Physics 562 - Computational Physics

Assignment 2: Quantum Harmonic Oscillator

Josh Fernandes
Department of Physics & Astronomy
California State University Long Beach

March 3, 2014

Abstract

The assignment focuses on solving for the eigenfunctions for the one dimensional quantum harmonic oscillator. In the process, programs to imitate the exponential function, hermit function, and factorial function are created. The ground state and the first six excited states are plotted on a graph. The results are not normalized, so this paper only concludes that the amplitude decreases and the wavelength decreases as the energy state is increased.

1 Introduction

While the classical harmonic oscillator is well suited for solving systems of mechanical springs and pendulums, the quantum harmonic oscillator can solves systems of atomic particles. In addition, the quantum harmonic oscillator is one of the few quantum mechanical systems that has an exact, analytical solution. The eigenfunctions for the quantum harmonic oscillator are of the form

$$\Psi_n(x) = \frac{1}{\sqrt{2^n\, n!}} \cdot \left(\frac{mw}{\pi\hbar}\right)^{\frac{1}{4}} \cdot e^{-\frac{mwx^2}{2\hbar}} \cdot H_n\left(\sqrt{\frac{mw}{\hbar}}\, x\right). \tag{1}$$

The parameters are set as $m = 1$, $w = 1$, and $\hbar = 3$. Although these values have no physical meaning, they preserve the shape of the curves. The first seven energy states are plotted in order to determine traits about the curves.

2 The Fortran95 code

The code is going to solve the problem. First a module called `NumType` is created to store all my global parameters. These include values of pi and the exponential of one.

Listing 1: Module `NumType`

```

module NumType

    save
    integer, parameter :: dp = kind(1.d0)
    real(dp), parameter :: pi = 4*atan(1._dp), &
    e = exp(1._dp)
    complex(dp), parameter :: iic = (0._dp,1._dp)

end module NumType
```

The main program is `oscillator` and it begins with its own module. In this module, I call in numtype.

Listing 2: `oscillator.f95`

```

module initiate_phase_one

    use NumType
    implicit none
    real(dp), parameter :: w = 1._dp, &
    m = 1._dp, Xmin = -10._dp, Xmax = 10._dp
    real(dp) :: x, k, plot1(5000),plot2(5000),&
    plot3(5000), plot4(5000),plot5(5000), &
    plot6(5000),plot7(5000), plot8(5000)
    REAL, DIMENSION(:, :), ALLOCATABLE :: psi
    integer, Dimension(:), ALLOCATABLE :: n
```

```
    real(dp), Dimension(:), ALLOCATABLE :: curve
    integer :: i, j, DeAllocateStatus, &
    AllocateStatus, steps


end module initiate_phase_one

program awesome

    use initiate_phase_one

    n = (/0,1, 2, 3, 4, 5, 6, 7/)

    j = 0._dp
    k = 0.5_dp

    steps = 5000
    x = Xmin
    dx = (Xmax-Xmin)/steps



    ALLOCATE (curve(steps), STAT = AllocateStatus)
    IF (AllocateStatus /= 0) &
    STOP "*** Not enough memory ***"

    ALLOCATE ( psi(size(n), steps), STAT = AllocateStatus)
    IF (AllocateStatus /= 0) &
    STOP "*** Not enough memory ***"

    do while(x < Xmax)
        j = j + 1
        curve(j) = 0.5_dp*k*x**2
        plot1(j) = 1
        plot2(j) = 3
        plot3(j) = 5
        plot4(j) = 7
        plot5(j) = 9
        plot6(j) = 11
        plot7(j) = 13
```

```
        plot8(j) = 15
        do i=1,size(n)
            psi(i,j) = quantum_oscillator(n(i),m,w,x)
        end do
        print *, x, 1+psi(1,j), 3+psi(2,j), &
        5+psi(3,j), 7+psi(4,j), 9+psi(5,j), &
        11+psi(6,j), 13+psi(7,j), 15+psi(8,j), &
        curve(j), plot1(j), plot2(j), plot3(j), &
        plot4(j), plot5(j), plot6(j), plot7(j), plot8(j)
        !iterate x
        x = x + dx
    end do

    DEALLOCATE (curve, STAT = DeAllocateStatus)
    DEALLOCATE (psi, STAT = DeAllocateStatus)




    contains

        function quantum_oscillator(n,m,w,x) result(psi)

            implicit none
            real(dp), parameter :: hbar = 3._dp
            real(dp) :: m,w,x,psi
            integer :: n

            psi = 1/sqrt(real((2**n)*factorial(n)))*&
            ((m*w)/(pi*hbar))**(1/4)&
            *exxp(-(m*w*x**2)/(2*hbar))*&
            hermite(n,(sqrt(m*w/hbar)*x))

        end function quantum_oscillator

        recursive function exxp(x) result(ex)

        real(dp) :: x, ex, E0
        integer :: i, imax

```

```fortran
    imax = 20
    if ( abs(x) < 1._dp) then
        E0 = 1._dp
        ex = 1._dp
        do i =1, imax
            E0 = E0*x/i
            ex = ex + E0
        end do
    else if (1._dp <= x) then
        ex = e * exxp(x-1)
    else if (x <= 1._dp) then
        ex = exxp(x+1) / e
    end if

    end function exxp

    recursive function hermite(n,x) result(hpol)

        real(dp) :: x, hpol
        integer :: n

        if ( n < 0 ) then
            stop "n can't be less than zero"
        else if ( n == 0) then
            hpol = 1._dp
        else if ( n == 1) then
            hpol = 2*x
        else
            hpol = 2*x*hermite(n-1,x) - &
            2*(n-1)*hermite(n-2,x)
        end if


    end function hermite


    recursive function factorial(n) &
    result(factorial_number)

        implicit none
```

```
            integer :: n, factorial_number


            if ( n < 0 ) then
                stop "you know that's wrong"
            else if ( n == 0) then
                factorial_number = 1
            else
                factorial_number = n*factorial(n-1)
            end if



        end function factorial


end program awesome
```

The code is run by typing `./wowza`. The results are printed on the screen, or the data can redirect to the file `amazing data` by typing `./wowza > amazingdata.data`. The `Plot` provides the picture in Fig. 1.

3 Data Analysis

the code returns an array of values.

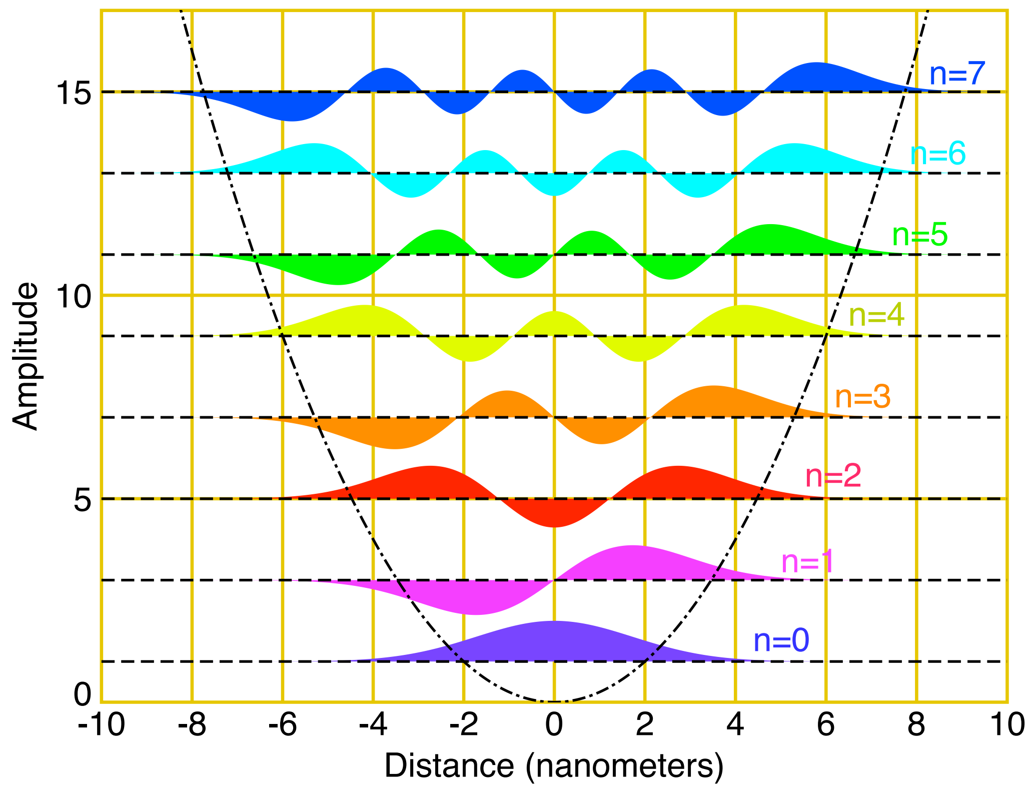

Figure 1: Results of the `oscillator.f95` code plotted on a linear scale.

4 Summary and conclusions

The eigenfunctions Ψ for the first eight energy states were plotted. The graph demonstrates that as n increases the amplitude decreases and the wavelength decreases.

References

[1] M. Metcalf, J. Reid and M. Cohen, *Fortran 95/2003 explained*. Oxford University Press, 2004.