Numerical Analysis Lab

MD ARIF HOSSAIN

Department of Mathematics, Shahjalal University of Science and Technology, Sylhet, Bangladesh.

November 29, 2022

Abstract

This paper contains a collection of numerical method code written in Fortran.As a compiler, we used codeblocks 20.03.

Introduction

Numerical analysis is the study of algorithms that use numerical approximation (as opposed to symbolic manipulations) for the problems of mathematical analysis (as distinguished from discrete mathematics). It is the study of numerical methods that attempt at finding approximate solutions of problems rather than the exact ones. Numerical analysis finds application in all fields of engineering and the physical sciences, and in the 21st century also the life and social sciences, medicine, business and even the arts. Current growth in computing power has enabled the use of more complex numerical analysis, providing detailed and realistic mathematical models in science and engineering. Examples of numerical analysis include: ordinary differential equations as found in celestial mechanics (predicting the motions of planets, stars and galaxies), numerical linear algebra in data analysis,and stochastic differential equations and Markov chains for simulating living cells in medicine and biology.

Contents

1 Root Finding

1.1 Bisection Method

1.1.1 Source Code

```
PROGRAM bisection
    IMPLICIT NONE
    REAL :: a,b,tol,er,p,f
    INTEGER :: i=1
    WRITE (*,*) "Enter [a,b] of the given equation"!--Take 0 and 1 as a interval--!
    READ (*,*) a,b
    WRITE (*,*) "The tolerance up to"
    READ (*,*) tol
    IF ((f(a)*f(b))== 0) THEN
        IF ((f(a)==0).and.(f(b)==0)) THEN
            WRITE (*,*) 'The roots are',a,' and ',b
        ELSE IF (f(a)==0) THEN
            WRITE (*,*) 'The root is',a
        ELSE IF (f(b)==0) THEN
            WRITE (*,*) 'The root is',b
        END IF
    ELSE IF ((f(a)*f(b))>0) THEN
        WRITE (*,*) 'The root does not exist in  ',a,' and ',b
    ELSE IF ((f(a)*f(b))<0) THEN
        open(10,file="output.txt")!--Here it will open a file named output.txt--!
        DO
            p=(a+b)/2
            IF (f(p)==0) EXIT
            er=abs((a-b)/a)
            WRITE (10,*) "Total Steps =",i,"The root is",p
            IF (er<((tol))) EXIT
            IF ((f(p)*f(a))<0) THEN

                b=p
            ELSE IF ((f(p)*f(b))<0) THEN
                a=p

            END IF
            i = i + 1
        END DO
    END IF

END PROGRAM

REAL FUNCTION f(x)
    f=x*exp(x)-1
END FUNCTION f
```

Listing 1: Bisection Method

```
 Total Steps =            1 The root is   0.500000000
 Total Steps =            2 The root is   0.750000000
 Total Steps =            3 The root is   0.625000000
 Total Steps =            4 The root is   0.562500000
```

```
Total Steps =           5 The root is  0.593750000
Total Steps =           6 The root is  0.578125000
Total Steps =           7 The root is  0.570312500
Total Steps =           8 The root is  0.566406250
Total Steps =           9 The root is  0.568359375
Total Steps =          10 The root is  0.567382812
Total Steps =          11 The root is  0.566894531
Total Steps =          12 The root is  0.567138672
Total Steps =          13 The root is  0.567260742
Total Steps =          14 The root is  0.567199707
Total Steps =          15 The root is  0.567169189
Total Steps =          16 The root is  0.567153931
```

Listing 2: Bisection Method Output

1.2 Regula Falsi Method

1.2.1 Source Code

```
PROGRAM regulafalsi
    REAL :: a,b,tol,p
    INTEGER :: i=1
   !!open(10,file="regulaFalsi.txt")!!--input from the file named
    regula falsi.txt--!!
    WRITE(*,*) "ENTER THE INTERVAL "
    READ (*,*) a,b!!--0 and 1 is the boundary value of this
    function where the root lies and tol is tolerance upto--!!
    WRITE(*,*)"ENTER THE TOLERANCE "
    READ(*,*)TOL
    open(11,file="regulaFalsioutput.txt")
    IF ((f(a)*f(b))>0) THEN
        WRITE (*,*) 'The root does not exist in  ',a,' and ',b
    ELSE IF ((f(a)*f(b))<0) THEN
        DO
            p=(a*f(b)-b*f(a))/(f(b)-f(a))
            IF (f(p)==0) EXIT

            WRITE (11,*) 'Total Steps =',i,'The root is',p
            IF (abs((a-b)/a)<((tol))) EXIT
                IF ((f(p)*f(a))<0) THEN

                b=p
                ELSE IF ((f(p)*f(b))<0) THEN
                a=p
            END IF
            i = i + 1
        END DO

   END IF

END PROGRAM

REAL FUNCTION f(x)
    f=x*exp(x)-2
```

```
END FUNCTION f
```

Listing 3: Regula Falsi

```
 Total Steps =            1 The root is  0.735758901
 Total Steps =            2 The root is  0.839520812
 Total Steps =            3 The root is  0.851183891
 Total Steps =            4 The root is  0.852451563
 Total Steps =            5 The root is  0.852588832
 Total Steps =            6 The root is  0.852603674
 Total Steps =            7 The root is  0.852605283
 Total Steps =            8 The root is  0.852605402
 Total Steps =            9 The root is  0.852605462
```

Listing 4: Bisection Method Output

1.3 Iterative Method

1.3.1 Source Code

```
program iteration

    real :: x,x0,e
    integer :: i=1,n
    !open(10,file='iterationMethod.txt')
    open(11,file='iterationMethodOutput.txt')
    write(*,*)"Enter the initial guess."
    read(*,*)x0!!--x0 is initial guess,n is number of steps and e
    is tolerance up to--!!
    write(*,*)"Enter the tolerance"
    read(*,*)e!!--0.0001--!

    do
        x=g(x0)
        write(11,*)"number of steps = ",i," The root x is : ",x
        IF (abs(x-x0)<=e) EXIT
        x0=x
        i=i+1
    end do

end program

real function f(x)
    real :: x
    f=x*exp(x)-1
    return
end function

real function g(x)
    real :: x
    g=exp(-x)
    return
end function
```

Listing 5: Iterative Method

```
number of steps =            1  The root x is :   0.367879450
number of steps =            2  The root x is :   0.692200601
number of steps =            3  The root x is :   0.500473499
number of steps =            4  The root x is :   0.606243551
number of steps =            5  The root x is :   0.545395792
number of steps =            6  The root x is :   0.579612315
number of steps =            7  The root x is :   0.560115457
number of steps =            8  The root x is :   0.571143091
number of steps =            9  The root x is :   0.564879358
number of steps =           10  The root x is :   0.568428695
number of steps =           11  The root x is :   0.566414773
number of steps =           12  The root x is :   0.567556620
number of steps =           13  The root x is :   0.566908896
number of steps =           14  The root x is :   0.567276239
number of steps =           15  The root x is :   0.567067921
number of steps =           16  The root x is :   0.567186058
number of steps =           17  The root x is :   0.567119062
```

Listing 6: Iterative Method Output

1.4 Newton Raphson Method

1.4.1 Source Code

```
program newtonRaphson
    integer::i=1
    real::x,x0,e
    !open(10,file="nr.txt")!!--Reading the file from the file nr.
    txt--!!
    open(11,file="nroutput.txt")
    WRITE(*,*)"Enter the initial guess"
    READ(*,*)x0!!--x0=2 is initial guess,e is tolerance up to--!!
    WRITE(*,*)"Enter the tolerance"!!--0.0001--!
    read(*,*)e

    do
        x=x0-(f(x0)/df(x0))
        write(11,*)"Number of steps ",i, " the root x is =",x
        IF (ABS(x-x0)<=e) EXIT
        x0=x
        i=i+1
    end DO

 end program

 real function f(x)
    real::x
    f=x**3-2*x-5
    return
 end

real function df(x)
    real::x
    df=3*x**2-2
    return
```

```
end
```

Listing 7: Newton Raphson Method

```
 Number of steps              1   the root x is =    2.09999990
 Number of steps              2   the root x is =    2.09456801
 Number of steps              3   the root x is =    2.09455156
```

Listing 8: Newton Rhapson Method Output

2 Interpolation And Extrapolation

2.1 Finite Difference

2.1.1 Finite Forward Difference

x	y
45	2.871
50	2.404
55	2.083
60	1.862
65	1.712

2.1.2 Source Code

Data Value in file given as below
5
45
2.871
50
2.404
55
2.083
60
1.862
65
1.712

```
program finiteForwardDifference
    real x(10),y(10,10)
    real i,j,n
    print*,'enter number of data'

    open(10,file="data.txt")!!--The data will read from the file
    named data--!!
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
```

```
        !print*,'y[',i,']='
        read(10,*),y(i,1)
    end do
    do j=2,n
        do i=1,n-j+1
            y(i,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'Forward difference table:'
    do i=1,n
        print"(10f10.3)",x(i),(y(i,j),j=1,n-i+1)

    end do

end program
```

Listing 9: Finite Forward Difference

```
 Forward difference table:
    45.000     2.871    -0.467     0.146    -0.046     0.017
    50.000     2.404    -0.321     0.100    -0.029
    55.000     2.083    -0.221     0.071
    60.000     1.862    -0.150
    65.000     1.712

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.
```

Listing 10: Finite Forward Difference Output

2.1.3 Finite Backward Difference

x	y
1	1
2	8
3	27
4	64
5	125

2.1.4 Source Code

Data Value in file given as below
5
1
1
2
8
3
27
4
64
5

125

```
program finiteBackwardDifference
    real x(10),y(10,10)
    real i,j,n
   !print*,'enter number of data'

    open(10,file="data2.txt")
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*)y(i,1)
    end do
    do j=2,n
        do i=1,n-j+5
            y(i+1,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'Backward difference table:'
    do i=1,n
        print"(10f10.3)",x(i),(y(i,j),j=1,i)
    end do


end program
```

Listing 11: Finite Backward Difference

```
 Backward difference table:
       1.0       1.0
       2.0       8.0       7.0
       3.0      27.0      19.0      12.0
       4.0      64.0      37.0      18.0       6.0
       5.0     125.0      61.0      24.0       6.0       0.0

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

Listing 12: Finite Backward Difference Output

2.2 Newton Gregory Interpolation

2.2.1 Newton Forward Interpolation

x	y
45	2.871
50	2.404
55	2.083
60	1.862
65	1.712

2.2.2 Source Code

Data Value in file given as below
5
45
2.871
50
2.404
55
2.083
60
1.862
65
1.712
Find the value on $f(46)$.

```
program newtonForwardInterpolation
    real x(10),y(10,10)
    real i,j,n,p,h,u,poly,ut,fact!!--poly is declared for the total
     functioanl value --!!
    !print*,'enter number of data'

    open(10,file="data.txt")
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*),y(i,1)
    end do
    do j=2,n
        do i=1,n-j+1
            y(i,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'Forward difference table:'
    do i=1,n
        print"(10f10.3)",x(i),(y(i,j),j=1,n-i+1)

    end do
    print*,"Enter the data value you want"!!--for ex enter 46 --!
    read*,p
    h=x(2)-x(1)
    print*,"The gap between two number",h
    u=(p-x(1))/h
    ut=u
    poly=y(1,1)
    fact=1
    do i=2,n
        poly=poly+ut*y(1,i)/fact
        fact=fact*i
        ut=ut*(u-(i-1))
    end do
    print*,"The functional value of the demand value is",poly
```

```
end program
```

Listing 13: Newton Forward Interpolation

```
 Forward difference table:
     45.000      2.871     -0.467      0.146     -0.046      0.017
     50.000      2.404     -0.321      0.100     -0.029
     55.000      2.083     -0.221      0.071
     60.000      1.862     -0.150
     65.000      1.712
 Enter the data value you want
46
 The gap between two number   5.00000000
 The functional value of the demand value is   2.76314092

Process returned 0 (0x0)   execution time : 2.682 s
Press any key to continue.
```

Listing 14: Newton Forward Interpolation Output

2.2.3 Newton Backward Interpolation

x	y
1	1
2	8
3	27
4	64
5	125

2.2.4 Source Code

Data Value in file given as below
5
1
1
2
8
3
27
4
64
5
125
Find the value on $f(3.5)$

```
program newtonBackwardInterpolation
    real x(10),y(10,10)
    real i,j,n,p,h,v,poly,vt,factfun
    !print*,'enter number of data'

```

```
    open(10,file="data2.txt")
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*)y(i,1)
    end do
    do j=2,n
        do i=1,n-j+5
            y(i+1,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'Backward difference table:'
    do i=1,n
        print"(10f10.3)",x(i),(y(i,j),j=1,i)
    end do
    print*,"Enter the data value you want"
    read*,p
    h=x(2)-x(1)
    print*,"The gap between two number",h
    v=(p-x(n))/h
!   print*,"the value of y(1,2)",y(1,2)
    vt=v
    print*,"the value of v is",v
    poly=y(n,1)
    !print*,"the value stored in poly",poly
    fact=1
    do i=2,n
        poly=poly+vt*y(n,i)/fact
        fact=fact*i
        vt=vt*(v+(i-1))
    end do
    print*,"The functional value of the demand value is",poly
end program
```

Listing 15: Newton Backward Interpolation

```
 Backward difference table:
     1.000     1.000
     2.000     8.000     7.000
     3.000    27.000    19.000    12.000
     4.000    64.000    37.000    18.000     6.000
     5.000   125.000    61.000    24.000     6.000     0.000
 Enter the data value you want
3.5
 The gap between two number   1.00000000
 the value of v is  -1.50000000
 The functional value of the demand value is   42.8750000

Process returned 0 (0x0)   execution time : 1.260 s
Press any key to continue.
```

Listing 16: Newton Backward Interpolation Output

2.3 Newton Divided Interpolation

x	y
4	48
5	100
7	294
10	900
11	1210
13	2028

2.3.1 Source Code

Data Value in file given as below
6
4
48
5
100
7
294
10
900
11
1210
13
2028
Find the value on $f(8)$

```
program newtonDividedInterpolation
    real x(10),y(10,10)
    real i,j,n,f,poly
    !print*,'enter number of data'

    open(10,file="input.txt")
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*),y(i,1)
    end do
    do j=2,n
        do i=1,n-j+1
            y(i,j)=(y(i+1,j-1)-y(i,j-1))/(x(i+j-1)-x(i))
        end do
    end do
    print*,'Divided difference table:'
    do i=1,n
        print"(10f10.1)",x(i),(y(i,j),j=1,n-i+1)
    end do
```

```
    print*,"Enter the data value you want"!!--f(8)--so enter 8 to
    get the answer 448--!
    read*,f
    poly=y(1,1)
    !print*,"the value stored in poly",poly
    DO i=1,n-1
        p=1.0
            DO j=1,i
                p=p*(f-x(j))
            END DO
        poly=poly+p*y(1,i+1)
    END DO
    print"(A,F7.1)","The functional value of the demand value is",
    poly
end program
```

Listing 17: Newton Divided Interpolation

```
 Divided difference table:
    4.0      48.0      52.0      15.0       1.0       0.0       0.0
    5.0     100.0      97.0      21.0       1.0       0.0
    7.0     294.0     202.0      27.0       1.0
    10.0      900.0     310.0      33.0
    11.0     1210.0     409.0
    13.0     2028.0
 Enter the data value you want
8
The functional value of the demand value is  448.0

Process returned 0 (0x0)   execution time : 1.527 s
Press any key to continue.
```

Listing 18: Newton Divided Interpolation Output

2.4 Lagrange Polynomial

x	y
1	4
2	5
7	5
8	4

2.4.1 Source Code

Data Value in file given as below
4
1
4
2
5
7
5

8
4
Find the value on $f(6)$

```
program lagrangian_polynomial
    real x(10),y(10),p,k,s
    integer i,j,n

!print *,'Number of terms?'
    open(10,file="lagrange_input.txt")
    read(10,*)n
    do i=1,n
        read(10,*),x(i)
        read(10,*),y(i)
    end do
    print*,"The given data values are:"
    do i=1,n
        print*," x ",i," = ",x(i)," y ",i,"=",y(i)

    end do
    print *,"enter the data point to calculate the value"
    READ(*,*)k!!--Enter the value 6 to get f(6)=5.66--!!

    do i=1,n
        p=1.0
        do j=1,n
            if(i .ne. j) then
            p=p*((k-x(j))/(x(i)-x(j)))
            end if
        end do
        s=s+(p*y(i))
    end do
print *,"the value of that point ",k ,"is",s

end program
```

Listing 19: Lagrange Polynomial

```
 The given data values are:
  x  1  =    1.00000000      y  1 =   4.00000000
  x  2  =    2.00000000      y  2 =   5.00000000
  x  3  =    7.00000000      y  3 =   5.00000000
  x  4  =    8.00000000      y  4 =   4.00000000
 enter the data point to calculate the value
6
 the value of that point    6.00000000     is   5.66666698

Process returned 0 (0x0)   execution time : 1.931 s
Press any key to continue.
```

Listing 20: Lagrange Polynomial Output

3 Numerical Differentiation

3.1 Derivative using Newton Forward Difference formula

x	y
3.0	-14.00
3.2	-10.032
3.4	-5.296
3.6	0.256
3.7	6.672
4.0	14.00

3.1.1 Source Code

Data Value in file given as below
6
3
-14.00
3.2
-10.032
3.4
-5.296
3.6
0.256
3.8
6.672
4
14
Find the first and second derivative of the function tabulated below, at the point $x = 3$.

```
program forward_differentiation
    real x(10),y(10,10)
    real i,j,n,h,diff,sm,fact,p,term,second_diff
    !print*,'enter number of data'
    open(10,file="data.txt")
    read(10,*),n
    do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*),y(i,1)
    end do
    do j=2,n
        do i=1,n-j+1
            y(i,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'Forward difference table:'
    do i=1,n
        print"(10f10.3)",x(i),(y(i,j),j=1,n-i+1)
```

```
    end do
    fact=1
    sm=0
    h=x(2)-x(1)
    !sm=sm/h
    do i=1,n
        term=y(1,i+1)/i
        sm=sm+fact*term
        fact=-fact
    end do
    diff=sm/h
    do i=1,1
         second_diff=y(1,i+2)-y(1,i+3)+(11/12)*y(1,i+4)-(5/6)*y(1,i
    +5)
         second_diff=second_diff*(1/h**2)
    end do
    write(*,"(A,F6.2,1x,A,F6.2)")"The first derivative of tabulated
     value on",x(1),"is",diff
    write(*,"(A,F6.2,1x,A,F6.2)")"The second derivative of
    tabulated value on",x(1), "is",second_diff

end program
```

Listing 21: Numerical Forward Differentiation

```
 Forward difference table:
    3.000  -14.000    3.968   0.768    0.048   -0.000   0.000
    3.200  -10.032    4.736   0.816    0.048    0.000
    3.400   -5.296    5.552   0.864    0.048
    3.600    0.256    6.416   0.912
    3.800    6.672    7.328
    4.000   14.000
The first derivative of tabulated value on  3.00 is 18.00
The second derivative of tabulated value on  3.00 is 18.00

Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

Listing 22: Numerical Forward Differentiation Output

3.2 Derivative using Newton backward Difference formula

x	y
1.4	4.0552
1.6	4.9530
1.8	6.0496
2.0	7.3891
2.2	9.0250

3.2.1 Source Code

Data Value in file given as below
5

1.4
4.0552
1.6
4.9530
1.8
6.0496
2.0
7.3891
2.2
9.0250
Find the first and second derivative of the function tabulated below, at the point $x = 2.2$.

```
program backward_differentiation
    real x(10),y(10,10)
    real i,j,n,h,diff,sm,fact,p,term,second_diff
    !print*,'enter number of data'
    open(10,file="data.txt")
    read(10,*),n
   do i=1,n
        !print*,'X[',i,']='
        read(10,*),x(i)
        !print*,'y[',i,']='
        read(10,*)y(i,1)
    end do
    do j=2,n
        do i=1,n-j+10
             y(i+1,j)=y(i+1,j-1)-y(i,j-1)
        end do
    end do
    print*,'backward difference table:'
    do i=1,n
        print"(10f10.4)",x(i),(y(i,j),j=1,i)
    end do
    sm=0
    h=x(2)-x(1)
    !sm=sm/h
    do i=1,n
        term=y(n,i+1)/i
        sm=sm+term
    end do
    second_diff=0
    diff=sm/h
    do i=n,n
         second_diff=y(n,n-2)+y(n,n-1)+(11/12)*y(n,n)
    end do
    second_diff=(1/h**2)*second_diff
    write(*,"(A,F6.2,1x,A,F6.2)")"The first derivative of tabulated
     value on ",x(n),"is",diff
    write(*,"(A,F5.2,1x,A,F6.2)")"The second derivative of
    tabulated value on ",X(n),"is",second_diff

end program
```

Listing 23: Numerical Backward Differentiation

```
 backward difference table:
    1.4000    4.0552
    1.6000    4.9530    0.8978
    1.8000    6.0496    1.0966    0.1988
    2.0000    7.3891    1.3395    0.2429    0.0441
    2.2000    9.0250    1.6359    0.2964    0.0535    0.0094
The first derivative of tabulated value on   2.20 is  9.02
The second derivative of tabulated value on  2.20 is  8.75

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

Listing 24: Numerical Backward Differentiation Output

4 Numerical Integration

We know there are 4 kind of Integration Rule
Trapezoidal rule is for any no. of sub interval.
Simspon's 1/3 rule is for $n = 2$ or it's multiple no. of sub intervals.
Simspon's 3/8 rule is for $n = 3$ or it's multiple no. of sub intervals.
Weddle's Rule is for $n = 6$ or it's multiple no.of sub intervals.
We will use the modular arithmetic concepts to solve these rules in Fortran code.

4.1 Technique to solve each rule

1.Trapezoidal Rule

$$\int_{x_0}^{x_0+nh} y\,dx = h/2[(y_0 + y_n) + 2(y_1 + y_2 + y_3 + y_4 + ... + y_{n-1})]$$

We will store the y_a and y_b value and the rest value will multiply by 2.and this total will multiply by $h/2$.Thus we will get the desired output value.

2.Simpson 1/3 Rule

$$\int_{x_0}^{x_0+nh} y\,dx = h/3[(y_0+y_n)+4(y_1+y_3+.......+y_{n-1})+2(y_2+y_4+.....+y_{n-2})]$$

We will store the y_a and y_b value and the rest value will multiply by 2.and this total will multiply by $h/2$.Thus we will get the desired output value.

$2n+1 \cdot \cdot 7\ 5\ 3\ 1 \qquad 0\ 2\ 4\ 6 \cdot \cdot 2n$

we will take $mod(n, 2) = [0]$ we will multiply it by 2 and the rest value we will multiply it by 4 then store in the total sum.

3.Simpson 3/8 Rule

$$\int_{x_0}^{x_0+nh} y\,dx = 3h/8[(y_0+y_n)+3(y_1+y_2+y_4+y_5.......+y_{n-1})+2(y_3+y_6+.....+y_{n-3})]$$

We will store the y_a and y_b value and the rest value will multiply by 2.and this total will multiply by $3h/8$.Thus we will get the desired output value.

$3n$
.
.
6
3
0

2 1
5 4
. . 8 7 . .
$3n+2$ $3n+1$

we will take $mod(n, 3) = [0]$ we will multiply it by 2 and the rest value we will multiply it by 3 then store in the total sum.

4.Weddle's Rule

$$\int_{x_0}^{x_0+nh} y\,dx = 3h/10[(y_0+5y_1+y_2+6y_3+y_4+5y_5+2y_6+5y_7+y_8+6y_9+y_{10}+5y_{11}+2y_{12}.....)]$$

$$
\begin{array}{ccccccccccccc}
 & & & 6n+5 & & & & & & 6n & & & \\
 & & & \cdot & & & & & & \cdot & & & \\
 & & & & \cdot & & & & \cdot & & & & \\
 & & & & 17 & & & & 12 & & & & \\
 & & & & & 11 & & 6 & & & & & \\
 & & & & & & 5 \quad 0 & & & & & & \\
6n+4 & \cdot & 16 & 10 & 4 & & & & 1 & 7 & 13 & \cdot & 6n+1 \\
 & & & & & & 3 \quad 2 & & & & & & \\
 & & & & & 9 & & 8 & & & & & \\
 & & & & 15 & & & & 14 & & & & \\
 & & & & \cdot & & & & \cdot & & & & \\
 & & & \cdot & & & & & & \cdot & & & \\
 & & & 6n+3 & & & & & & 6n+2 & & &
\end{array}
$$

First We will store the interval's functional value in a variable.Then If the sub-interval value $mod(n, 6) = 1$ or $mod(n, 6) = 5$ in other words if the value is from residue class 1 or 5 we will multiply it by 5 and if the sub-interval functional value is from residue class 3 then we will multiply it by 6 or if its from residue class 0 we will multiply it by 2 and the rest values will be added in that variable.

4.2 Integration Using Trapezoidal Rule

1.Evaluate the value of the integral $\int_{0.2}^{1.4} (sinx - lnx + e^x)\, dx$
1.Trapezoidal rule
2.Simpson's 1/3 rule
3.Simpson's 3/8 rule
4/Weddle's rule

x	y=$sinx - lnx + e^x$
x0 = 0.2	y0 = 0.256
x1 = 0.3	y1 = 6.672
x2 = 0.4	y2 = 0.256
x3 = 0.5	y3 = 6.672
x4 = 0.6	y4 = 0.256
x5 = 0.7	y5 = 6.672
x6 = 0.8	y6 = 0.256
x7 = 0.9	y7 = 6.672
x8 = 1.0	y8 = 0.256
x9 = 1.1	y9 = 6.672
x10 = 1.2	y10 = 0.256
x11 = 1.3	y11 = 6.672
x12 = 1.4	y12 = 0.256

4.2.1 Trapezoidal's rule Source Code

```
program trapezoidal
    real i,n,a,b,x
    real h,sm,trapezoidal
    print*,"Enter the value of the upper limit"!!--1.4--!!
    read*,b
    print*,"Enter the value of the lower limit"!!--0.2--!!
    read*,a
    print*,"Enter the number of any interval"
    read*,n!!--12--!!
    open(10,file="trapezoidal.txt")
    sm=0
    h=(b-a)/n
    sm=fun(a)+fun(b)
    write(10,*),"the functional value of point",a,"=",fun(a)
     do i=1,n
        x=a+i*h
        y=fun(x)
        write(10,*)"the functional value of point",x,"=",y
    end do
    do i=1,n-1
        sm=sm+(2*fun(a+(i*h)))
    end do
    trapzoidal=sm*(h/2)
    write(10,*)"The value of the of  integration by trapezoidal is
    ",trapzoidal
    write(10,"(A,f10.5)")"the value with 5 decimal point",
    trapzoidal

end program

real function fun(x)
    fun=sin(x)-log(x)+exp(x)
end function
```

Listing 25: Integration by Trapezoidal's Rule

```
the functional value of point  0.200000003     =   3.02951002
the functional value of point  0.300000012     =   2.84935188
the functional value of point  0.399999976     =   2.79753375
the functional value of point  0.500000000     =   2.82129383
the functional value of point  0.599999964     =   2.89758682
the functional value of point  0.699999988     =   3.01464534
the functional value of point  0.799999952     =   3.16604042
the functional value of point  0.899999917     =   3.34829044
the functional value of point  0.999999940     =   3.55975270
the functional value of point   1.10000002     =   3.80006337
the functional value of point   1.19999993     =   4.06983423
the functional value of point   1.29999995     =   4.37049055
the functional value of point   1.39999998     =   4.70417786
The value of the of  integration by trapezoidal is    4.05617285
the value with 5 decimal point   4.05617
```

Listing 26: Trapezoidal rule Output

4.3 Integration Using Simpson's 1/3 Rule

4.3.1 Simpson's 1/3 rule Source Code

```
program simpson1_3
    integer i,n
    real a,b,x,y,h,sm,simpson
    print*,"Enter the value of the upper limit"
    read*,b!!--1.4--!!
    print*,"Enter the value of the lower limit"
    read*,a!!--0.2--!!
    print*,"Enter the number of any 2's multiple of interval"
    read*,n
    h=(b-a)/n!!--12--!!
    open(10,file="simpson1_3.txt")

    write(10,*)"the functional value of point",a,"=",fun(a)
    do i=1,n
        x=a+i*h
        y=fun(x)
        write(10,*)"the functional value of point",x,"=",y
    end do
    sm=fun(a)+fun(b)
    do i=1,n-1
        if(mod(i,2)==0)then
        sm=sm+(2*fun(a+(i)*h))
        else
        sm=sm+(4*fun(a+((i)*h)))
        endif
    end do

    write(10,*)"the functional value of point",b,"=",fun(b)
    simpson=sm*(h/3)
    write(10,*)"The value of the of  integration by simpsons 1/3
    rule is ",simpson
    write(10,"(A,f10.5)")"the value with 5 decimal point",simpson

end program
```

```
real function fun(x)
    fun=sin(x)-log(x)+exp(x)
end function
```

Listing 27: Simpson's 1/3 rule

```
 the functional value of point  0.200000003     =   3.02951002
 the functional value of point  0.300000012     =   2.84935188
 the functional value of point  0.399999976     =   2.79753375
 the functional value of point  0.500000000     =   2.82129383
 the functional value of point  0.599999964     =   2.89758682
 the functional value of point  0.699999988     =   3.01464534
 the functional value of point  0.799999952     =   3.16604042
 the functional value of point  0.899999917     =   3.34829044
 the functional value of point  0.999999940     =   3.55975270
 the functional value of point   1.10000002     =   3.80006337
 the functional value of point   1.19999993     =   4.06983423
 the functional value of point   1.29999995     =   4.37049055
 the functional value of point   1.39999998     =   4.70417786
 the functional value of point   1.39999998     =   4.70417786
 The value of the of  integration by simpsons 1/3 rule is
    4.05105734
the value with 5 decimal point   4.05106
```

Listing 28: Simpson's 1/3 rule Output

4.4 Integration Using Simpson's 3/8 Rule

4.4.1 Simpson's 3/8 rule Source Code

```
program simpson3_8
    integer i,n
    real a,b,x,y,h,sm,simpson
    print*,"Enter the value of the upper limit"
    read*,b!!--1.4--!
    print*,"Enter the value of the lower limit"
    read*,a!!--0.2--!
    print*,"Enter the number of 3's multiple interval"
    read*,n!!--12--!
    h=(b-a)/n
    open(10,file="simpson3.8.txt")

    write(10,*)"the functional value of point",a,"=",fun(a)
    do i=1,n
        x=a+i*h
        y=fun(x)
        write(10,*)"the functional value of point",x,"=",y
    end do
    sm=fun(a)+fun(b)
    do i=1,n-1
        if (mod(i,3)==0) then
            sm=sm+2*fun(a+(i*h))
        else
            sm=sm+3*fun(a+(i*h))
        end if
```

```
    end do

    write(10,*)"the functional value of point",b,"=",fun(b)
    simpson=sm*(3*h/8)
    write(10,*)"The value of the of  integration by simpsons 3/8 rule is ",simpson
    write(10,"(A,f10.5)")"the value with 5 decimal point",simpson

end program

real function fun(x)
    fun=sin(x)-log(x)+exp(x)
end function
```

Listing 29: Simpson's 3/8 rule

```
 the functional value of point  0.200000003     =   3.02951002
 the functional value of point  0.300000012     =   2.84935188
 the functional value of point  0.399999976     =   2.79753375
 the functional value of point  0.500000000     =   2.82129383
 the functional value of point  0.599999964     =   2.89758682
 the functional value of point  0.699999988     =   3.01464534
 the functional value of point  0.799999952     =   3.16604042
 the functional value of point  0.899999917     =   3.34829044
 the functional value of point  0.999999940     =   3.55975270
 the functional value of point   1.10000002     =   3.80006337
 the functional value of point   1.19999993     =   4.06983423
 the functional value of point   1.29999995     =   4.37049055
 the functional value of point   1.39999998     =   4.70417786
 the functional value of point   1.39999998     =   4.70417786
 The value of the of  integration by simpsons 3/8 rule is    4.05116034
the value with 5 decimal point   4.05116
```

Listing 30: Simpson's 3/8 rule Output

4.5 Integration Using Weddle's Rule

4.5.1 Weddle's rule Source Code

```
program weddles
    integer i,n
    real a,b,x,y,h,sm,weddle
    print*,"Enter the value of the upper limit"
    read*,b!!--1.4--!
    print*,"Enter the value of the lower limit"
    read*,a!!--0.2--!
    print*,"Enter the number 6's multiple of interval"
    read*,n!!--12--!
    h=(b-a)/n
    open(10,file="weddle.txt")

    write(10,*)"the functional value of point",a,"=",fun(a)
    do i=1,n-1
        x=a+i*h
        y=fun(x)
```

```
        write(10,*)"the functional value of point",x,"=",y
    end do
    sm=fun(a)+fun(b)
    do i=1,n-1
        if (mod(i,6)==1 .or. mod(i,6)==5)then
            sm=sm+5*fun(a+(i*h))
        elseif(mod(i,6)==3)then
            sm=sm+6*fun(a+(i*h))
        elseif(mod(i,6)==0)then
            sm=sm+2*fun(a+(i*h))
        else
            sm=sm+fun(a+(i*h))
        end if
    end do
    write(10,*)"the functional value of point",b,"=",fun(b)
    weddle=sm*(3*h/10)
    write(10,*)"The value of the of  integration by weddles  rule is ",weddle
    write(10,"(A,f10.5)")"the value with 5 decimal point",weddle

end program

real function fun(x)
    fun=sin(x)-log(x)+exp(x)
end function
```

Listing 31: Weddle's rule

```
 the functional value of point  0.200000003     =   3.02951002
 the functional value of point  0.300000012     =   2.84935188
 the functional value of point  0.399999976     =   2.79753375
 the functional value of point  0.500000000     =   2.82129383
 the functional value of point  0.599999964     =   2.89758682
 the functional value of point  0.699999988     =   3.01464534
 the functional value of point  0.799999952     =   3.16604042
 the functional value of point  0.899999917     =   3.34829044
 the functional value of point  0.999999940     =   3.55975270
 the functional value of point   1.10000002     =   3.80006337
 the functional value of point   1.19999993     =   4.06983423
 the functional value of point   1.29999995     =   4.37049055
 the functional value of point   1.39999998     =   4.70417786
 The value of the of  integration by weddles  rule is    4.05097532
the value with 5 decimal point   4.05098
```

Listing 32: Weddle's rule Output

5 System of Linear Equation

5.1 Gauss Elimination Method

The data stored in file named elments.txt
The given data is
3
1 -1 1 1

-3 2 -3 -6
2 -5 4 5
Here 3 means number of rows and the equations are
$x - y + z = 1$
$-3x + 2y - 3z = -6$
$2x - 5y + 4z = 5$
Here we need to find the solution of x,y,z.

5.1.1 Source code

```
program gauss_eli
    implicit none
    real A(20,20),ratio,v(20),sm
    integer i,j,n,k
    print *," Gauss elimination without pivot "
    !print *,"No. of row in augmented matrix"
    open(10,file="elements.txt")
    read(10,*)n
    !print *,"enter all elements in a row form"
    do i=1,n
        read(10,*)(A(i,j),j=1,n+1)
    end do

    print *," Given Matrix "
    do i=1,n
        write(*,"(4F10.2)")((A(i,j)),j=1,n+1)
    end do
    do i=1,n
        do j=1,n
            if(j>i)then
                ratio=A(j,i)/A(i,i)
                do k=1,n+2
                A(j,k)=A(j,k)-(ratio*A(i,k))
                end do
            end if
        end do
    end do
    print *," UPPER Triangular Matrix "
    do i=1,n
        write(*,"(4F10.2)")(A(i,j),j=1,n+1)
    end do
    v(n)=A(n,n+1)/A(n,n)
    do i=n-1,1,-1
        sm=0.
        do j=i+1,n
            sm=sm+A(i,j)*v(j)
        end do
        v(i)=(A(i,n+1)-sm)/a(i,i)
    end do

    print *," SOLUTIONS ARE "
    do i=1,n
        write(*,"(F10.2)")v(i)
    end do

```

```
end program
```

Listing 33: Gauss Elimination Method

```
  Gauss elimination without pivot
  Given Matrix
      1.00      -1.00       1.00       1.00
     -3.00       2.00      -3.00      -6.00
      2.00      -5.00       4.00       5.00
  UPPER Triangular Matrix
      1.00      -1.00       1.00       1.00
      0.00      -1.00       0.00      -3.00
      0.00       0.00       2.00      12.00
  SOLUTIONS ARE
     -2.00
      3.00
      6.00

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

Listing 34: Gauss Elimination Method Output

5.2 Gauss Jordan Method

The data stored in file named elments.txt
The given data is
3
1 -1 1 1
-3 2 -3 -6
2 -5 4 5
Here 3 means number of rows and the equations are
$x - y + z = 1$
$-3x + 2y - 3z = -6$
$2x - 5y + 4z = 5$
Here we need to find the solution of x,y,z.

5.2.1 Source code

```
program gauss_jordan
    implicit none

    real A(20,20),ratio
    integer i,j,n,k

    print *," Gauss Jordan Method "
    !print *,"No. of row in augmented matrix"
    open(10,file="elements.txt")
    read(10,*)n
    !print *,"enter all elements in a row form"
    do i=1,n
        read(10,*)(A(i,j),j=1,n+1)
```

```
    end do

    print *," Given Matrix "
    do i=1,n
        write(*,"(4F10.2)")((A(i,j)),j=1,n+1)
    end do

    do i=1,n
        do j=1,n
            if(i.ne.j)then
                ratio=A(j,i)/A(i,i)
                do k=1,n+2
                A(j,k)=A(j,k)-(ratio*A(i,k))
                end do
            end if
        end do
    end do

    print *," Diagonal Matrix "
    do i=1,n
        write(*,"(4F10.2)")(A(i,j),j=1,n+1)
    end do
    print *," SOLUTIONS ARE "
    do i=1,n
        print"(F10.2)",A(i,n+1)/A(i,i)
    end do

end program
```

Listing 35: Gauss-Jordan Elimination Method

```
  Gauss Jordan Method
  Given Matrix
      1.00     -1.00      1.00      1.00
     -3.00      2.00     -3.00     -6.00
      2.00     -5.00      4.00      5.00
  Diagonal Matrix
      1.00      0.00      0.00     -2.00
      0.00     -1.00      0.00     -3.00
      0.00      0.00      2.00     12.00
  SOLUTIONS ARE
     -2.00
      3.00
      6.00

Process returned 0 (0x0)   execution time : 0.073 s
Press any key to continue.
```

Listing 36: Gauss Jordan Method Output

5.3 Gauss Seidel Method

The data stored in file named elments.txt
The given data is
3

0.0001
10
27 6 -1 85
6 15 2 72
1 1 54 110
Here 3 means number of rows and the equations are
$27x + 6y - z = 85$
$6x + 15y + 2z = 72$
$x + y + 54z = 110$
Here we need to find the solution of x,y,z.

5.3.1 Source code

```
program gauss_seidel
    implicit none
    real A(20,20),ratio,v(20),sm,tol,maxit,tmp,er
    integer i,j,n,k,s
    print *," Gauss Seidel Method "
    !print *,"No. of row in augmented matrix"
    open(10,file="elements.txt")
    read(10,*)n
    read(10,*)tol
    read(10,*)maxit
    !print *,"enter all elements in a row form"
    do i=1,n
        read(10,*)(A(i,j),j=1,n+1)
    end do

    print *," Given Matrix "
    do i=1,n
        write(*,"(10F10.2)")((A(i,j)),j=1,n+1)
    end do
    do i=1,n
        s=0
        do j=1,n
            if (i.ne.j) then
                s=s+abs(A(i,j))
            end if
        end do
        if(abs(a(i,i)).le.s)then
            print*,"the Gauss Seidel method is not applicable."
            stop
        end if
    end do
    v=0
    print*,"Initially All the values of variable are stored as 0"
    print*,((v(i)),i=1,n)

    do i=1,maxit
        do j=1,n
            sm=0
            do k=1,n
                if(k.ne.j)then
                    sm=sm+A(j,k)*v(k)
```

```
                end if
            end do
            tmp=(A(j,n+1)-sm)/A(j,j)
            er=(abs(v(j)-tmp)/tmp)
            if(er.gt.tol)then
                v(j)=tmp
                print"(A,I2,2x,A,F10.5)","v",(j),"after iteration = ",tmp
            end if

        end do

    end do
     print *, " Solution are "
      do i = 1, n
        print "(4F10.3)", v(i)
      end do
end program
```

Listing 37: Gauss-Seidel Method

```
  Gauss Seidel Method
  Given Matrix
     27.00      6.00     -1.00     85.00
      6.00     15.00      2.00     72.00
      1.00      1.00     54.00    110.00
 Initially All the values of variable are stored as 0
   0.00000000        0.00000000        0.00000000
v 1  after iteration =   3.14815
v 2  after iteration =   3.54074
v 3  after iteration =   1.91317
v 1  after iteration =   2.43217
v 2  after iteration =   3.57204
v 3  after iteration =   1.92585
v 1  after iteration =   2.42569
v 2  after iteration =   3.57294
  Solution are
     2.426
     3.573
     1.926

Process returned 0 (0x0)   execution time : 0.037 s
Press any key to continue.
```

Listing 38: Gauss Seidel Method Output

Conclusion

Hopefully, it will be useful to everyone. Some of the most important Numerical analysis topics are covered.