# Physics 562 - Computational Physics

# Assignment 2: Quantum Harmonic Oscillator

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

February 11, 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 Makefile in Listing 1 describes the structure of the code. The codes in `OBJS1` are compiled together. The `.o` files were created from the `.f95` file by using the `F95` command with flags `F95FLAGS`. In building the executable code `LDFLAGS` were used, which contains the `LIB` library. These `LDFLAGS` are optimal for Intel Core2 architecture and uses the Apple's `vecLib` library from Xcode. By typing `make` we can produce the executable file.

Listing 1: The `Makefile`

```
1
2  OBJS = numtype.o oscillator.0
3
4  PROG = wowza
5
6  F95 = gfortran
7
8  F95FLAGS = -O3 -funroll-loops -fexternal-blas
9
10 LIBS = -framework vecLib
11
12 LDFLAGS = $(LIBS)
13
14 all: $(PROG)
15
16 $(PROG): $(OBJS)
17     $(F95) $(LDFLAGS) -o $@ $(OBJS)
18
19 clean:
20     rm -f $(PROG) *.{o,mod}
21
22 .SUFFIXES: $(SUFFIXES) .f95
23
24 .f95.o:
```

```
25      $(F95) $(F95FLAGS) -c $<
```

The Fortran95 code is shown below. The precision is defined in the Module `NumType`. The constants parameters are defined in Module `setup`. The implicit none statement ensures that all the variables has to be defined. `do while` is used to increase the value of x while also calculating $\Psi$ for the eight different $n$ values. The resulting wave functions are plotted with an offset on the y axis so they can be clearly seen. A horizontal line is plotted at the offset for clarity using the arrays $plot1$ through $plot8$. The function `contains` is a way of defining a function that can be reused within the code. In this code, three different functions are defined. Note that in each function, the variables that are used have to be defined.

Listing 2: Module `NumType`

```
1
2   module NumType
3
4       save
5       integer, parameter :: dp = kind(1.d0)
6       real(dp), parameter :: pi = 4*atan(1._dp), &
7       e = exp(1._dp)
8       complex(dp), parameter :: iic = (0._dp,1._dp)
9
10  end module NumType
```

Listing 3: `oscillator.f95`

```
1
2   module initiate_phase_one
3
4       use NumType
5       implicit none
6       real(dp), parameter :: w = 1._dp, &
7       m = 1._dp, Xmin = -10._dp, Xmax = 10._dp
8       real(dp) :: x, k, plot1(5000),plot2(5000),&
9       plot3(5000), plot4(5000),plot5(5000), &
10      plot6(5000),plot7(5000), plot8(5000)
11      REAL, DIMENSION(:, :), ALLOCATABLE :: psi
12      integer, Dimension(:), ALLOCATABLE :: n
13      real(dp), Dimension(:), ALLOCATABLE :: curve
```

```
14      integer :: i, j, DeAllocateStatus, &
15      AllocateStatus, steps
16
17
18  end module initiate_phase_one
19
20  program awesome
21
22      use initiate_phase_one
23
24      n = (/0,1, 2, 3, 4, 5, 6, 7/)
25
26      j = 0._dp
27      k = 0.5_dp
28
29      steps = 5000
30      x = Xmin
31      dx = (Xmax-Xmin)/steps
32
33
34
35      ALLOCATE (curve(steps), STAT = AllocateStatus)
36      IF (AllocateStatus /= 0) &
37      STOP "*** Not enough memory ***"
38
39      ALLOCATE ( psi(size(n), steps), STAT = AllocateStatus)
40      IF (AllocateStatus /= 0) &
41      STOP "*** Not enough memory ***"
42
43      do while(x < Xmax)
44          j = j + 1
45          curve(j) = 0.5_dp*k*x**2
46          plot1(j) = 1
47          plot2(j) = 3
48          plot3(j) = 5
49          plot4(j) = 7
50          plot5(j) = 9
51          plot6(j) = 11
52          plot7(j) = 13
53          plot8(j) = 15
```

```
54          do i=1,size(n)
55              psi(i,j) = quantum_oscillator(n(i),m,w,x)
56          end do
57          print *, x, 1+psi(1,j), 3+psi(2,j), &
58          5+psi(3,j), 7+psi(4,j), 9+psi(5,j), &
59          11+psi(6,j), 13+psi(7,j), 15+psi(8,j), &
60          curve(j), plot1(j), plot2(j), plot3(j), &
61          plot4(j), plot5(j), plot6(j), plot7(j), plot8(j)
62          !iterate x
63          x = x + dx
64      end do
65
66      DEALLOCATE (curve, STAT = DeAllocateStatus)
67      DEALLOCATE (psi, STAT = DeAllocateStatus)
68
69
70
71
72      contains
73
74          function quantum_oscillator(n,m,w,x) result(psi)
75
76              implicit none
77              real(dp), parameter :: hbar = 3._dp
78              real(dp) :: m,w,x,psi
79              integer :: n
80
81              psi = 1/sqrt(real((2**n)*factorial(n)))*&
82              ((m*w)/(pi*hbar))**(1/4)&
83              *exxp(-(m*w*x**2)/(2*hbar))*&
84              hermite(n,(sqrt(m*w/hbar)*x))
85
86          end function quantum_oscillator
87
88          recursive function exxp(x) result(ex)
89
90          real(dp) :: x, ex, E0
91          integer :: i, imax
92
93          imax = 20
```

```
94          if ( abs(x) < 1._dp) then
95              E0 = 1._dp
96              ex = 1._dp
97              do i =1, imax
98                  E0 = E0*x/i
99                  ex = ex + E0
100             end do
101         else if (1._dp <= x) then
102             ex = e * exxp(x-1)
103         else if (x <= 1._dp) then
104             ex = exxp(x+1) / e
105         end if
106
107         end function exxp
108
109         recursive function hermite(n,x) result(hpol)
110
111             real(dp) :: x, hpol
112             integer :: n
113
114             if ( n < 0 ) then
115                 stop "n can't be less than zero"
116             else if ( n == 0) then
117                 hpol = 1._dp
118             else if ( n == 1) then
119                 hpol = 2*x
120             else
121                 hpol = 2*x*hermite(n-1,x) - &
122                 2*(n-1)*hermite(n-2,x)
123             end if
124
125
126         end function hermite
127
128
129         recursive function factorial(n) &
130         result(factorial_number)
131
132             implicit none
133             integer :: n, factorial_number
```

```
134
135
136             if ( n < 0 ) then
137                 stop "you know that's wrong"
138             else if ( n == 0) then
139                 factorial_number = 1
140             else
141                 factorial_number = n*factorial(n-1)
142             end if
143
144
145
146         end function factorial
147
148
149 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.

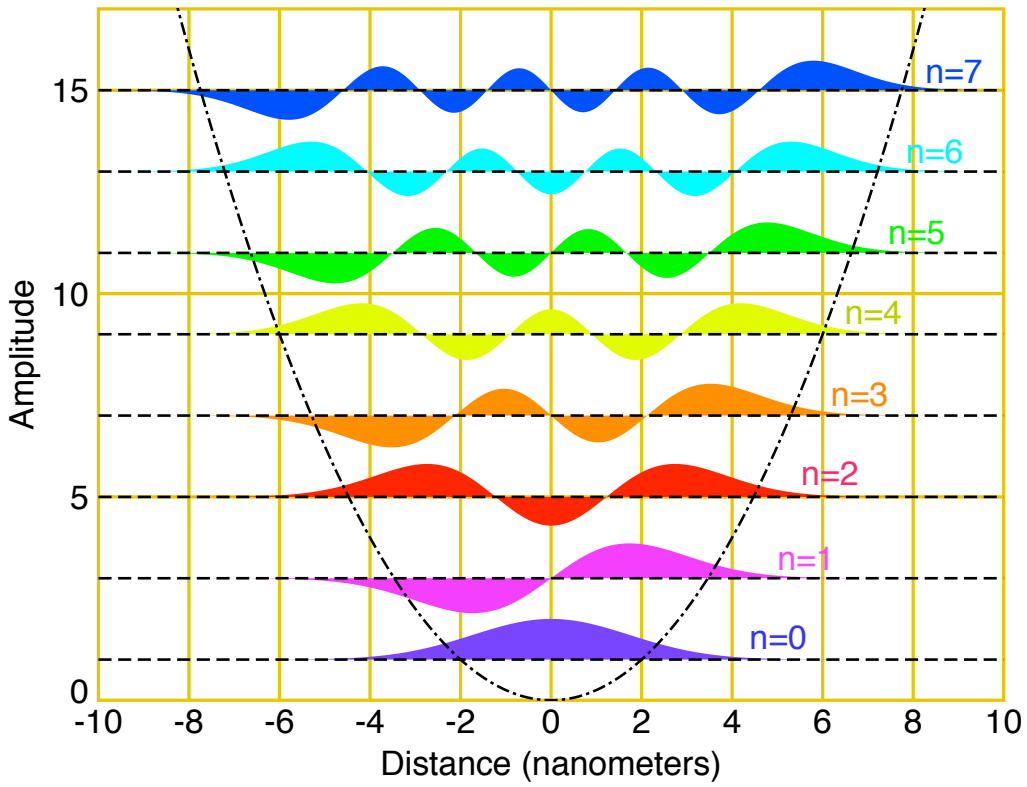


Figure 1: Results of the `oscillator.f95` code plotted on a linear scale.

## 3 Summary and conclusions

The eigenfunctions $\Psi$ 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.