Michael Timbes
Date: 09/12/2016
Class: Numerical Analysis

1 Textbook Problem 10

1.1 Problem

This problem asks to use bisection to estimate the value of $\sqrt{25}$ up until a precision of 10^{-10}.

1.2 Solution

To do this I begin with the equation $y(x) = x^3 - 25$ because the solution to this equation is what we are asked to approximate. My initial interval will be $[-1, 4]$ the approximation method converges slowly so I chose something close to the solution. The output is as follows.

```
Program BiSolution
Implicit None
DOUBLE PRECISION :: A, B, M, F,ERROR,TOL
Integer :: I=0

        A=-1 !INITIAL GUESSES AND INITIALIZING INTERVAL [A,B]
        B=4
        TOL = 1.0E-10
        PRINT*,"INITIAL [A,B]","[",A,",",B,"]"
                DO !FOR 14 ITERATIONS
                M= A + (B-A)/2.0
                IF( F(M) .LT.0.0) THEN !IF FUNCTION VAL IS NEGATIVE
                        A = M !ASSIGN MID POINT TO A
                ELSE
                        B = M !ASSIGN MID POINT TO B IF F(M) > 0
                END IF
I = I +1
ERROR = ABS(B-A)
PRINT*,"A:",A,"B:",B,"M:",M
                IF(ERROR.LT.TOL) THEN
                        PRINT*,"ERROR",ERROR
                        PRINT*,"Iterations",I
                        STOP;
                        END IF
END DO

PRINT*,"ERROR:",ERROR
```

```
End Program BiSolution

DOUBLE PRECISION FUNCTION F(G)
DOUBLE PRECISION G
F = G**3 - 25
RETURN
END
```

1.3 Output

For the sake of being brief I will only include the last few lines of output.

```
A:   2.9240177381725516      B:   2.9240177383180708      M:   2.9240177381725516
 A:   2.9240177381725516      B:   2.9240177382453112      M:   2.9240177382453112
 ERROR   7.2759576141834259E-011
 Iterations          36
```

2 Textbook Problem 17

2.1 Problem

This problem asks to solve for a constant negative value that describes the rate at which an angle decreases as an object moves down a slope. The given values for time and displacement used in place of the corresponding variables and then the equation is set to zero.

2.2 Solution

In the Fortran code, the resulting function is what is used as the main function at the end of the program.

```
Program Secantp2
Implicit None
Double Precision :: A, B, P, Pold, F, TOL
Integer :: I
!I =0 !Initialize Iterator
A =1!Set to initial P to test tolerance
P=-1 !Left interval
Pold = 1 !Outer interval expressed as Pold
Tol = 1.0E-8
DO! I = 1,20
        B=P !B holds current P to store in Pold later
        P=P - (F(P) *((P-Pold))/(F(P)-F(Pold))) !Redefine P as outlined in th
I = I + 1
```

```
        PRINT*,"Iteration:",I,"P:",P,"Pold:",Pold
        Pold=B !New Pold

If(abs(Pold-P) .LT. TOL) Then !If current P and Pold are very close- converge
Print*,"Convergence is assumed:"
Print*,"Solution: ",Pold
Print*,"Iterations Needed: ",I
Stop
END If
END DO
End Program Secantp2

Double Precision Function F(G)
Double Precision :: G, Y

Y = ((-32.17)/(2.0 * G**2)) * (sinh(G) - sin(G))-1.7
Return
End
```

The resulting output is less than the tolerance value (TOL in the code above).

```
mtimbes@mtimbes-VirtualBox:~$ ./a.out
 Iteration:           1 P: -0.31668855820625530      Pold:   1.0000000000000000
 Iteration:           2 P: -0.31706117667314709      Pold:  -1.0000000000000000
 Iteration:           3 P: -0.31706180146977464      Pold: -0.31668855820625530
 Iteration:           4 P: -0.31706180146968421      Pold: -0.31706117667314709
 Convergence is assumed:
 Solution:  -0.31706180146977464
 Iterations Needed:           4
```

3 Tangent Function Solutions

3.1 Problem

This problem proved to be more difficult than the previous since there is an issue regarding the asymptotic behavior of the tangent function. First, I tried to solve this problem using the secant method but I was only able to find one solution.

3.2 Solution

I then attempted to try to solve this by Newton's method but quickly realized that Newton's method does not do well around points where the derivative of the function is very small at a given point. I by chance stumbled on a version of Newton's Method in the textbook (pg. 49- q.8) but for functions of multiple zeros. There was still the issue of what to do around a point where the function approached an asymtote, I tried a simple idea of back tracking the point to half of its location and then using that new point as the next guess. This idea of cutting the point in half was inspired by the bisection method. The new value of the point would then be calculated and if need be, would also be cut in half until the derivative at the point is greater than the tolerance set (or greater than zero). Then from there the loop continues until the absolute value of the function at the point minus the function value at the previous point are less than the tolerance value. With two function values being almost equal, it seemed to me that the test would be grounds enough to assume convergence to a solution near the original point. For some odd reason, I couldn't get a nested do loop to work so the output shows several solutions near various points created from the program.

```
Program TanNewton
Implicit None
Double Precision :: F,J,T,P,B,N,TOL
Integer :: I
TOL = 1.0E-8
I = 0
P= 7
                        Do
B=P

            P = B - (F(B)*J(B))/((J(B)**2)-(F(B)*T(B))) !Numerical Method Based o
!Print*,"P:",P Optional Print Command
I = I+1 !Iteration
!Check For Asymtote
            If(abs(J(P)) < TOL) Then !Test For Asymtotic Behavior
            P = P/2 !Take the current point back half way- then continue with the
            !Print*,"Approaching Asymptote."
            End IF
If(abs(F(P)-F(B))<TOL) Then !Test For Solution- If function value at current
vious point within tolerance level- Convergence is Assumed
Print*,"Solution:",F(P)
Print*,"P:",P
Print*,"I:",I
Stop;
End IF




                        END DO
End Program TanNewton

!Original Function
Double Precision Function F(X)
Double Precision X,Y
Y = X-Tan(X)
Return
End
!First Derivative
Double Precision Function J(X)
Double Precision X,Y
Y = -(2/(cos(2*X) +1))
Return
End

!Second Derivative
Double Precision Function T(X)
Double Precision X,Y
Y = Tan(x) * (-1*(2/(cos(2*X) +1)))
Return
End
```

3.3 Output

The output with various points. The drawback to this method is that I have to put the starting points relatively near the actual solution point which means that I would have to have an idea of about where the solution point should be.

```
*test with P=7*
mtimbes@mtimbes-VirtualBox:~$ ./a.out
 Function Value:   2.8848035071860068E-012
 P:   7.7252518369376588
 I:           7
*test with P= -3.5*
mtimbes@mtimbes-VirtualBox:~$ ./a.out
 Function Value:   3.5561154021479524E-010
 P:  -4.4934094579266768
 I:           8
mtimbes@mtimbes-VirtualBox:~$
*test with P = 13*
mtimbes@mtimbes-VirtualBox:~$ ./a.out
 Function Value:   3.0359714742189681E-011
 P:   14.066193912831320
 I:           5
mtimbes@mtimbes-VirtualBox:~$
```

4 Summation of Expansion

4.1 Problem

The problem asks to write a function in FORTRAN to numerically estimate the sum of a series expansion at some value 'x' for 'I' iterations.

4.2 Solution

The sum builds from its original starting point of negative one. In this program, the user may input what 'x' value and 'I' value desired. Using the input, the function is repeatedly called for however many times the index is set for index is initally one so the function is still defined upon the first call. The program's output was compared to Wolfram Alpha's values given the same input values and the output was found to be very close

```
Program Summation
IMPLICIT NONE
INTEGER :: I

DOUBLE PRECISION :: x,func,sumval
print*,"Enter the x value."
READ(*,*) x
print*,"Enter the I value:"
READ(*,*) I
sumval = -1.0
!Begin Do Until the I value is reached
        do I = 1,I
        sumval = sumval + FUNC(x,I) !Sumvalue updated
        end do
print*,"Sum:",sumval
End Program Summation

!Separate Function created for clarity
DOUBLE PRECISION FUNCTION FUNC(G,Q)
DOUBLE PRECISION :: G, Y
INTEGER :: Q

Y=( cos(G *dble(Q))/dble(Q) )

RETURN
END
```

4.3 Output

```
mtimbes@mtimbes-VirtualBox:~$ ./a.out
```

```
 Enter the x value.
10
 Enter the I value:
50
 Sum:   -1.6585629797099213
```

5 Solution of Expansion

5.1 Problem

This problem asks to find the solutions to the series expansion of a function.

$$f(x) = -1 + \sum_{i=1}^{\infty} \frac{\cos(ix)}{i} \tag{1}$$

The solutions are not going to be exact, but only approximations.

5.2 Solution

The equation being a trignometric function has more than one root. I decided to use the same multiple root method used in the third problem. The program is similar except that I decided to step through the 'P' values and used ten million iterations of the variation of Newton's Method on each step. Although this is incredibly slow it will converge around each solution within an interval. The solution of the equation is where the sum is equal to zero at a given x (or p-value). This produced at least two solutions.

```
Program NewtonSummation
Implicit None
Double Precision :: FUNC,F,J,T,P,B,N,TOL,SUMM
Integer :: I,A
Real :: PI = 4.*atan(1.0)
TOL = 1.0E-2
N= 0.25
Do 10 A = 1,50 ! Max interval will be i=0,i=50 of sum ((N_-1) +1/5)
N = N + 1.0/5.0 !Step in P Guess
P = N
Print*,"PVAL:",P
SUMM=-1

                    Do 20 I=10000000,1,-1
B=P
SUMM = SUMM + F(P,I)
          P = B - (F(B,I)*J(B,I))/((J(B,I)**2)-(F(B,I)*T(B,I))) !Numerical Meth
!Print*,"P:",P! Optional Print Command
!Print*,"I:",I
!Print*,"Sum:",SUMM
```

```
!Print*,"F(x)",F(P,I),"Sum:",SUMM
!Check For Asymtote
          If(FUNC(P)==0) Then !Test For Asymtotic Behavior

          P = P/4
Print*,"Approaching␣Asymtote."

          End IF
If(abs(SUMM).LT.TOL) Then !Test For Solution- If function value at current pc

Print*,"Function␣Value:",F(P,I)
Print*,"P:",P
Print*,"I:",I
!Stop;
End IF



                    20 END DO
!Print*,"Sum:",SUMM
10 END DO
End Program NewtonSummation

!Original Function
Double Precision Function FUNC(X)
Double Precision X,Y
Y = cos(X)
Return
End

!MOD ORI FUNCT
Double Precision Function F(X,I)
Double Precision X,Y
Integer :: I
Y = (cos(dble(I)*x)/dble(I))
Return
End
!First Derivative
Double Precision Function J(X,I)
Double Precision X,Y
Integer :: I
Y = -(dble(I)*sin(dble(I)*x))
Return
End

!Second Derivative
```

```
Double Precision Function T(X, I)
Double Precision X,Y
Integer :: I
Y = dble(I)*cos((PI/2.0)+dble(I)*x)
Return
End
```

5.3 Output

There is a lot of output numbers so I will place the solutions found after one run of the program. I had to bring the tolerance level down to $1.0 * 10^{-2}$ in order to get a close approximation. The approximated solutions are near the 'PVAL' variable.

```
 PVAL:    1.2500000149011612
  Function Value:    5.9493503315847631E-003
 P:    3.8697353286847513E-003
  I:          143
  Function Value:    5.9628326761942130E-003
 P:    3.9507209199390256E-003
  I:          142
  Function Value:    5.9768950214381692E-003
 P:    4.0314838068232876E-003
  I:          141
 PVAL:    1.4500000178813934
 ..............................
PVAL:    6.6500000953674316
  Function Value:    5.4136515712971338E-002
 P:    6.7436072526465871
  I:           15
 PVAL:    6.8500000983476639
```