nd order.
4. Problems based on Runge-Kutta 4th order

4. Objectives:

1.Students should be able to solve all sums based on differential equations.

5. Procedure:

1. Define a function.
2. Use of for loop to calculate k1
3. Use of for loop to calculate k2 k3,k4.
4. Calling a function by passing parameters.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on differential equations.

9. Questions:

1. **Write a scilab program to find y(0.1) for the differential equation $\frac{dy}{dx} = x^2 + y \; given \; y(0) = 1$ and $h = 0.02$ using Euler's method.**

Program:

```
     function euler(x0, y0, h, x, f)
      n=(x-x0)/h;
    xc=x0;
    yc=y0;
    printf("\nx\t\ty");
    for i=1:n
      yc=yc+f(xc,yc)*h;
      xc=xc+h;
     printf("\n%.5f\t%.5f",xc,yc);
    end
endfunction
deff('y=f(x,y)','y=x.^2+y')
euler(0,1,0.02,0.1,f)
```

OUTPUT:

```
X                    y
0.02000              1.02000
0.04000              1.04041
0.06000              1.06125
0.08000              1.08255
0.10000              1.10432
```

Euler's modified method:

2. **Write a scilab program to find y(0.1) for the differential equation $\frac{dy}{dx} = x^2 + y\ given\ y(0) = 1$ and $h = 0.02$ using Euler's method.**

Program:

```
function eulermod(x0, y0, h, x, f)
    n=(x-x0)/h;
     printf("\nx\ty")
    for i=1:n
       x1=x0+h;
       y1=y0+h*f(x0,y0);
     while(n>1)
        yn=y0+((h/2)*(f(x0,y0)+f(x1,y1)));
         if(abs(yn-y1)<0.0001) then
             break;
         end
         y1=yn;
       end
    printf("\n%.5f\t%.5f",x1,y1);
       x0=x1;
       y0=y1;
    end
endfunction
deff('y=f(x,y)', 'y=x.^2+y')
eulermod(0,1,0.02,0.1,f)
```

OUTPUT:

```
X                 y
0.02000           1.02000
0.04000           1.04083
0.06000           1.06191
0.08000           1.08346
0.10000           1.10551
```

Runge-Kutta 2nd order method

3. **Write a scilab program to find y(0.1) for the differential equation $\frac{dy}{dx} = x^2 + y \; given \; y(0) = 1$ and $h = 0.05$ using Runge-kutta 2nd order method.**

Program:

```
function rk2(x0, y0, h, x, f)
n=(x-x0)/h;
xc=x0;
yc=y0;
printf("\nx\ty")
for i=1:n
k1=h*f(xc,yc);
k2=h*f(xc+h,yc+k1)
yc=yc+(k1+k2)/2;
xc=xc+h;
printf("\n%.5f\t%.5f",xc,yc);
end
endfunction
deff('y=f(x,y)','y=x.^2+y')
rk2(0,1,0.05,0.1,f)
```

OUTPUT:

```
X                 y
0.05000           1.05131
0.01000           1.10551
```

---

4. **Write a scilab program to find y(0.6) for the differential equation $\frac{dy}{dx} = 1 + y^2 given \; y(0) = 0$ and $h = 0.2$ using Runge-kutta 4th order method.**

Program:

```
function rk4(x0, y0, h, x, f)
   n=(x-x0)/h;
```

```
    xc=x0;
    yc=y0;
    printf("\nx\ty");
        for i=1:n
        k1=h*f(xc,yc);
        k2=h*f(xc+h/2,yc+k1/2);
        k3=h*f(xc+h/2,yc+k2/2);
        k4=h*f(xc+h,yc+k3);
        yc=yc+(k1+2*k2+2*k3+k4)/6;
        xc=xc+h;
            printf("\n%.5f\t%.5f",xc,yc);
        end
endfunction
deff('y=f(x,y)','y=1+y.^2')
rk4(0,0,0.2,0.6,f)
```

O/p-

```
X                   y
0.20000             0.20271
0.40000             0.42279
0.6000              0.68413
```

Practical : 7

1.Title:

Programs on Regression.

2. Prior Concepts:

1. Equations based on Linear regression.
2. Equations based on polynomial regression.

3. New Concepts:

1. Problems based on linear regression using a straight line.
2. Problems based on polynomial regression using a second degree curve.

4. Objectives:

1. Students should be able to solve all sums based on regressions.

5. Procedure:

1.Define a function.

2.Use for loop to calculate and print the x and y values.

3. Calling a function by passing parameters in it.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Regression.

9. Questions:

1. **Write a program to find regression equation of y on x for the given data.**

X	5	6	8	10	21	34	45	23	12
Y	54	56	63	67	64	78	89	100	76

Program:

```
function regressionlinear(x, y)
n=length(x);
m=length(y);
sx=0;
```

```
sy=0;
sx2=0;
sxy=0;
for i=1:n
    sx=sx+x(i);
    sy=sy+y(i);
    sx2=sx2+x(i)^2;
    sxy=sxy+x(i)*y(i);
end
A=[n,sx;sx,sx2];
B=[sy;sxy];
Y=inv(A)*B;
printf("\nThe regression equation of y on x is y=%.4f+%.4fx",Y(1,1),Y(2,1));
endfunction

x=[5,6,8,10,21,34,45,23,12];
y=[54,56,63,67,64,78,89,100,76];
regressionlinear(x,y)
```

OUTPUT:

```
The regression equation of y on x is y =57.5731+ 0.7856 x.
```

2. **Write a program to fit second degree curve for the given data.**

X	5	6	8	10	21	34	45	23	12
Y	54	56	63	67	64	78	89	100	76

Program:

```
function regression2(x, y)
n=length(x);
m=length(y);
sx=0;
sy=0;
sx2=0;
sx3=0;
sx4=0;
sxy=0;
sx2y=0;

for i=1:n
    sx=sx+x(i);
    sy=sy+y(i);
    sx2=sx2+x(i)^2;
    sx3=sx3+x(i)^3;
    sx4=sx4+x(i)^4;
    sxy=sxy+x(i)*y(i);
    sx2y=sx2y+x(i)^2*y(i);
end
A=[n,sx,sx2;sx,sx2,sx3;sx2,sx3,sx4];
B=[sy;sxy;sx2y];
Y=inv(A)*B;
```

```
printf("\nThe regression equation of y on x is y=%.4f+%.4fx+%.4fx^2",Y(1,1),Y(2,1),Y(3,1));
endfunction

x=[5,6,8,10,21,34,45,23,12];
y=[54,56,63,67,64,78,89,100,76];
regression2(x,y)
```

OUTPUT:

```
The regression equation of y on x is y = 46.4773 + 2.2274 x + -0.0302x^2.
```

Practical :8

1.Title:

Programs on Regression.

2. Prior Concepts:

1. Equations based on polynomial regression.
2. Equations based on multiple linear regression

3. New Concepts:

1. Problems based on polynomial regression using a third degree curve.
2. Problems based on multiple linear regressions.

4. Objectives:

1. Students should be able to solve all sums based on differential equations.

5. Procedure:

1. Define a function.
2. Use for loop to calculate and print the x and y values.
3. Calling a function by passing parameters in it.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Regression.

9. Questions:

1. **Write a program to fit third degree polynomial for the given data.**

X	5	6	8	10	21	34	45	23	12
Y	54	56	63	67	64	78	89	100	76

Program:

```
function regression3(x, y)
n=length(x);
m=length(y);
sx=0;
```

```
sx2=0;
sx3=0;
sx4=0;
sx5=0;
sx6=0;
sy=0;
sxy=0;
sx2y=0;
sx3y=0;
for i=1:n
   sx=sx+x(i);
   sy=sy+y(i);
   sx2=sx2+x(i)^2;
   sx3=sx3+x(i)^3;
   sx4=sx4+x(i)^4;
   sx5=sx5+x(i)^5;
   sx6=sx6+x(i)^6;
   sxy=sxy+x(i)*y(i);
   sx2y=sx2y+x(i)^2*y(i);
   sx3y=sx3y+x(i)^3*y(i);
end
A=[n,sx,sx2,sx3;sx,sx2,sx3,sx4;sx2,sx3,sx4,sx5;sx3,sx4,sx5,sx6];
B=[sy;sxy;sx2y;sx3y];
Y=inv(A)*B;
printf("\nThe required polynomial is y=%.4f+%.4fx+%.4fx^2+%.4fx^3",Y(1,1),Y(2,1),Y(3,1),Y(4,1));
endfunction

x=[5,6,8,10,21,34,45,23,12];
y=[54,56,63,67,64,78,89,100,76];
regression3(x,y)
```

OUTPUT:

```
The required polynomial is y = 29.0248 + 5.7726 x + - 0.2057 x ^ 2 + 0.0024 x ^ 3
```

---

2. **Write a program to fit regression equation of the type $y = a_0 + a_1x_1 + a_2x_2$**

x1	18	22	15	10	30	32	35	42
x2	2	3	1	1	3	4	4	5
Y	4	5	3	2	6	7	8	9

Program:

```
function regressionmulti(x1, x2, y)
n=length(x1);
sx1=0;
sx12=0;
sx2=0;
sx22=0;
```

```
sx1x2=0;
sy=0;
sx1y=0;
sx2y=0;

for i=1:n
   sx1=sx1+x1(i);
   sx12=sx12+x1(i)^2;
   sx2=sx2+x2(i);
   sx22=sx22+x2(i)^2;
   sx1x2=sx1x2+x1(i)*x2(i);
   sy=sy+y(i);
   sx1y=sx1y+x1(i)*y(i);
   sx2y=sx2y+x2(i)*y(i);
end
A=[n,sx1,sx2;sx1,sx12,sx1x2;sx2,sx1x2,sx22];
B=[sy;sx1y;sx2y];
Y=inv(A)*B;
printf("\nThe required equation is y=%.4f+%.4fx1+%.4fx2",Y(1,1),Y(2,1),Y(3,1));
endfunction
x1=[18,22,15,10,30,32,35,42];
x2=[2,3,1,1,3,4,4,5];
y=[4,5,3,2,6,7,8,9];
regressionmulti(x1,x2,y)
```

OUTPUT:

```
The required equation is y = 0.0503 + 0.1577x1 + 0.4966x2
```

Practical : 9

1.Title:

Programs on Random variables and Distribution.

2. Prior Concepts:

1. Formula of random variables.
2. Formula on continuous random variables.
3. Probability concepts.
4. Formula on Binomial distribution to calculate probability.
5. Formula on Poisson distribution to calculate probability.

3. New Concepts:

1. Problems based on random variables.
2. Problems based on continuous random variables.
3. Problems based on Binomial distribution.
4. Problems based on Poissons distribution.

4. Objectives:

1. Students should be able to solve all problems based on probability.

5. Procedure:

1.Define a function.

2.Use for loop.

3. Calling a function by passing parameters in it.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on finding probabilities in real life..

9. Questions:

1. Program to fit Binomial distribution

```
function [expfre]=binfit(x, obsfre)
   n=length(x)-1;
   N=sum(obsfre);
```

```
    xbar=sum(x*obsfre)/N;
    p=xbar/n;
    q=1-p;
    for r=0:n
        prob(r+1)=factorial(n)*p^r*q^(n-r)/(factorial(r)*factorial(n-r));
    end

    expfre=round(prob*N);
    printf('Expected Frequencies\n');
    prinf('----------------------\n');
    return(expfre);
endfunction

x=[0,1,2,3,4,5]

obsfre=[12,25,36,21,10,8]

binfit(x,obsfre)
```

O/P:

Expected Frequencies :

```
5
27
65
84
62
24
4
```

2.Program to fit Poisson Fitting distribution :

```
function [expfre]=poissonfit(x, obsfre)
    n=length(x)-1;
    N=sum(obsfre);
    xbar=(x*obsfre)/N;
    m=xbar;
    prob(1)=exp(-m);
    for i=1:n
        prob(i+1)=m*prob(i)/(i);

    end

    expfre=round(prob*N);
    printf('Expected Frequencies\n');
    printf('---------------------\n');

    return(expfre);
 endfunction
```

```
x=[0,1,2,3,4,5]

obsfre=[100,70,45,20,10,5]

poissonfit(x, obsfre)
```

O/P:

Expected Frequencies :

```
80
91
52
20
6
1
```

3.Program to generate random number using binomial distribution

```
function [x]=binrand(N, n, pr)
   for i=1:N
      p=rand();
      q=1-p;
      x(i)=round(cdfbin("S",n,pr,1-pr,p,q));

   end
   return(x)
endfunction
```

O/P:

```
binrand(10,8,0.9)
   6.
   7.
   6.
   6.
   6.
   8.
   7.
   6.
   8.
   6.
```

4.Program to generate Random number using Poisson Distribution

```
function [x]=poirand(N, m)
   for i=1:N
      p=rand();
```

```
        q=1-p;
        x(i)=round(cdfpoi("S",m,p,q));
    end
    return(x);
endfunction
```

O/P

```
poirand(20,5)
    5.
    3.
    3.
    5.
    2.
    5.
    5.
    3.
    1.
    4.
    4.
    3.
    4.
    4.
    3.
    2.
    5.
    3.
    6.
    4.
```

Practical : 10

1.Title:

Programs on Distribution.

2. Prior Concepts:

1. Formula based on uniform distribution.
2. Formula based on normal distribution.
3. Formula based on exponential distribution.

3. New Concepts:

1. Problems based on uniform distribution.
2. Problems based on normal distribution.
3. Problems based on exponential distribution

4. Objectives:

1. Students should be able to solve all based on normal, uniform, exponential distribution.

5. Procedure:

1.Define a function.

2.Use for loop.

3. Calling a function by passing parameters in it.

6. Implementation :

Done by students in the lab.

7. Results:

O/P of the program

8. Application:

Based on Normal distribution.

9. Questions:

1.Program for negative binomial distribution

```
function [expfre]=negbinfit(x, obsfre)
   n=length(x)-1;
   N=sum(obsfre);
   xbar=sum(x*obsfre')/N;
   ran=sum(obsfre*(x^2))/N;
   var=ran-xbar^2;
   p=xbar/var;
   q=1-p;
```

```
    r=round(p*xbar/q);
    prob(1)=p^r;
    for i=1:n
        prob(i+1)=((i-1+r)/(i-1+1))*q*prob(i);
    end

    printf('Expected Frequencies\n');
    printf('----------------------\n');
    expfre=round(prob*N);
    return(expfre);
endfunction

x=[0,1,2,3,4,5]

obsfre=[12,50,150,200,120,45,10]

negbinfit(x,obsfre);
```

O/P;
Expected Frequencies :

```
8
52
133
183
142
58
10
```

2.Program to generate random number using Uniform Distribution

```
function [x]=unirand(N, a, b)
    for i=1:N
        u=rand();
        x(i)=(b-a)*u+a;
    end;
    return(x);
endfunction
```

O/P:

```
unirand(40,2,8)

    3.7222405
    5.901677
    2.5288009
    4.699258
    6.3363518
    7.3860778
    3.4566931
    4.6026327
    7.8062319
    5.0411207
    5.1397858
    5.3581686
    5.3703842
    4.809056
    6.676728
    6.7406431
    7.8851252
    6.9122397
    4.5541234
    3.4769363
    7.5377195
    2.6004475
    4.8069309
    4.3702986
    2.2196703
    5.1052211
    6.9952709
    5.6628993
    3.1226671
    2.1137449
    7.0601389
    2.4491569
    7.1196891
    2.0747541
    3.1205236
    4.9523504
    6.4937649
    7.6489742
    3.2744334
    5.477012
```

3.Program to generate random number using exponential distribution

```
function [x]=exporand(N, m)
    for i=1:N
        u=rand();
        x(i)=-m*log(1-u);
    end
    return(x);
endfunction
```

O/P:

```
exporand(25,800)
 ans  =

    107.93057
    1242.3935
    338.86772
    247.38145
    2831.8674
```

1748.0178
185.21813
1531.229
897.83999
1974.7931
23.11966
216.17151
967.28036
102.49642
1411.6636
303.96339
604.90209
678.00427
39.185442
1393.7447
693.78013
261.89358
2472.2294
1901.1185
327.59649