**ALGORITHMS**

**ROOTS OF QUADRATIC EQUATION**

Read a, b, c, d

If (a==0) then

Print("Not a Quadratic Equation")

If(b≠0) then

x = −c/b

Print (x)

end if

Else

D =b∗b −4∗a∗c

If(D==0) then

Print("Roots are real and equal")

alpha = −b/(2∗a)

beta  = alpha

end if

if(D>0) then

Print("Roots are real and distinct")

alpha = (−b + sqrt(D))/(2∗a)

beta = (−b − sqrt(D))/(2∗a)

end if

If (D<0) then

Print("Roots are imaginary")

xReal = −b/(2∗a)

xImag = sqrt(abs(D))/(2∗a)

alpha = xReal + i\*xImag

beta = xReal - i\*xImag

end if

Print(x1 = alpha, x2 = beta )

end if

End

**SYSTEM OF SYMULTANEOUS EQUATION**

Read a, b, c, p, q and r

Compute

q = q−((p∗b)/a)

r = r−((p∗c)/a)

If  (q = 0) then

Print("No solution")

STOP

else

y = r/q

x = (c−b∗y)/a

Print(x, y)

End if

END

**ADDITION OF FLOATING POINT**

Read x1, e1, x2 and e2

Set  k = | e1-e2 |

If (e1>e2) then

x2 = x2/(10^k)

e = e1

else

x1 = x1/(10^k)

e = e2

End if

Set  x = x1 +x2

If (x >= 1.0) then

x = x/10

e = e+1

End if

If (e>99) then

Print ("Overflow")

STOP

End if

Print(x, e)

END

**SUBTRACTION OF FLOATING POINT**

Read x1, e1, x2 and e2

Set  k = | e1-e2 |

If (e1>e2) then

x2 = x2/10k

e = e1

else

x1 = x1/10k

e = e2

End if

Set  x = x1 - x2

While ((abs(x)<0.1) and (abs(x)>0.0))  do

x = x\*10

e = e-1

End while

If (e < -99) then

Print ("Underflow")

STOP

End if

Print(x, e)

END

**MULTIPLICATION OF FLOATING POINT**

Read x1, e1, x2 and e2

Set  x = x1 \* x2

Set  e = e1 + e2

If (abs(x) >= 1.0) then

x =x/10

e = e+1

End if

If (abs(x) < 0.1) then

x = x\*10

e = e-1

End if

If (e < -99) then

Print ("Underflow")

STOP

End if

If (e  > 99) then

Print ("Overflow")

STOP

End if

Print(x, e)

END

**DIVISION OF FLOATING POINT**

Read x1, e1, x2 and e2

Set  x = x1/x2

Set  e = e1 - e2

If (abs(x) >= 1.0) then

x = x/10

e = e+1

End if

If (abs(x) < 0.1) then

x = x\*10

e = e-1

End if

If (e < -99) then

Print ("Underflow")

STOP

End if

If (e  > 99) then

Print ("Overflow")

STOP

End if

Print(x, e)

END

**BISECTION METHOD**

Read a. b,  E

If ( f(a)\*f(b) < 0 ) then

Set  m = (a+b)/2

Set  i = 1

While ( |f(m)| > E ) do

Print i , a, b, m, f(m)

If (f(a)\*f(m) > 0) then

a = m

Else

b = m

End if

m = (a+b)/2

i = i  + 1

End while

Print i,a, m, f(m)

End if

Print (Root = m)

END

**REGULA FALSI METHOD**

Read a. b,  E

If ( f(a)\*f(b) < 0 ) then

Set  m =  [a∗f(b)−b∗f(a)]/[f(b)−f(a)]

Set  i = 1

While ( |f(m)| > E ) do

Print i , a, b, m, f(m)

If (f(a)\*f(m) > 0) then

a = m

Else

b = m

End if

m = [a∗f(b)−b∗f(a)[/[f(b)−f(a)]

i = i  + 1

 End while

Print i, a, m, f(m)

End if

Print (Root = m)

END

**SECANT METHOD**

Read a, b, E

Set  m = [a\*f(b)-b\*f(a)]/[f(b)-f(a)]

Set  i = 1

While ( |f(m)| > E ) do

Print i, a , b, m, f(m)

a = b, b = m

m =[a∗f(b)−b∗f(a)]/[f(b)−f(a)]

  i = i + 1

End while

Print i, a, b, m, f(m)

Print (Root = m)

END

**NEWTON RAPHSON'S METHOD**

Read b, E

Set  m = b – [(f(b)/f’(b))]

Set  i = 1

While ( |f(m)| > E ) do

Print i, b, m, f(m)

b = m

m = b – [(f(b)/f’(b))]

i = i + 1

End while

Print i, a, b, m, f(m)

Print (Root = m)

END

**FIXED POINT ITERATION METHOD**

Read b, E

Set  m =  g(b)

Set  i = 1

While ( |(b-m)| > E ) do

Print i, b, m, |(b-m)|

b = m

i = i + 1

End while

Print i, a, b, m, |(b-m)|

Print (Root = m)

END

**GAUSS ELIMINATION METHOD**

Read number of equations say n

// Read equation

For i = 0 to (n−1)in steps of 1 do

For j = 0 to n in steps of 1 do

Read a[i][j]

End for

End for

// Forward Elimination

For k = 0 to (n−2)in steps of 1 do

For i = (k+1)to (n−1)in steps of 1 do

u =a[i][k]/a[k][k]

For j = k to n in steps of 1 do

a[i][j]= a[i][j]− (a[k][j]∗u)

End for

End for

End for

// Backward Substitution

x[n−1]=a[n−1][n]/a[n−1][n−1]

For i = n−2 to 0 in steps of −1 do

Sum = 0.0

For j = (i+1)to (n−1)in steps of 1 do

Sum = sum = a[i][j]∗x[j]

End for

x[i]=[a[i][n]−sum]/a[i][i]

End for

*END*

**GAUSS JORDAN METHOD**

Read number of equations say n

// Read equation

For i = 0 to (n−1)in steps of 1 do

For j = 0 to n in steps of 1 do

Read a[i][j]

End for

End for

// formation of diagonal matrix

For k = 0 to (n−1)in steps of 1 do

For i =0  to (n−1)in steps of 1 do

u =a[i][k]/a[k][k]

*If (i ≠ k) then*

For j = k to n in steps of 1 do

a[i][j]= a[i][j]− (a[k][j]∗u)

End for

*End if*

End for

End for

// Calculate values

*For i = 0 to (n-1) in steps of 1 do*

X[i]=a[i][n]/a[i][i]

*End for*

*END*

**GAUSS JACOBI METHOD**

Read number of equations say n

// Read equation

For i = 0 to (n−1)in steps of 1 do

For j = 0 to n in steps of 1 do

Read a[i][j]

End for

End for

// Perform Iteration

*Read maxIteration and error*

*For k = 0 to (maxIteration-1) in steps of 1 do*

*big\_error = 0.0*

*For i = 0 to (n-1) in steps of 1 do*

*sum = 0.0*

*For j = 0 to (n-1) in steps of 1 do*

*If (i ≠ j) then*

sum += a[i][j]∗oldx[j]

End if

End for

newx[i]=(a[i][n]−sum)/a[i][i]

E = abs((newx[i]−oldx[i])/newx[i])

If  (E > big\_error) then

big\_error = E

End for

If (rel\_error <= e) then

Print "Solution is convergent. It converges in (k) iterations"

Print new\_x

STOP

End if

For i  = 0 to (n-1) in steps of 1 do

oldx[i]= newx[i]

End for

End for

Print "Solution is not convergent in maxIteration"

Print new\_x

END

**GAUSS SEIDEL METHOD**

Read number of equations say n

// Read equation

For i = 0 to (n−1)in steps of 1 do

For j = 0 to n in steps of 1 do

Read a[i][j]

End for

End for

// Perform Iteration

*Read maxIteration and error*

*For k = 0 to (maxIteration-1) in steps of 1 do*

*big\_error = 0.0*

*For i = 0 to (n-1) in steps of 1 do*

*sum = 0.0*

*For j = 0 to (n-1) in steps of 1 do*

*If (i ≠ j) then*

sum += a[i][j]∗x[j]

End if

End for

temp = (a[i][n]-sum)/a[i][i]

E = abs((temp−x[i])/temp)

x[i] = temp

If  (E > big\_error) then

big\_error = E

End for

If (rel\_error <= e) then

Print "Solution is convergent. It converges in (k) iterations"

Print x

STOP

End if

End for

Print "Solution is not convergent in maxIteration"

Print x

END

**FORWARD DIFFERENCE TABLE**

Read n

// Read elements

For i = 0 to (n-1) in steps of 1 do

Read  y[i]

End for

// Create forward difference table

For j = o to (n-2) in steps of 1 do

For j = 0 to (n-j-2) in steps of 1 do

If (j == 0) then

d[i][j] = y[i+1] - y[i]

Else

d[i][j] = d[i+1][j-1] - d[i][j-1]

End if

End for

End for

End

**DIVIDED DIFFERENCE TABLE**

Read n

// Read elements

For i = 0 to (n-1) in steps of 1 do

Read  x[i], y[i]

End for

// Create divided difference table

For j = o to (n-2) in steps of 1 do

For j = 0 to (n-j-2) in steps of 1 do

If (j == 0) then

d[i][j]=(y[i+1]−y[i])/(x[i+1]−x[i])

Else

d[i][j] = (d[i+1][j−1]−d[i][j−1])/(x[i+j+1]−x[i])

End if

End for

End for

END

**NEWTON’S FORWARD DIFFERENCE INTERPOLATION**

Read n

// Read elements

For i = 0 to (n−1)in steps of 1 do

Read  xarr[i], yarr[i]

End for

// Create forward difference table

For j = o to (n−2)in steps of 1 do

For j = 0 to (n−j−2)in steps of 1 do

If (j == 0)then

d[i][j]= yarr[i+1]− yarr[i]

Else

d[i][j]= d[i+1][j−1]− d[i][j−1]

End if

End for

End for

//Forward Interpolation Formula

Read x

Set h = xarr[1]− xarr[0]

Set u = x –(xarr[0]/h)

Set result = yarr[0]

For i = 0 to (n−2)in steps of 1 do

result += (num(u, i)/fac(i+1))∗d[0][i]

End for

Print result

*END*

**NEWTON’S BACKWARD DIFFERENCE INTERPOLATION**

Read n

// Read elements

For i = 0 to (n-1) in steps of 1 do

Read  = xarr[i],  yarr[i]

End for

// Create Backward difference table

For j = o to (n-2) in steps of 1 do

For j = 0 to (n-j-2) in steps of 1 do

If (j == 0) then

d[i][j] = yarr[i+1]− yarr[i]

Else

d[i][j] = d[i+1][j−1]− d[i][j−1]

End if

End for

End for

// Backward Interpolation Formula

Read x

Set h =xarr[1]− xarr[0]

Set u =x –(xarr[n−1])/h

Set result =yarr[n−1]

For i = 0 to (n-2) in steps of  1 do

result += (num(u, i)/fac(i+1))∗d[n−2−i][i]

End for

Print result

END

**NEWTON’S DIVIDED DIFFERENCE INTERPOLATION**

Read n

// Read elements

For i = 0 to (n-1) in steps of 1 do

Read  x[i], y[i]

End for

// Create divided difference table

For j = o to (n-2) in steps of 1 do

For j = 0 to (n-j-2) in steps of 1 do

If (j == 0) then

d[i][j]=(y[i+1]−y[i])/(x[i+1]−x[i])

Else

d[i][j] = (d[i+1][j−1]−d[i][j−1])/(x[i+j+1]−x[i])

End if

End for

End for

// Divided Difference Interpolation Formula

Read x

Set  result = yarr[0]

For i = 0 to (n-2) in steps of 1 do

result += term(x, xarr, i)∗d[0][i]

End for

Print result

END

**LAGRANGEE’S INTERPOLATION**

Read n

For i = 0 to (n-1) in steps of 1 do

Read xarr[i],  yarr[i]

End for

Read x

Set result = 0.0

For i = 0 to (n-1) in steps of 1 do

result += (num(x, i, xarr)/deno(i, xarr))∗yarr[i]

End for

Print result

END

**INVERSE INTERPOLATION**

Read n

For i =  to (n-1) in steps of 1 do

Read xarr[i],  yarr[i]

End for

Read x

Set result = 0

For i = 0 to (n-1) in steps of 1 do

result += (num(y, i, yarr)/deno(i, yarr))∗xarr[i]

End for

Print result

END

**TRAPEZOIDAL RULE**

Read function, lower limit a, upper limit b, number of sub-intervals n

Set h = (b – a)/n

sum = f(a)+ f(b)

For i  = 1 to (n-1) in steps of 1 do

sum += 2∗f(a+i∗h)

End for

Integral  = (sum∗h)/2

Print integral

END

**SIMPSON’S 1/3rd RULE**

Read function f, lower limit a, upper limit b, number of sub-intervals n

Set h = (b – a)/n

sum =f(a)+ f(b)

For i = 1 to (n-1) in steps of 1 do

If (i%2==0)then

sum += 2∗f(a+i∗h)

Else

sum += 4∗f(a+i∗h)

End if

 End for

Integral = (sum∗h)/3

Print integral

END

**SIMPSON’S 3/8 RULE**

Read function f, lower limit a, upper limit b, number of sub-intervals n

Set h = (b – a)/n

sum = f(a)+ f(b)

For i = 1 to (n-1) in steps of 1 do

If (i%3==0)then

sum += 2∗f(a+i∗h)

Else

 sum += 3∗f(a+i∗h)

End if

 End for

Integral =(sum∗h∗3)/8

 Print integral

END

**GAUSSIAN QUADRATURE FORMULA METHOD**

Read function f, lower limit a, upper limit b, number of points n

Initialize Values dictionary(key−value pair)of w1, w2, w3,……wn and x1, x2, x3,……xn with its predefined values

Set  p =(a+b)/2

Set q = (b-a)/2

Set integral = 0.0

For i = 0 to n in steps of 1 do

F = f(p+q∗values[n][i])

integral += (Values[n][i])∗F

End for

Integral = q∗integral

Print integral

END

**EULAR’S METHOD**

Read x0, y0, h,  xn

Set  x = x0

Set  y = y0

Set  i = 1

Print i, x, y

While(x <= xn) do

y = y + h∗f(x, y)

x = x + h

i = i+1

Print i, x, y

End while

END

**MODIFIED EULAR’S METHOD**

Read  x0, y0, xn, h, error = 0.0001

Set  x = x0

Set  y = y0

Set  i = 1

Print i, x, y

While (x <= xn)

Set y1 = y  + h∗f(x, y)

do

Set  y0 = y1

y1 = y + (h/2)∗[f(x, y)+ f(x+h, y1)]

While (|(y1-y0)/y1| > error) do

End while

Set y = y1

Set x = x + h

Set i = i + 1

Print i, x, y

End while

END

**RANGE KUTTA 4th ORDER METHOD**

Read  x0, y0, xn, h

While (x0 <=xn) do

Print x0,y0

k1 = h\*f(x0,y0)

k2 = h\*f(x0+0.5\*h, y0+0.5\*k1)

k3 = h\*f(x0+0.5\*h, y0+0.5\*k2)

k4 = h\*f(x0+h, y0+k3)

k = (k1+2\*k2+2\*k3+k4)/6

x0 =x0 + h

y0 = y0 + k

End while

Print x0,y0

END

**RANGE KUTTA 2nd ORDER METHOD**

Read  x0, y0, xn, h

While (x0 <=xn) do

Print x0,y0

k1 = h\*f(x0,y0)

x0 =x0 + h

k2 = h\*f(x0, y0+k1)

y0 = y0 + 0.5\*(k1+k2)

End while

Print x0,y0

END

SNIGDHA RAI

IT-2K19-59

MTECH IT 4TH SEM