PHYS 562 HW1

Raymond Ly

September 18, 2021

1 Abstract

Problem 1 asks us to write out a code of a D-dimensional quantum harmonic oscillator and plot it according to a set of values.

For number 2, I gathered the patterns of the incomplete gamma functions via continuous fractions from the online databases available to us in order to properly output a good set of data points which matches what a gamma function is supposed to do.

2 Problem 1

This problem asked for us to solve and plot the values of the harmonic oscillator when we vary n from 0 to 5, l and D between values 0 or 1 and 2 or 3 respectively. The wavefunction is represented in the image below.

$$\psi_{nl}(v,r) = v^{1/4}\left(\frac{2\Gamma(n+1)}{\Gamma(n+l+D/2)}\right)^{1/2}\exp\left(-\frac{v}{2}r^2\right)(vr^2)^{l/2+(D-1)/4}L_n^{(L+D/2-1)}(vr^2)$$

Figure 1: The D-dimensional harmonic oscillator

The process of making the equation work in the Fortran code resulted in splitting up the wavefunction into 4 parts, with the Laguerre and factorial (gamma) portions needing its own separate functions in order for the code to properly execute. Below, this image represents a generalized equation of the Laguerre Polynomial that is used in the code as a separate function.

$$L_0^{(\alpha)}(x) = 1$$
$$L_1^{(\alpha)}(x) = 1 + \alpha - x$$

$$L_{k+1}^{(\alpha)}(x) = \frac{(2k+1+\alpha-x)L_k^{(\alpha)}(x) - (k+\alpha)L_{k-1}^{(\alpha)}(x)}{k+1}.$$

Figure 2: The Laguerre

$$\Gamma(n) = (n-1)!\,.$$

Figure 3: The Gamma

The image above represents the portion of the code where the numerator and denominator which contains Gamma functions where this equation comes into play.

3 The Code, Problem 1

The makefile simply has one difference which is the oscillator output file.

Listing 1: The `Makefile`

```

OBJS1 = numtype.o oscillator.o # object files

PROG1 = osc # code name

F90 = gfortran

F90FLAGS = -O3 -funroll-loops # -fexternal-blas # optimization

#LIBS = -framework Accelerate # library

LDFLAGS = $(LIBS)
```

```
all: $(PROG1)

$(PROG1): $(OBJS1)
    $(F90) $(LDFLAGS) -o $@ $(OBJS1)

clean:
    rm -f $(PROG1) *.{o,mod} fort.*

.SUFFIXES: $(SUFFIXES) .f90

.f90.o:
    $(F90) $(F90FLAGS) -c $<
```

Listing 2: The `Numtype`

```
module NumType

    save
    integer, parameter :: dp = selected_real_kind(15,307)
    !integer, parameter :: qp = selected_real_kind(33,4931)
    real(dp), parameter :: pi = 4*atan(1._dp)
    complex(dp), parameter :: iic = (0._dp,1._dp)

end module NumType
```

Listing 3: The `Oscillation`

```
program oscillator

    use numtype
    implicit none
    real(dp) :: r, l, D, delta, charlie ! we are setting r = x
    real(dp) :: psi, a, b, c
        ! the psi function is split up into 3 seperate equations
    integer :: v, n


    v = 1 ! these will be changed according to the given values
```

```
    n = 3 ! n from 0 to 5, we will test graphs for n= 1, 2, 3
    l = 1 ! l from 0 to 1
    D = 2 ! D from 2 to 3
    r = 0 ! plug in a beginning r value

    delta = l + (D/2.0) +n !exponent
    charlie = l - 1 + (D/(2.0)) !exponent

    do while (r < 10)

        r = r + .05 ! step function

        a = ( v**(1.0/4.0) )*( (2*fact(n) ) / GAMMA(delta))**(1.0/2.0)
            ! replace delta
        b = exp((-v*r**2)/2)
        c = (v*r**2)**((l/2) + (D-1)/4)

        psi = a*b*c*Laguerre(n, r, charlie) !final equation

        print *, a, b, c !check for value consistency

        print *, r, psi ! final values of r (x) and psi (y)
        write(3,*) r, psi

    end do

    contains

        recursive function fact(n) result(s0) !factorial equation

            implicit none
            integer, intent(in) :: n
            real(dp) :: s0

            if (n<0) then
                stop 'something is wrong'
            else if (n == 0) then
                s0 = 1._dp
            else
                s0 = n * fact(n-1) !simple factorial
```

```
            end if

        end function fact


        recursive function Laguerre(n,r,charlie) result(s0)

            implicit none
            integer, intent(in) :: n
            real(dp) :: r
            real(dp) :: charlie !replace charlie with actual value

            real(dp) :: s0

            if (n < 0) then
                s0 = 0._dp

            else if (n == 0) then
                ! note, k = n - 1 so in long equation, k + 1 = n, etc...
                s0 = 1._dp
            else
                s0 = ( (2*(n-1)+1+charlie-r**2)*Laguerre(n-1,r,charlie)
                - (n-1+charlie)*Laguerre(n-2,r,charlie) )/ (n)
                !laguerre equation for first few terms
            end if

        end function Laguerre




end program oscillator
```

4 The Graphs, Problem 1

For the 6 graphs, I have separated them into graphs of n = values from 0 to 5. In these graphs they have 2 differing values where l can be 0 or 1 and D can be 2 or 3. For these graphs, the number of humps in most figures is n+1 except for the figures with values of D = 2 which ends up with 1 hump regardless of the value of n. All of the graphs are normalized to one and when compared with online sources of similar graphs demonstrates a consistency which shows that the data values I receive as output from executing the code is correct. In addition, they represent that the equations used and the values of different variables used is correct as well.

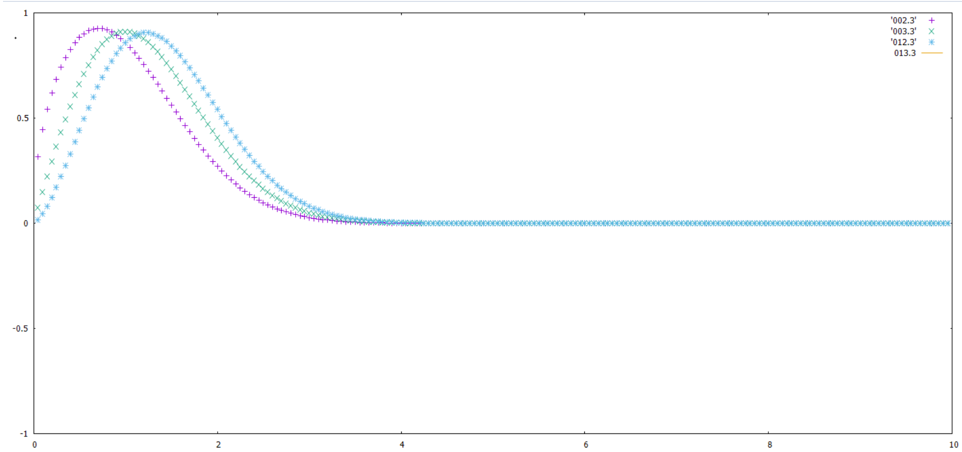

Figure 4: n=0

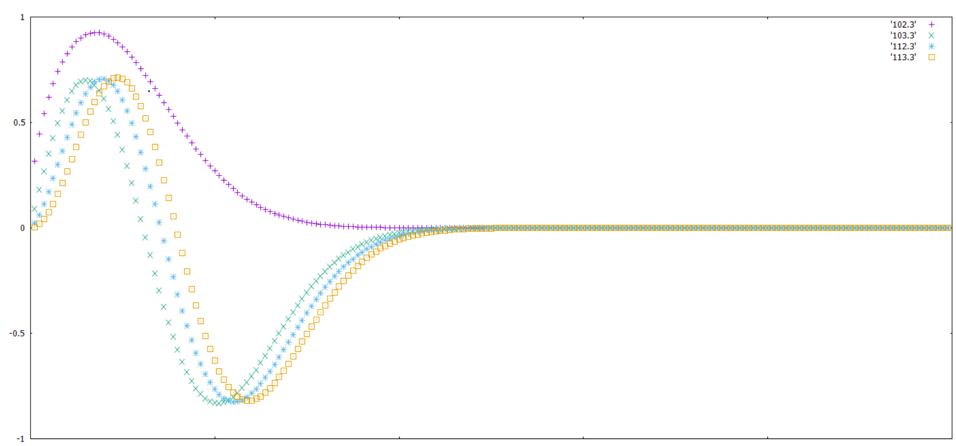

Figure 5: n=1

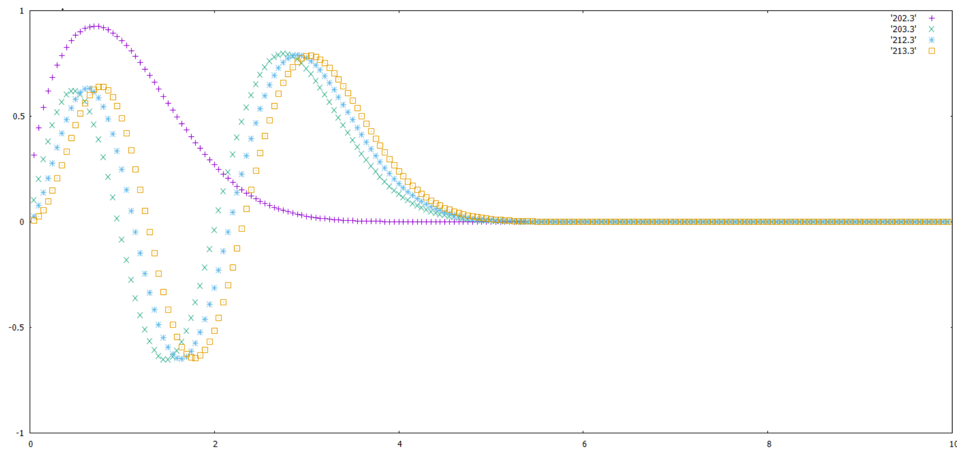

Figure 6: n=2

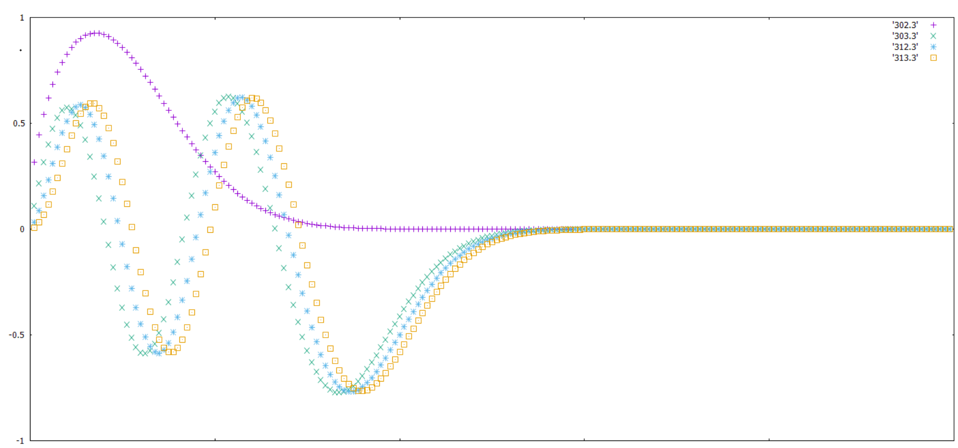

Figure 7: n=3

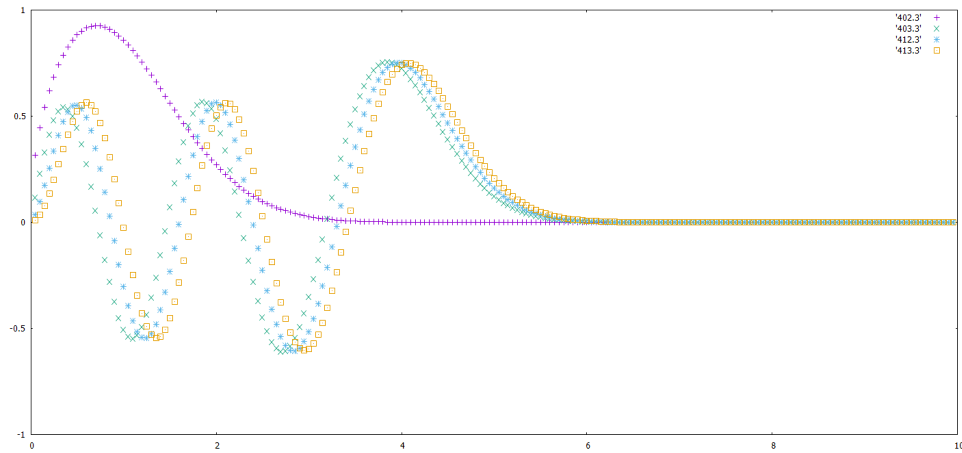

Figure 8: n=4

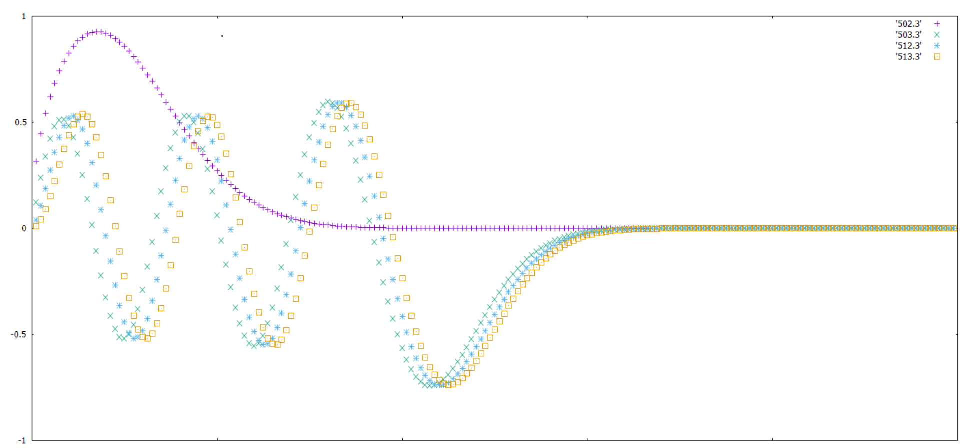

Figure 9: n=5

The graphs, along with the outputted Fortran files with the values of r and psi confirm that the code does work and does represent the wavefunction properly.

5 Problem 2

Problem two deals with two portions of the incomplete gamma function. The two functions that we are looking at are for the upper and lower gamma functions. The upper gamma function being

$$\Gamma\left(s,x\right)=\int\limits_{0}^{\infty}t^{s-1}e^{-t}ds \tag{1}$$

The lower function is as follows

$$\gamma\left(s,x\right)=\int\limits_{0}^{x}t^{s-1}e^{-t}ds \tag{2}$$

These two equations are then compiled in the form of continuous equations, most notably using these two patterns

$$\Gamma\left(a+1\right)\mathrm{e}^{z}\gamma^{*}\left(a,z\right)=\frac{1}{1-}\,\frac{z}{a+1+}\,\frac{z}{a+2-}\,\frac{(a+1)z}{a+3+}\,\frac{2z}{a+4-}\,\frac{(a+2)z}{a+5+}\,\frac{3z}{a+6-}\cdots,$$

$$z^{-a}\mathrm{e}^{z}\Gamma\left(a,z\right)=\frac{z^{-1}}{1+}\,\frac{(1-a)z^{-1}}{1+}\,\frac{z^{-1}}{1+}\,\frac{(2-a)z^{-1}}{1+}\,\frac{2z^{-1}}{1+}\,\frac{(3-a)z^{-1}}{1+}\,\frac{3z^{-1}}{1+}\cdots,$$

Figure 10: Continuous Fraction form of IGF

The two functions are placed into the code as two separate recursive functions with distinct bounds. The initial values of x and s were set to 1 and treated with a x-step size of one-half. The do loop tells the code that I want to keep running these equations up until x = 15, upon which the function will terminate itself once it reaches that boundary.

6 The Code, Problem 2

The numtype remains the same even for problem 2, the only difference in the Makefile is defining an output file for gamma, differing from the makefile of problem 1.

Listing 4: The `Makefile`

```

OBJS1 = numtype.o gamma.o # object files

PROG1 = gamma # code name

F90 = gfortran

F90FLAGS = -O3 -funroll-loops # -fexternal-blas # optimization

#LIBS = -framework Accelerate # library

LDFLAGS = $(LIBS)

all: $(PROG1)

$(PROG1): $(OBJS1)
	$(F90) $(LDFLAGS) -o $@ $(OBJS1)

clean:
	rm -f $(PROG1) *.{o,mod} fort.*

.SUFFIXES: $(SUFFIXES) .f90

.f90.o:
	$(F90) $(F90FLAGS) -c $<
```

Listing 5: The `Gamma`

```

program gamma

    use numtype
    implicit none
    real(dp) :: x, dx, s
```

```
x = 1._dp ! complex numbers
s = 1._dp ! real numbers
dx = 0.5_dp ! step functions

do while (x<15) ! while our x is less than 15

    x = x + dx ! ends when it hits 15

    write(1,*) x, lower(s,x)
        !lower function data, x, lower
    write(2,*) x, upper(s,x)
        !upper function data, x, upper
    write(3,*) x, upper(s,x) + lower(s,x)
        ! checks for proper addition of &
        gammas against x values

end do

print *, "lower␣gamma␣function:", lower(s,x)
! gives out a value of lower and upper gamma function
print *, "upper␣gamma␣function:", upper(s,x)

print *, "Total␣Gamma:", upper(s,x) + lower(s,x)

    contains

        recursive function upper(s,x) result(s1)
        !code for x to infinity of the gamma function

            implicit none
            real(dp) :: s1
            real(dp), intent(in) :: x, s
                !declare these as already &
                !being defined earlier
            integer :: n, i

            n = 500 ! max variable start
            s1 = 0._dp ! starts at 0

```

```
            do i = n , 1, -1 !range, ends at 1

                s1 =  x + (  i-s)/1 + &
                i/(x + (i + 1 - s)/1 + (i + 1))/s1
                    ! upper function
                    !pattern for both + and -

            end do

            s1 = ( exp(-x) * x ** s)/s1
            ! complete equation

        end function upper


        recursive function lower(s,x) result(s2)
            !code for 0 to x of the gamma function

            implicit none
            real(dp) :: s2
            real(dp), intent(in) :: x, s
                ! declare these values are already defined
            integer :: n, i

            n = 500
            s2 = 0._dp

            do i = n , 0, -1 !range, ends at 0

                s2 =  (( s + i) * x) / &
                ( i + 1 + s + x - s2)
                ! equation of the pattern for lower

            end do

            s2 = (exp(-x) * x**s)/ (s - s2)
            ! full equation


        end function lower
```

```
end program gamma
```

7 Problem 2 Graphs

The method I emplyed to check for proper output and verification of my data being correct is done in the same way. All 3, from my upper, lower and total gamma values are outputted into a fortran file where I have two defined values, x and the three differing 'y' values we are looking at.

In addition, I was able to verify this correctly, not only by looking at the values and comparing it with an online incomplete gamma function calculator for a single value of s, and differing values of x, but by plotting the outputs out and seeing if the general trend of the points follows that of normal upper and lower gamma function graphs.

When using the gnuplot application, the graphs clearly represent proper behavior of both upper and lower gamma function values. In addition, the total gamma graph clearly represents the fact that the combined values are as close as you can get to 1, which represents the consistency found from the code. This further confirms that the problem is represented properly up until a certain point. For the gamma graphs, they have been cut off at X=.5. Here, in my code, there is some sort of error that is telling me my output for the upper gamma function between $x = 0$ and $x = .5$ does not approach 1 as x gets closer to 0 rather the value of the upper gamma returns closer to 0, which in turn affects the final Total Gamma graph and values. The lower gamma values are unaffected, as when tested the values of gamma when x is close to 0 approach 0 as it should.

These errors are most likely coming either from the equation itself, which doesn't seem as likely, or coming from the actual ranges of variables defined and how I defined my own variable values.

```
1.5000000000000000       0.21104004841453536
2.0000000000000000       0.13106639672550482
2.5000000000000000        8.0440985405976884E-002
3.0000000000000000        4.9113279113307526E-002
3.5000000000000000        2.9907971775691607E-002
4.0000000000000000        1.8186683107943687E-002
4.5000000000000000        1.1049815573467811E-002
5.0000000000000000        6.7101200878672369E-003
5.5000000000000000        4.0734185012423216E-003
6.0000000000000000        2.4722329765953328E-003
6.5000000000000000        1.5002086649323673E-003
7.0000000000000000        9.1026024591363145E-004
7.5000000000000000        5.5226095859103820E-004
8.0000000000000000        3.3504031778802971E-004
8.5000000000000000        2.0324982128220185E-004
9.0000000000000000        1.2329578918049427E-004
9.5000000000000000        7.4791914751050199E-005
10.000000000000000        4.5368235701603061E-005
10.500000000000000        2.7519582589388063E-005
11.000000000000000        1.6692675182801360E-005
11.500000000000000        1.0125239396424451E-005
12.000000000000000        6.1415894366708831E-006
12.500000000000000        3.7252297808757655E-006
13.000000000000000        2.2595538713636469E-006
13.500000000000000        1.3705349596283412E-006
14.000000000000000        8.3129596512245043E-007
14.500000000000000        5.0421951574167575E-007
15.000000000000000        3.0583155216723284E-007
```

Figure 11: Upper Gamma: x, upper

1.5000000000000000	0.77686983985157021
2.0000000000000000	0.86466471676338730
2.5000000000000000	0.91791500137610083
3.0000000000000000	0.95021293163213638
3.5000000000000000	0.96980261657768163
4.0000000000000000	0.98168436111126556
4.5000000000000000	0.98889100346175740
5.0000000000000000	0.99326205300091697
5.5000000000000000	0.99591322856153253
6.0000000000000000	0.99752124782333018
6.5000000000000000	0.99849656080703597
7.0000000000000000	0.99908811803444486
7.5000000000000000	0.99944691562985277
8.0000000000000000	0.99966453737207939
8.5000000000000000	0.99979653163102278
9.0000000000000000	0.99987659019588582
9.5000000000000000	0.99992514817012090
10.000000000000000	0.99995460007003434
10.500000000000000	0.99997246355073244
11.000000000000000	0.99998329829915633
11.500000000000000	0.99998986990692540
12.000000000000000	0.99999385578791233
12.500000000000000	0.99999627334709484
13.000000000000000	0.99999773967174710
13.500000000000000	0.99999862904321646
14.000000000000000	0.99999916846440706
14.500000000000000	0.99999949565808033
15.000000000000000	0.99999969408833611

Figure 12: Lower Gamma: x, lower

1.5000000000000000	0.9879098826610560
2.0000000000000000	0.9957311134889209
2.5000000000000000	0.9983559867820771
3.0000000000000000	0.9993262107454392
3.5000000000000000	0.9997105883533723
4.0000000000000000	0.9998710442192095
4.5000000000000000	0.9999408190352225
5.0000000000000000	0.9999721730887417
5.5000000000000000	0.9999866470627485
6.0000000000000000	0.9999934807992556
6.5000000000000000	0.9999967694719683
7.0000000000000000	0.9999983782803585
7.5000000000000000	0.9999991765884438
8.0000000000000000	0.9999995776898674
8.5000000000000000	0.9999997814523049
9.0000000000000000	0.9999998859850662
9.5000000000000000	0.9999999400848719
10.000000000000000	0.9999999683057359
10.500000000000000	0.9999999831333218
11.000000000000000	0.9999999909743391
11.500000000000000	0.9999999951463218
12.000000000000000	0.9999999973773490
12.500000000000000	0.9999999985768757
13.000000000000000	0.9999999992256184
13.500000000000000	0.9999999995781760
14.000000000000000	0.9999999997603722
14.500000000000000	0.9999999998775960
15.000000000000000	0.9999999999198883

Figure 13: Total Gamma: x, total

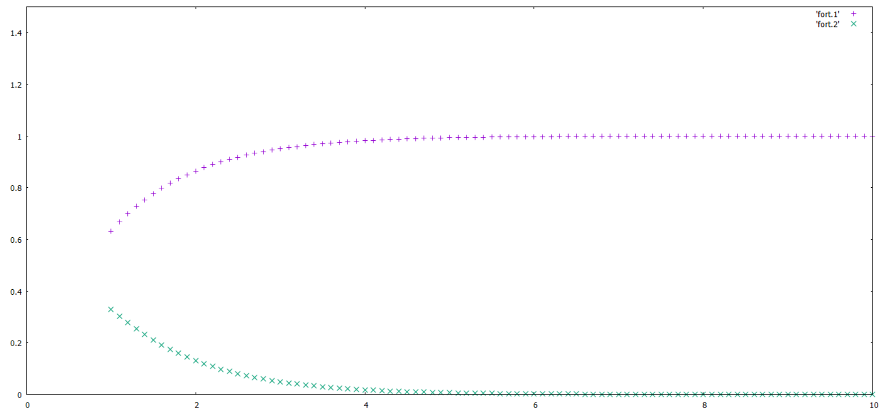

Figure 14: Upper+Lower Gamma Graph: x, Gamma (value)

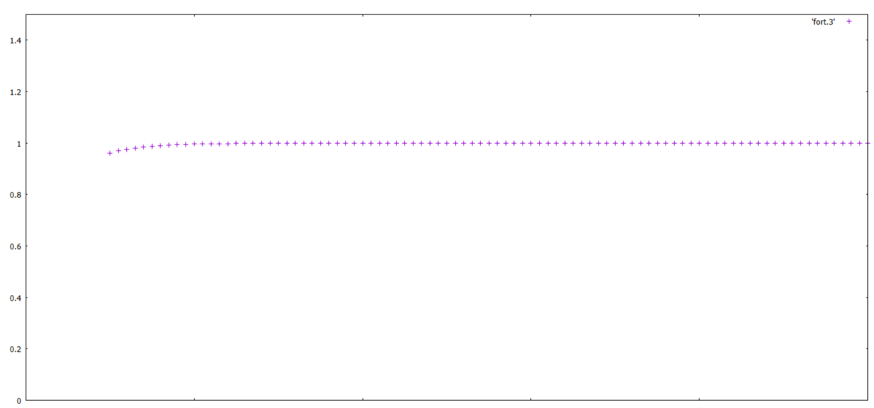

Figure 15: Total Gamma Graph: x, Total Gamma Value

References

[1] §8.9 continued fractions. DLMF. (n.d.). Retrieved September 22, 2021, from https://dlmf.nist.gov/8.9.

[2] Incomplete gamma function calculator. High accuracy calculation for life or science. (n.d.). Retrieved September 22, 2021, from https://keisan.casio.com/exec/system/1180573447.

[3] Wikimedia Foundation. (2021, September 13). Incomplete gamma function. Wikipedia. Retrieved September 22, 2021, from https://en.wikipedia.org/wiki/Incomplete_gamma_function.

[4] Papp, Zoltan. "Mastering Computational Physics Lecture Notes."

[5] Armando Reynoso for his help in guiding me along the wavefunction problem and helping debug my code

[6] Fanuel Mendez for his help in understanding parts of the physics behind the problems

[7] Derek Wingard for his help in fixing the problems in my code and guiding me.

[8] Rami Allaf for his help in understanding the concepts of the gamma function

[9] Anise Mansour for his help in understanding how to operate gnuplot properly