01.

Lambert Function

(-1/e;0) as a solution of the equation:

$$W_{-1}\exp\left(W_{-1}\right)=z.$$

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

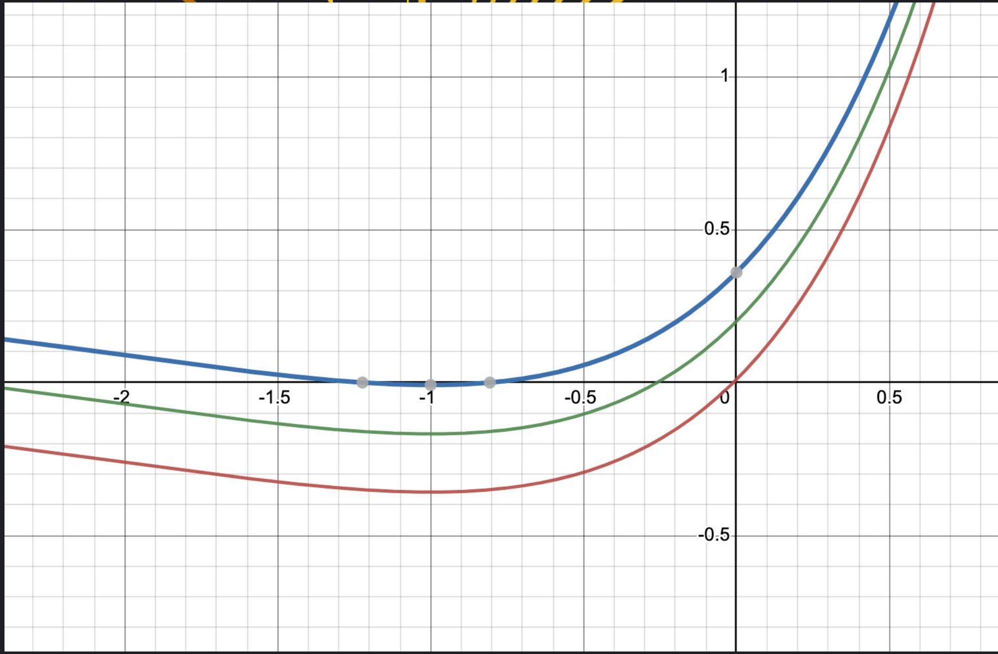
The Mathematical Form

f(w)=wexp(w)-z

The root on the right gives W0.

Therefore, the boundary between -1 to 2

The root on the left gives W1. Therefore, the boundary between -10 to -1.



003-1040559 1250 003-771

CODE-BISECTION

```fortran
implicit none
double precision, parameter :: tol=1.0d-6 ! maximum relative error
double precision, parameter :: x_left=-10.0,x_right=-1.0 ! initial interval to check for intersection
! function must have different signs on the left and right
double precision :: f,x_left_n,x_right_n,x_middle_n,z
double precision :: x_iteration_old,x_iteration,relative_error
double precision :: tmp,tmp1,tmp2
integer, parameter :: Nz=1000 ! creating a thoundsand points
double precision, parameter :: z1=-0.3678794,z2=-.01 ! going from -e^(-1) to -.01
double precision :: W1(Nz) ! create the W0 array of 1000 points
integer n,k,iterations(Nz) ! See each iteration

do k=1,Nz ! going through a thoundsand values
  z=z1+(z2-z1)*float(k-1)/float(Nz-1) ! create each value of z

  ! find the value of f=W0*exp(W0)-z where W0=x_left/x_right
  call lambert(z,x_left,tmp1)
  call lambert(z,x_right,tmp2)

  ! check if interval is correct
  tmp=tmp1*tmp2
  if(tmp>0.0)then
    write(*,*) 'function must have different signs in x_left and x_right'
    write(*,*) 'exit with error'
    stop
  endif

  n=0 !zeroth iteration
  x_left_n=x_left     !initial point on the left
  x_right_n=x_right   !initial point on the right
  x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
  relative_error=1.0 !set inital error to any value higher than tol
  x_iteration_old=x_middle_n !zeroth iteration
```

```fortran
  do while (relative_error>tol) !iterations untill relative error is less than tol
    n=n+1 ! update the iteration values

    !calculate the value f(w) again on both ends
    call lambert(z,x_left_n,tmp1)
    call lambert(z,x_middle_n,tmp2)

    !Update the boundary
    if(tmp1*tmp2<=0.0)then
      x_right_n=x_middle_n !<--- the root is in left subinterval
    else
      x_left_n=x_middle_n  !<--- the root is in right subinterval
    endif

    !update data
    x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
    x_iteration=x_middle_n
    relative_error=abs((x_iteration-x_iteration_old)/x_iteration) !find the differences btw two values
    x_iteration_old=x_iteration
  enddo

  !update the value of each z for W1 and the iteration array
  W1(k)=x_iteration
  iterations(k)=n
enddo

open(file='/Users/phihung/Documents/PHY/first/homework/hw_02/lambert_function_W1.dat',unit=32) !file to record Lambert function W0
do k=1,Nz
  z=z1+(z2-z1)*float(k-1)/float(Nz-1)
  write(32,*) z,W1(k),iterations(k)
enddo
close(unit=32)

end
```

```fortran
subroutine lambert(z,W1,f)
  implicit none
  double precision z,W1,f
  f=W1*exp(W1)-z
end subroutine lambert
```

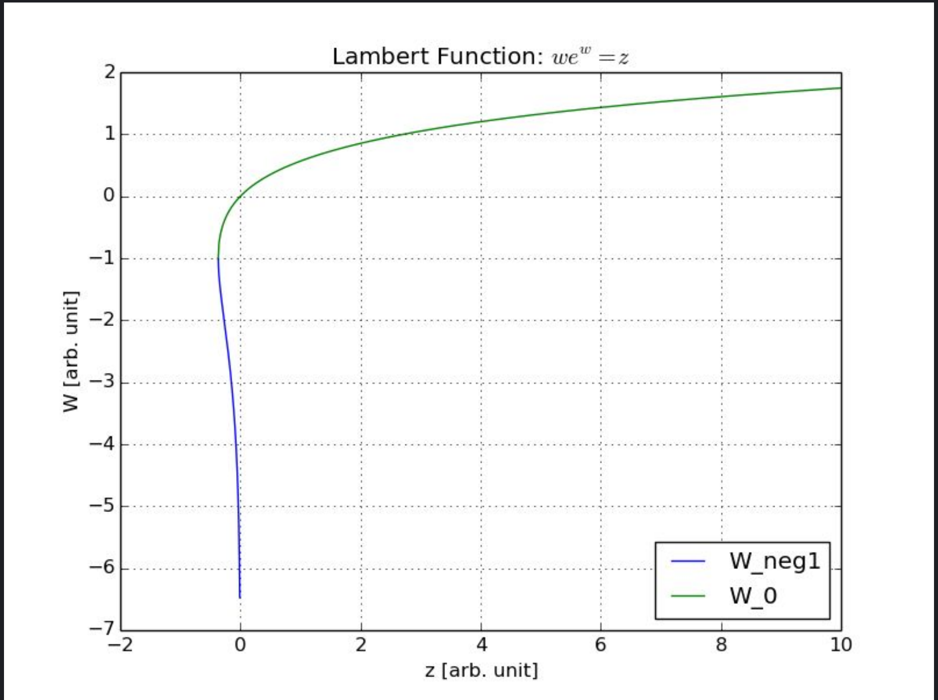
For each z value, we find a root on the left.

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Graph Bisection

I was able to generate Lambert Functions using the right boundary. Most converge with 20-23 iterations.

−0.36787939071655273	−1.0005241036415100	23
−0.3675211530879841	−1.0447977185249329	23
−0.36716291545941548	−1.0637502074241638	23
−0.36680467783084680	−1.0784589052200317	22
−0.36644644020227818	−1.0909708738327026	22
−0.36608820257370955	−1.1020838022232056	22
−0.36572996494514093	−1.1122010946273804	22
−0.36537172731657225	−1.1215652227401733	22
−0.36501348968800362	−1.1303349733352661	22
−0.36465525205943500	−1.1386197805404663	22
−0.36429701443086637	−1.1464968919754028	22
−0.36393877680229769	−1.1540285348892212	22
−0.36358053917372907	−1.1612597703933716	22
−0.36322230154516044	−1.1682292222976685	22
−0.36286406391659182	−1.1749669313430786	22
−0.36250582628802314	−1.1814965009689331	22
−0.36214758865945451	−1.1878393888473511	22
−0.36178935103088589	−1.1940127611160278	22
−0.36143111340231726	−1.2000316381454468	22
−0.36107287577374858	−1.2059088945388794	22
−0.36071463814517996	−1.2116574048995972	22
−0.36035640051661133	−1.2172836065292358	22
−0.35999816288804271	−1.2227982282638550	22
−0.35963992525947402	−1.2282098531723022	22
−0.35928168763090540	−1.2335227727890015	22
−0.35892345000233677	−1.2387455701828003	22
−0.35856521237376815	−1.2438803911209106	22
−0.35820697474519947	−1.2489379644393921	22
−0.35784873711663084	−1.2539182901382446	22
−0.35749049948806222	−1.2588256597518921	22
−0.35713226185949359	−1.2636665105819702	22



003-1040559   1250 003-77156.8   1760 0009-14563.7   73273

CODE-NR

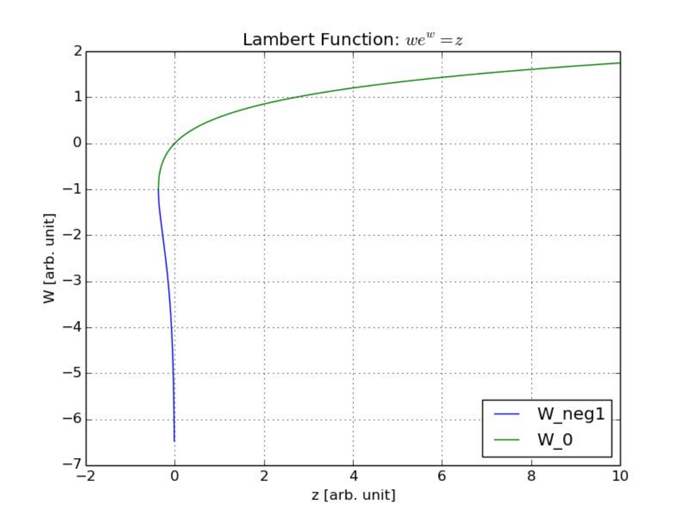
```
implicit none
double precision, parameter :: tol=1.0d-6 !maximum relative error
double precision, parameter :: x_initial=-2.51 !initial interval !-3->-2
!function must have different signs on the left and right
double precision f,x_left_n,x_right_n,x_middle_n,z,dfdx1
double precision x_iteration_old,x_iteration,relative_error
double precision tmp,tmp1,tmp2
integer, parameter :: Nz=10000
double precision, parameter :: z1=-0.3678794,z2=-.02
double precision W1(Nz)
integer n,k,iterations(Nz)

do k=1,Nz
  z=z1+(z2-z1)*float(k-1)/float(Nz-1) ! find z value

  n=0 !zeroth iteration
  call lambert(z,x_initial,tmp1,dfdx1) ! fidn the value of function and the derivati
  relative_error=1.0 !set inital error to any value higher than tol
  x_iteration_old=x_initial-tmp1/dfdx1 !zeroth iteration
  do while (relative_error>tol) !iterations untill relative error is less than tol
    n=n+1
    call lambert(z,x_iteration_old,tmp1,dfdx1)
    x_iteration=x_iteration_old-tmp1/dfdx1
    relative_error=abs((x_iteration-x_iteration_old)/x_iteration)
    x_iteration_old=x_iteration
  enddo
  W1(k)=x_iteration
  iterations(k)=n
enddo

open(file='/Users/phihung/Documents/PHY/first/homework/hw_02/lambert_function_W1_NR.dat',unit=32) !file to record Lambert function W
do k=1,Nz
  z=z1+(z2-z1)*float(k-1)/float(Nz-1)
  write(32,*) z,W1(k),iterations(k)
enddo
close(unit=32)

end
```

```

subroutine lambert(z,W1,f,df_dx)
  implicit none
  double precision z,W1,f, df_dx
  f=W1*exp(W1)-z
  df_dx=W1*exp(W1)+exp(W1)
end subroutine lambert
```

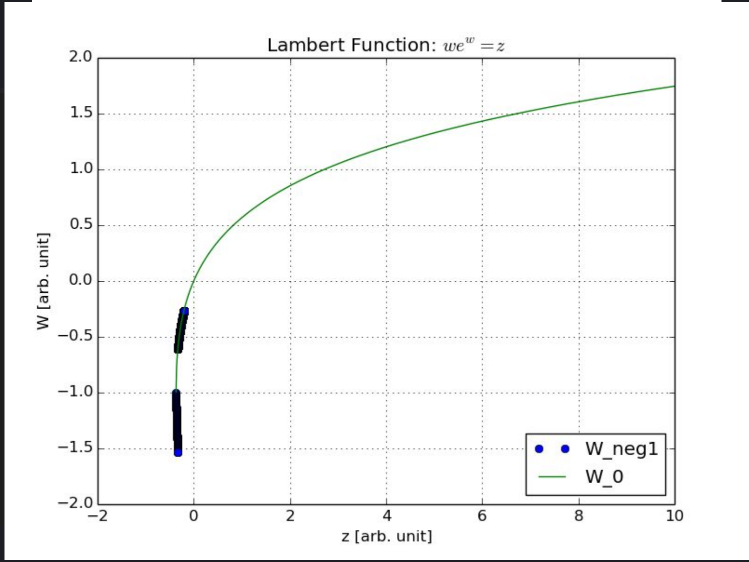
Graph N-R

003-1040559 1250 003-7715

I was able to generate Lambert Functions with the initial between 2-3. Most converges within 10 iterations, however some take up to 100 iterations (close to 0/steep).



0.36787939071655273	-1.0005238291947034
0.36784359919830689	-1.0140244959180484
0.36780780768006105	-1.0198652348225252
0.36777201616181526	-1.0243634239254609
0.36773622464356942	-1.0281664117315397
0.36770043312532358	-1.0315250827517217
0.36766464160707774	-1.0345681365122341
0.36762885008883189	-1.0373720363750698
0.36759305857058611	-1.0399866062114982
0.36755726705234026	-1.0424464554139938
0.36752147553409442	-1.0447767835711188
0.36748568401584858	-1.0469966146280034
0.36744989249760274	-1.0491207283943924
0.36741410097935695	-1.0511608784787960
0.36737830946111111	-1.0531265946173074
0.36734251794286527	-1.0550257306129383
0.36730672642461942	-1.0568648500270850
0.36727093490637358	-1.0586495047420019
0.36723514338812774	-1.0603844406706975
0.36719935186988195	-1.0620737525807802
0.36716356035163611	-1.0637210027458586
0.36712776883339027	-1.0653293130739105
0.36709197731514442	-1.0669014377542951
0.36705618579689858	-1.0684398211743109
0.36702039427865280	-1.0699466446104868
0.36698460276040695	-1.0714238642443821
0.36694881124216111	-1.0728732423952316

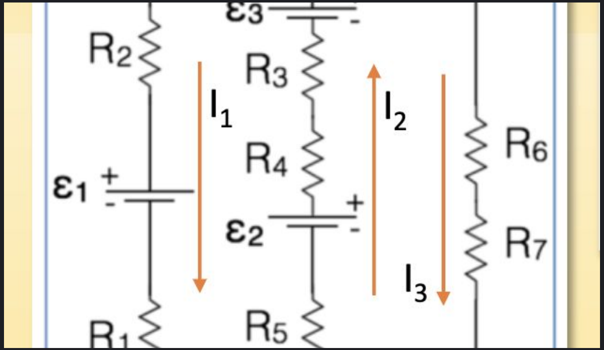


003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

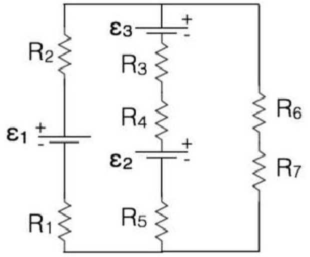
Conclusion

Using bisection is much easier than NR to generate data for W1, but it is much faster to generate W0 with NR.

02.



LU Decomposition



Find all three currents (in **A**) in the circuit shown in the figure if:

$\varepsilon_1 = \mathbf{2\ V}$, $\varepsilon_2 = \mathbf{3\ V}$, $\varepsilon_3 = \mathbf{4\ V}$,

$R_1 = \mathbf{1\ \Omega}$, $R_2 = \mathbf{2\ \Omega}$,

$R_3 = \mathbf{3\ \Omega}$, $R_4 = \mathbf{1\ \Omega}$, $R_5 = \mathbf{2\ \Omega}$,

$R_6 = \mathbf{3\ \Omega}$, $R_7 = \mathbf{1\ \Omega}$.

CODE

003-1040559 125

Ludcmp and lubksb are given methods

```
implicit none
integer, parameter :: n_size=3
double precision current(n_size),A(n_size,n_size),B(n_size),tmp
integer indx(n_size)

!define A and B
A(1,1)= 1.0 !first row
A(1,2)=-1.0
A(1,3)= 1.0

A(2,1)=-3.0 ! second row
A(2,2)=-6.0
A(2,3)= 0.0

A(3,1)= 0.0 ! third row
A(3,2)=-6.0
A(3,3)=-4.0
! Ax = B
B(1)= 0.0
B(2)= -5.0
B(3)= -7.0

call ludcmp(A,n_size,n_size,indx,tmp) !this call performs LU decomposition of A
                                      !note that A is replaced by LU decomposition
call lubksb(A,n_size,n_size,indx,B)   !B is replaced by the solution of A*X=B

write(*,*) 'current I1=',B(1)
write(*,*) 'current I2=',B(2)
write(*,*) 'current I3=',B(3)

end
```

003-1040559 1250 003-77156.8 1760 0009-14563.7 73273

Conclusion

I was able to solve linear equations using LU decompositions. Check: I1+I3=I2

```
current I1=  0.14814814814814814
current I2=  0.75925925925925930
current I3=  0.61111111111111116
```

03

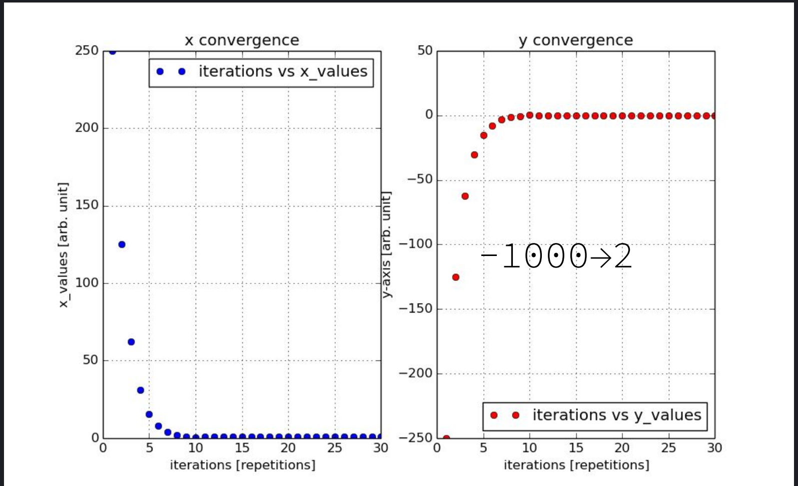
Transcendental Equation

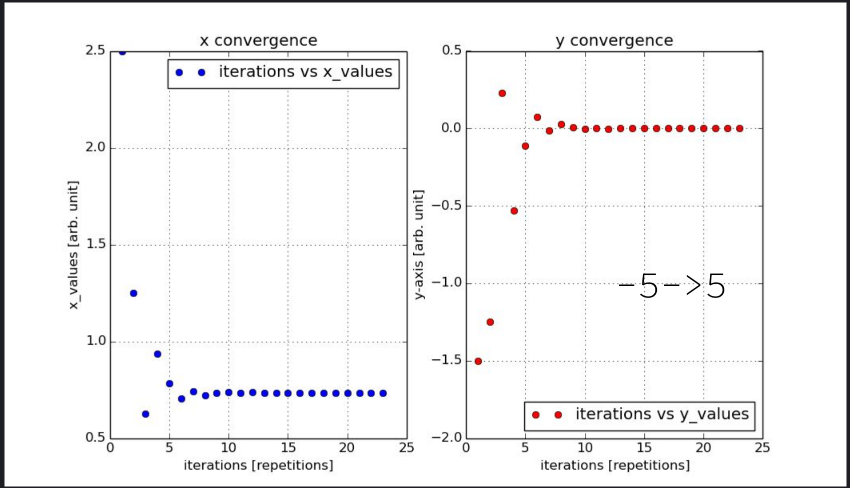
$$\sin^2\left(x^2 - \frac{\pi}{2}\right) - x = 0$$

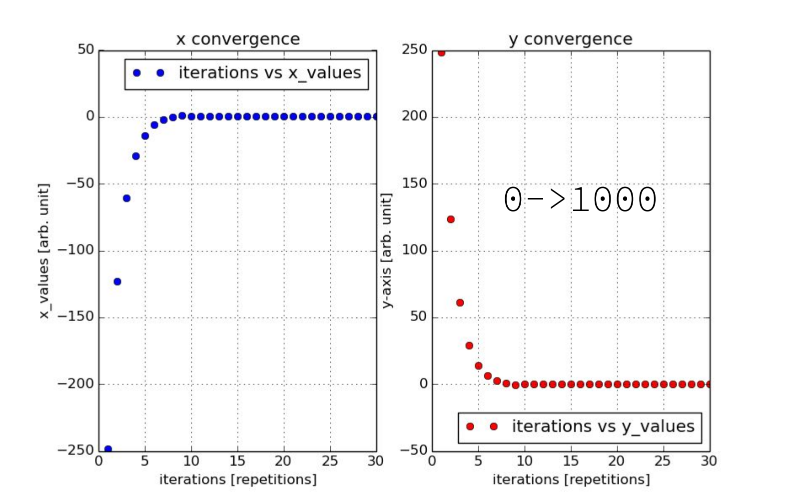
Bisection Method


```fortran
implicit none
double precision, parameter :: tol=1.0d-6 !maximum relative error
double precision, parameter :: x_left=-0.0,x_right=1000 !initial interval
                                !function must have different signs on the left and right
double precision f,x_left_n,x_right_n,x_middle_n
double precision x_iteration_old,x_iteration,relative_error
double precision tmp
integer n

tmp=f(x_left)*f(x_right) !check if interval is correct
if(tmp>0.0)then
  write(*,*) 'function must have different signs in x_left and x_right'
  write(*,*) 'exit with error'
  stop
endif


n=0 !zeroth iteration
x_left_n=x_left     !initial point on the left
x_right_n=x_right   !initial point on the right
x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
relative_error=1.0 !set inital error to any value higher than tol
x_iteration_old=x_middle_n !zeroth iteration
```

```fortran
open(file='/Users/phihung/Documents/PHY/first/homework/hw_02/bisectional_-0_1000.dat',unit=11) !file to re

  do while (relative_error>tol) !iterations untill relative error is less than tol
    n=n+1
    if(f(x_left_n)*f(x_middle_n)<=0.0)then
      x_right_n=x_middle_n !<--- the root is in left subinterval
    else
      x_left_n=x_middle_n  !<--- the root is in right subinterval
    endif
    x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
    x_iteration=x_middle_n
    relative_error=abs((x_iteration-x_iteration_old)/x_iteration)
    x_iteration_old=x_iteration
    write(11,*) n,x_iteration,f(x_iteration)
  enddo
close(unit=11)

end

double precision function f(x)
  implicit none
  double precision pi
  double precision x
  pi = 4.0*atan(1.0)
  f=(sin(x**2-pi/2))**2-x
end function f
```





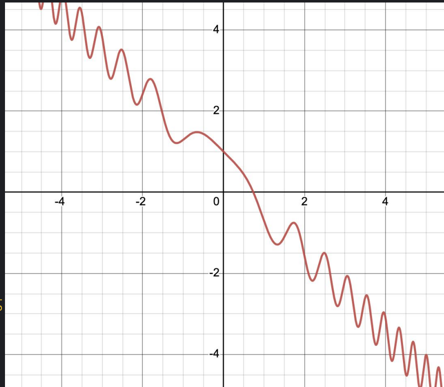


Different intervals to show the convergence of data. The bigger the range, the more steps to take to come to convergence. The method

Converges for all proper

intervals

Convergence around .75

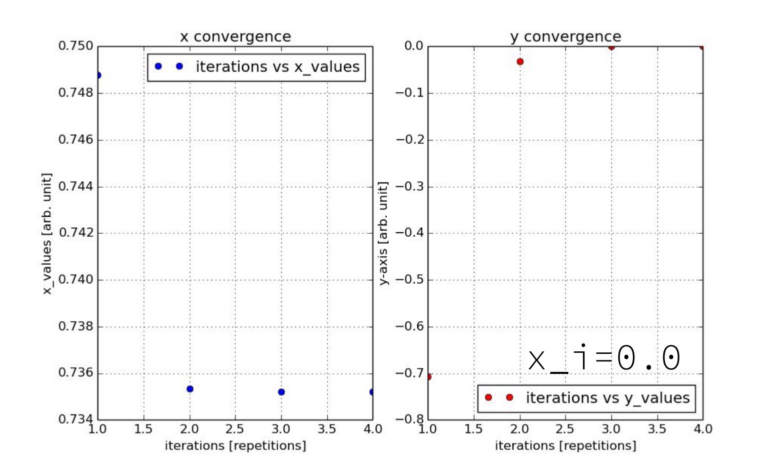


Newton-Raphson

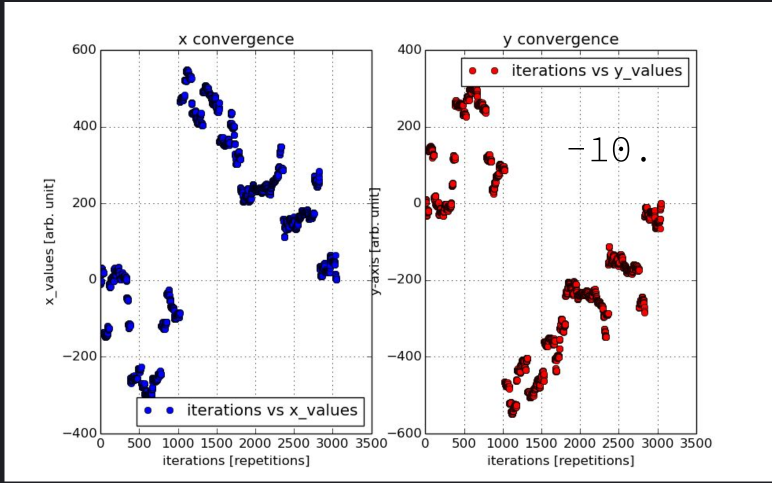
```fortran
implicit none
double precision, parameter :: tol=1.0d-6 !maximum relative error
double precision, parameter :: x_initial=-1.8 !initial guess
double precision f,df_dx
double precision x_iteration_old,x_iteration,relative_error
double precision tmp
integer n

n=0 !zeroth iteration
call f_and_deriv(x_initial,f,df_dx)
x_iteration_old=x_initial-f/df_dx
relative_error=1.0 !set inital error to any value higher than tol

!newton method
open(file='/Users/phihung/Documents/PHY/first/homework/hw_02/Newton_Raphson.dat',unit=11) !file to
  do while (relative_error>tol) !iterations untill relative error is less than tol
    n=n+1
    call f_and_deriv(x_iteration_old,f,df_dx)
    x_iteration=x_iteration_old-f/df_dx
    relative_error=abs((x_iteration-x_iteration_old)/x_iteration)
    x_iteration_old=x_iteration
    write(11,*) n,x_iteration,f
  enddo
close(unit=11)
end

!define function
subroutine f_and_deriv(x,f,df_dx)
  implicit none
  double precision x,f,df_dx, pi
  pi = 4.0*atan(1.0) !define pi
  f=(sin(x**2-pi/2))**2-x !function
  df_dx=4*x*sin(x**2-pi/2)*cos(x**2-pi/2)-1 !the derivative
end subroutine f_and_deriv
```

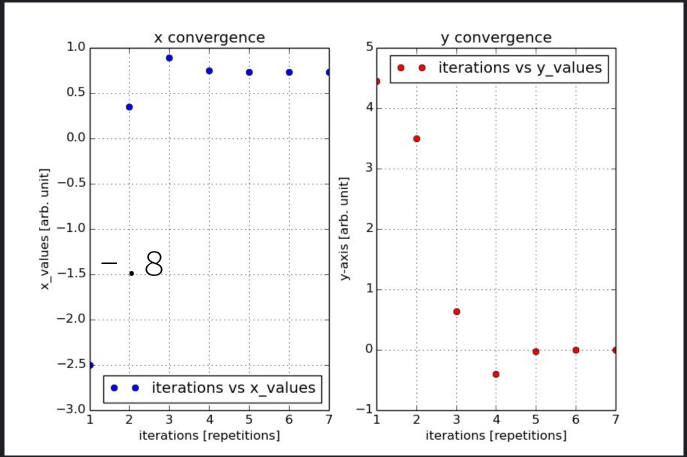
01
02
03
04
05
06

X_i = -1.8 doesn't converge. Break at 100000.



387993	-357527.82388274831	357527.99804926023
387994	-357527.10490780167	357527.96458747040
387995	-357528.09404438152	357527.17349240789
387996	-357526.71509536542	357529.06001787313
387997	-357526.20186016290	357527.10227776656
387998	-357527.47883265634	357527.16193782154
387999	-357528.91355904349	357528.44748688914
388000	-357530.25800018740	357528.94942327990
388001	-357530.76173138537	357530.81873220118
388002	-357524.01690509508	357531.76035568962
388003	-357523.50384145475	357524.62901576946
388004	-357524.73569227388	357524.46080113784
388005	-357524.13877074968	357525.50881017023
388006	-357524.63877472008	357524.64037375496
388007	-357524.11377080297	357524.98630754586
388008	-357525.06173811021	357525.03856473637
388009	-357525.58097674936	357525.42689945316
388010	-956872.72810165398	357526.58097674936
388011	-956873.26657877362	956873.04249220958



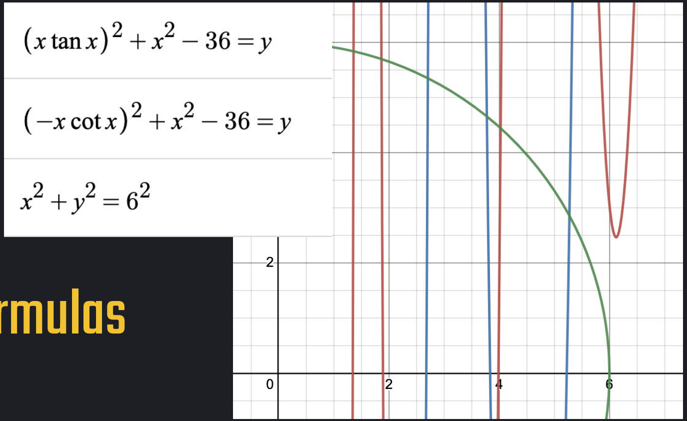


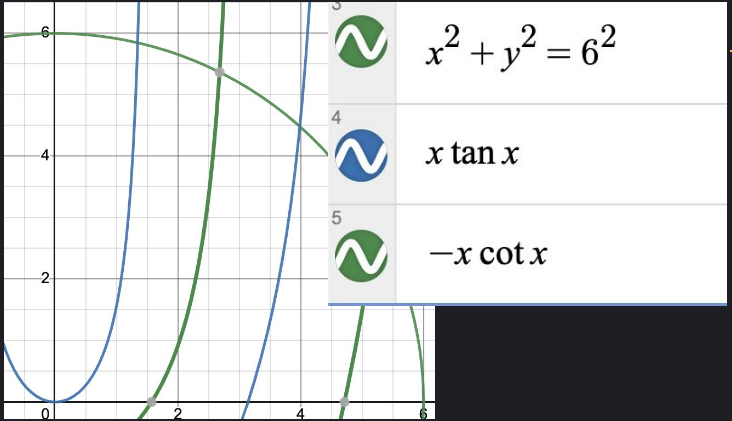
Conclusion

Bisection is better for NR for functions that iterate a lot.

04

Energy State





Mathematical Formulas

$$z \tan(z) = \sqrt{z_0^2 - z^2}$$

$$-z \cot(z) = \sqrt{z_0^2 - z^2}$$

->

Because we square the expression. There will be an extra negative values.

$$z \tan(z) = \sqrt{z_0^2 - z^2}$$

$$\Rightarrow (z \tan(z))^2 + z^2 - z_0^2 = \overset{0}{\overset{\shortparallel}{f_1(z)}}$$

$$-z \cot(z) = \sqrt{z_0^2 - z^2}$$

$$\Rightarrow (z \cot(z))^2 + z^2 - z_0^2 = \overset{0}{\overset{\shortparallel}{f_2(z)}}$$

```fortran
implicit none
double precision, parameter :: tol=1.0d-6 !maximum relative error
double precision x_left_n,x_right_n,x_middle_n,f1, f2
double precision x_iteration_old,x_iteration,relative_error
double precision :: tmp_odd,tmp_even, double_check
integer n
double precision :: r=6.0, dx=0.1, bound_1=0.00, bound_2=0.00, even_value=0.0, odd_value=0.0

 open(file='/Users/phihung/Documents/PHY/first/homework/hw_02/energies.dat', unit=12) !file to recc
    !find energy of the tangent expression
    do while(bound_2<r) !when the bound given is not searched
        bound_2 = bound_2 + dx !update second bound
        tmp_even = f1(bound_1)*f1(bound_2) ! check if the two bounds have the same sign
        if(tmp_even>0.0)then !if yes
            continue  ! continue to search for a good range
        else !initiate the zeroth iteration
            n=0 !zeroth iteration
            x_left_n=bound_1     !initial point on the left
            x_right_n=bound_2   !initial point on the right
            x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
            relative_error=1.0 !set inital error to any value higher than tol
            x_iteration_old=x_middle_n !zeroth iteration
        endif

        do while (relative_error>tol) !iterations untill relative error is less than tol
            n=n+1
            if(f1(x_left_n)*f1(x_middle_n)<=0.0)then
                x_right_n=x_middle_n !<--- the root is in left subinterval
                else
                x_left_n=x_middle_n  !<--- the root is in right subinterval
            endif
            x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
            x_iteration=x_middle_n
            relative_error=abs((x_iteration-x_iteration_old)/x_iteration)
            x_iteration_old=x_iteration
        enddo

        double_check = x_iteration*tan(x_iteration) !check if the value is positive
        if(even_value<x_iteration .AND. double_check>0.0)then !if satisfies the condition
            write(12,*) x_iteration  !write
            even_value = x_iteration !update
        endif
        bound_1 = bound_2 ! skip that range to initiate left bound as the old right bound
    enddo

bound_1=0.00
bound_2=0.00
   !find energy of the cotangent expression
   do while(bound_2<r)
      bound_2 = bound_2 + dx !update second bound
      !tmp=f1(x_left)*f1(x_right) !check if interval is correct
      tmp_odd = f2(bound_1)*f2(bound_2) ! check if the two bounds have the same sign
      if(tmp_odd>0.0)then !if yes
         continue  ! continue until the range is good
      else !initiate the zeroth iteration
         n=0 !zeroth iteration
         x_left_n=bound_1     !initial point on the left
         x_right_n=bound_2   !initial point on the right
         x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
         relative_error=1.0 !set inital error to any value higher than tol
         x_iteration_old=x_middle_n !zeroth iteration
      endif
      do while (relative_error>tol) !iterations untill relative error is less than tol
         n=n+1
         if(f2(x_left_n)*f2(x_middle_n)<=0.0)then
            x_right_n=x_middle_n !<--- the root is in left subinterval
            else
            x_left_n=x_middle_n  !<--- the root is in right subinterval
         endif
         x_middle_n=(x_left_n+x_right_n)/2.0 !finding the middle point
         x_iteration=x_middle_n
         relative_error=abs((x_iteration-x_iteration_old)/x_iteration)
         x_iteration_old=x_iteration
      enddo
      double_check = -x_iteration/tan(x_iteration)
      if(odd_value<x_iteration .AND. (-x_iteration/tan(x_iteration))>0.0)then
         write(12,*) x_iteration
         odd_value = x_iteration
      endif

      bound_1 = bound_2 ! skip that range to initiate left bound as the old right bound
   enddo
close(unit=12)
```

```fortran
double precision function f1(x)
  implicit none
  double precision x
  double precision :: z_0 = 6
  f1=(x*tan(x))**2+x**2-z_0**2
end function f1

double precision function f2(x)
  implicit none
  double precision x
  double precision :: z_0 = 6
  f2=(-x/tan(x))**2+x**2-z_0**2
end function f2
```

Z0=6

1.344751759540315 6
3.985824644354352 1
2.678782693725565 9
5.225961381583829 3

z0=5

1.3733253683645330
4.0886322630738050
6.6159607919448717
2.7394882610363993
5.4017182201403102

z0=20

1.495929740308668 0
4.486154241653594 0
7.471136585937983 9
10.44602676698605 1
13.40277119581060 1
16.32396264558974 5
19.14329862510112 4
2.991456648580083 3
5.979562467032110 4
8.960235729221039 9
11.92739885937271 5
14.86969016298189 7
17.75692165131749 8

z0=4

1.252353686874414 5
3.595303398300757 2
2.474577368417044 5

z0=2

1.0298668061176954
1.8954941078349066

05

06

01 02 03 04 05 06

Conclusion

All energies for the system with z_0 = 6.

```
1.3447517595403156
3.9858246443543521
2.6787826937255659
5.2259613815838293
```