# Homework 5, Computational Physics

Seann Smallwood

December 10, 2020

**Abstract**

The goal of this work was to analyze systems through various computational methods and tools. First we used a numerical method to compute the integral of a given function. Second, the one dimensional harmonic oscillator was analyzed via differential equation solvers and matrix operations. Next, curve fitting allowed us to determine a scattering potential function from energy and reflection coefficient data. Lastly, given various charge distributions, an O.D.E solver was used to calculate the electric potential function, energy, and total charge from the Poisson equation.

## 1 Introduction

### 1.1 Numerical Integration

The task was to calculate the following integral numerically.

$$\int_0^1 \sqrt{x}$$

Optimizing for accuracy and computation time we would then compare our result to the formally calculated value of 2/3.

### 1.2 One Dimensional Harmonic Oscillator

The first objective was to calculate the first 25 wave functions of the one-dimensional harmonic oscillator. We then were to apply the following Hamiltonian operator

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} - \frac{\lambda(\lambda+1)}{2} sech^2(x)$$

and calculate the eigenvalues and eigenvectors.Lastly we were to find the value for omega in the potential function that gives the lowest states, and plot.

## 1.3 Scattering Potential Function

Given the below data showing the dependence of the reflection coefficients on scattering energy we were to determine the scattering potential function.

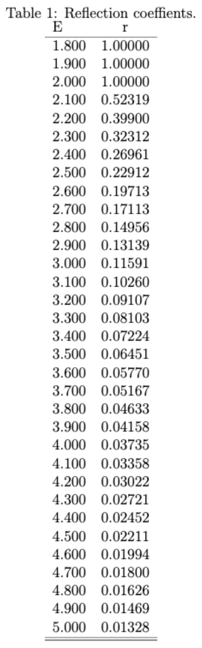

Table 1: Reflection coeffients.

| E | r |
|---|---|
| 1.800 | 1.00000 |
| 1.900 | 1.00000 |
| 2.000 | 1.00000 |
| 2.100 | 0.52319 |
| 2.200 | 0.39900 |
| 2.300 | 0.32312 |
| 2.400 | 0.26961 |
| 2.500 | 0.22912 |
| 2.600 | 0.19713 |
| 2.700 | 0.17113 |
| 2.800 | 0.14956 |
| 2.900 | 0.13139 |
| 3.000 | 0.11591 |
| 3.100 | 0.10260 |
| 3.200 | 0.09107 |
| 3.300 | 0.08103 |
| 3.400 | 0.07224 |
| 3.500 | 0.06451 |
| 3.600 | 0.05770 |
| 3.700 | 0.05167 |
| 3.800 | 0.04633 |
| 3.900 | 0.04158 |
| 4.000 | 0.03735 |
| 4.100 | 0.03358 |
| 4.200 | 0.03022 |
| 4.300 | 0.02721 |
| 4.400 | 0.02452 |
| 4.500 | 0.02211 |
| 4.600 | 0.01994 |
| 4.700 | 0.01800 |
| 4.800 | 0.01626 |
| 4.900 | 0.01469 |
| 5.000 | 0.01328 |

Figure 1: Scattering Energy and Reflection Coefficients

## 1.4 Electric Potential

Using the Poisson equation for spherically symmetric charge distributions

$$\frac{1}{r^2}\frac{d}{dr}(r^2(\frac{d\Phi(r)}{dr})) = -4\pi\rho(r)$$

we were to determine the potential $\Phi$, electric field, and total charge for the following charge distributions

$$\rho(r) = \frac{1}{8\pi}e^{-r}$$

$$\rho(r) = \frac{1}{24\pi}re^{-r}$$

$$\rho(r) = \frac{1}{2\pi}sin(r)e^{-r}$$

$$\rho(r) = \frac{1}{2\pi}cos(r)e^{-r}$$

## 1.5 Setup and General Methods

Fortran was used to analyze each system. To implement common mathematical terms and define the precision to which values were calculated a file was created. This file, named numtype, is shown below.

```
    module numtype

    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
```

A Makefile was used to compile the fortran code and create and executable file.

```
OBJS1 = numtype.o prob2.o

PROG1 = run

F90 = gfortran

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

LIBS = -framework Accelerate

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 $<
```

# 2 Solutions

## 2.1 Numerical Integration

The Romberg method of numerical integration was used as follows

```
program prob1
    use numtype
    implicit none
    real(dp), external :: func1
    real(dp) :: a, b, res, eps
    integer :: nint

    b = 1
    a = 0
    nint = 20
    eps = 1.e-10_dp

    call rombint(a,b, func1, res, nint, eps)
    print *, res

end program prob1

subroutine rombint( a, b, func, res, n, eps )
    ! Romberg integration
    !   res = int_a^b f(x) dx
    !   eps  required accuracy
    !   n    n_max in approx (input)
    !        accuaracy reached after n steps (output)
        use numtype
        implicit none
        integer, parameter :: maxint = 300
        real(dp) :: a, b, eps, res
        real(dp), external :: func
        integer ::  np, i, j, k, m, n
        real(dp) :: h, sumt, r(maxint,maxint)

        h = b - a
        np = 1
        r(1,1) = h/2 * ( func(a) + func(b) )
        res = r(1,1)
        do i=2,n
            h = h/2
            np = 2*np
            sumt = 0.0_dp
            do k=1,(np-1),2
                sumt = sumt + func( a + k*h)
            end do
            r(i,1) = 0.5_dp * r(i-1,1) + h * sumt
            m = 1
            do j=2,i
                m = 4*m
                r(i,j) = r(i,j-1) + (r(i,j-1)-r(i-1,j-1))/(m-1)
            end do
            if ( abs(res-r(i,i)) < eps ) then
                n = i
                res = r(i,i)
                return
            end if
            res = r(i,i)
        end do
        !print *,' romint :',eps,r(i-1,i-1),res

end subroutine rombint




function func1(x)
    use numtype
    implicit none
    real(dp) :: x, func1

    func1 = sqrt(x)

end function func1
```

## 2.2 One Dimensional Harmonic Oscillator

The one dimensional harmonic oscillator has a potential function

$$V(x) = \frac{1}{2} m\omega^2 x^2$$

and the n by n Hamiltonian operator

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} - \frac{\lambda(\lambda+1)}{2} sech^2(x)$$

The operator acts on the wave functions $\psi_n$ as follows allowing us to use internal LAPACK functions to calculate the eigenvalues and vectors.

$$\begin{bmatrix} H & 0 & \dots & 0 & 0 \\ 0 & H & \dots & 0 & 0 \\ 0 & 0 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & H \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \dots \\ \dots \\ \psi_n \end{bmatrix}$$

The wave functions were generated using the reverse Runge-Kutta O.D.E. method as follows

```
module setup

    use numtype
    implicit none

    integer, parameter :: n_eq = 3

    real(dp), parameter :: hbar = 1._dp, hbar2 = hbar**2, mass = 1._dp

    real(dp), parameter ::  omega = 1.0_dp, x0 = sqrt(hbar/(mass*omega))

    real(dp) :: energy, xmax, dstep

    real(dp), allocatable, dimension(:,:) :: wf

    integer :: imax

end module setup
```

```
program prob2

    use setup
    use chebyshev
    implicit none

    real(dp) :: e_min, e_max, e0
    real(dp), external :: psi0
    integer :: nch, iz

    xmax = 15
    dstep = 0.001_dp
    imax = abs(nint(xmax/dstep))

    allocate( wf( -imax:imax , 2 ))

    e_min = 0
    e_max = 12

    nch = 40

    call chebyex( psi0, nch, cheb, e_min, e_max )

    call chebyzero( nch, cheb, e_min, e_max , z0, iz0 )

    print *, iz0

    do iz = 1, iz0

        e0 = z0(iz)
        print *,' 1   :', z0(iz), e0
        call wavefun( iz, e0 )


    end do

end program prob2


subroutine wavefun(iw, e)
```

```
use setup
implicit none

real(dp), intent(in) :: e
real(dp) :: x, psi(n_eq), parity
integer :: i, iw

energy = e

x = xmax

psi(1) =  exp( -x**2/(2*x0**2))          ! psi
psi(2) =  - x/x0**2 * psi(1)                        ! psi'
psi(3) = 0


do while ( x > 0 )

    call rk4step ( x, -dstep , psi )

end do

if ( abs(psi(1)) > abs(psi(2)) ) then
    parity = 1
else
    parity = -1
end if

x = xmax

psi(1) =  exp( -x**2/(2*x0**2))/sqrt(2*psi(3))          ! psi
psi(2) =  - x/x0**2 * psi(1)                        ! psi'
psi(3) = 0

i = imax + 1

do while (x > 0)
    i = i-1
    wf(i,1) = x; wf(-i,1) = -x
```

```
        wf(i,2) = psi(1); wf(-i,2) = parity * psi(1)
        call rk4step( x, -dstep, psi)
    end do

    do  i = -imax/2 , imax/2
        write(30+iw,*) wf(i,1), wf(i,2)
    end do

end subroutine  wavefun



function psi0(e)

    use setup
    implicit none

    real(dp), intent(in) :: e
    real(dp) ::  psi0, x
    real(dp), dimension(n_eq) :: psi

    energy = e

    x = xmax

    psi(1) =  exp( -x**2/(2*x0**2))           ! psi
    psi(2) =  - x/x0**2 * psi(1)                          ! psi'
    psi(3) = 0

    do while ( x > 0 )

        call rk4step ( x, -dstep , psi )

    end do

    psi0 = psi(1) * psi(2)


end function  psi0
```

```
subroutine rk4step ( x, h, y )

    use setup
    implicit none
    real(dp), intent(inout) :: x
    real(dp), intent(in) :: h
    real(dp), intent(inout), dimension(n_eq) :: y
    real(dp), dimension(n_eq) :: k1, k2, k3, k4, dy

    k1 = kv( x, h, y )
    k2 = kv( x+h/2, h, y + k1/2 )
    k3 = kv( x+h/2, h, y + k2/2 )
    k4 = kv( x+h , h,  y + k3 )
    dy = ( k1 + 2*k2 + 2*k3 + k4 ) / 6

    x = x + h
    y = y + dy

    contains

        function kv( x, dx, y ) result(k)

            use setup
            implicit none
            real(dp), intent(in) :: x, dx
            real(dp), intent(in), dimension(n_eq) :: y
            real(dp), dimension(n_eq) :: f, k
!-----------------------------------------------------------
            f(1) =  y(2)
            f(2) =  - 2*mass/hbar2 * ( energy - potential(x)) * y(1)
            f(3) = - abs(y(1))**2
!_____________________________________________________________
            k = dx * f

        end function kv

        function potential(x)
```

```
            use setup
            real(dp) :: x, potential

            potential = 0.5_dp * mass * omega**2 * x**2

        end function potential


end subroutine rk4step
```

## 2.3 Scattering Potential

The data was read in and set to be used in a curve fit routine called the downhill method.

```
    module setup

    use numtype
    implicit none

    !real(dp), parameter ::


    integer, parameter :: npmax = 100, npar = 2
    integer, parameter :: nspmin = 1, nspmax = 43
    real(dp) :: xx(1:npmax), yy(1:npmax)
    integer ::  nsp, ical, iprint

end module setup

program prob3

    use setup
    implicit none

    integer :: stat, i, itmin, itmax
    real(dp), external :: scatpot
    real(dp) :: xstart(npar), fstart, stepi, epsf
    open(unit=2, file='reflectiondata.txt')
    stat = 0

    i = 1
    do
        read(2,*, iostat= stat) xx(i),yy(i)
        if( stat /= 0 ) exit
        print *,i,xx(i),yy(i)
        i = i + 1
    end do
    nsp = i-1
    close(2)

    xstart(1:npar) = (/ 1 , 1 /)


    ical = 0
    iprint = 7
    fstart = scatpot (xstart)
    stepi= 0.05_dp
    epsf = 0.001_dp
    itmin = 100
    itmax = 1000

    iprint = 0
    call downhill(npar,scatpot,xstart,fstart,stepi,epsf,itmin,itmax)

    iprint = 17
    fstart = scatpot (xstart)
    print *, xstart(1:npar)

end program prob3



function scatpot (par)

    use setup
    implicit none
    real(dp) :: scatpot
    real(dp) :: par(npar)
    real(dp) :: scale, temp, x, fi
    integer :: i

    ical = ical + 1
    scale = par(1); temp = par(2)

    scatpot = 0

    do i = nspmin, nspmax

        x = xx(i)
        !fi = scale * x**3 * 1/( exp( hc * x / ( k_b * temp) ) - 1 )

        !scatpot = scatpot +  ( yy(i) - fi )**2  * 1/sqrt(  2._dp + yy(i) )

    end do
    scatpot = scatpot / abs(nspmax-nspmin)
    print '(i4,2x,2f12.2,3x,f20.4)',ical, par(1:npar), scatpot

    ! printing
    if (  iprint /= 0 ) then

        do i = nspmin, nspmax

            x = xx(i)
            !fi =  scale * x**3 * 1/( exp( hc * x / ( k_b * temp) ) - 1 )
            write( unit=iprint, fmt='(3f15.4)') xx(i), yy(i)
            write( unit=iprint + 1, fmt='(3f15.4)') xx(i),  fi

        end do

    end if

end function scatpot
```

## 2.4 Electric Potential

The spherically symmetric Poisson equation

$$\frac{1}{r^2}\frac{d}{dr}(r^2(\frac{d\Phi(r)}{dr})) = -4\pi\rho(r)$$

was expanded to a form that can be used in standard O.D.E solvers

$$\frac{2}{r}\Phi^{'} + \Phi^{''} = -4\pi\rho(r)$$

where $\rho(r)$ were the various charge distributions. The Runge-Kutta method was again used to solve this differential equation.

```
    module setup

    use numtype
    implicit none

    integer, parameter :: n_eq = 2


end module setup



program prob4

    use setup
    implicit none

    real(dp), dimension(n_eq) :: phi
    real(dp) :: r, dr, rmax

    r = 0
    dr = 0.01_dp
    rmax = 6._dp


    phi(1) = 0
    phi(2) = 1

    do while ( r < rmax )
```

```
        write(17,*) r, phi(1)
        call rk4step ( r, dr, phi )

    end do

end program prob4


subroutine rk4step ( x, h, y )

    use setup
    implicit none
    real(dp), intent(inout) :: x
    real(dp), intent(in) :: h
    real(dp), intent(inout), dimension(n_eq) :: y
    real(dp), dimension(n_eq) :: k1, k2, k3, k4, dy

    k1 = kv( x, h, y )
    k2 = kv( x+h/2, h, y + k1/2 )
    k3 = kv( x+h/2, h, y + k2/2 )
    k4 = kv( x+h , h,  y + k3 )
    dy = ( k1 + 2*k2 + 2*k3 + k4 ) / 6

    x = x + h
    y = y + dy

    contains

        function kv( x, dx, phi ) result(k)

            use setup
            implicit none
            real(dp), intent(in) :: x, dx
            real(dp), intent(in), dimension(n_eq) :: phi
            real(dp), dimension(n_eq) :: f, k
!-----------------------------------------------------------
            f(1) =  phi(2)
            f(2) =  -2/x*phi(2) - 4*pi*rho(x)
!------------------------------------------------------------
```

```
        k = dx * f

    end function kv

    function rho(r)

        use setup
        real(dp) :: r, rho

        rho = 1._dp/(8._dp*pi)*exp(-r)

    end function rho

end subroutine rk4step
```

## 3 Results

### 3.1 Numerical Integration

Using the Romberg method with ten integration points we calculated to integral to be

$$I = 0.6666607$$

compared to the actual value of $0.\overline{6}$.

### 3.2 One Dimensional Harmonic Oscillator

Using the Runga-Kutta and Chebychev methods the first five wave functions were plotted

### 3.3 Scattering Potential Function

The data was read in and the reflection coefficients were plotted as a function of the energy The curve fit was unsuccessful as the proper relationship between the energy, reflection coefficients and the potential function $V(x)$ was not found. An assumed potential function

$$V(x) = v_0 \frac{1 + tanh(\frac{x}{x_0}}{2}$$

was the fit aim.

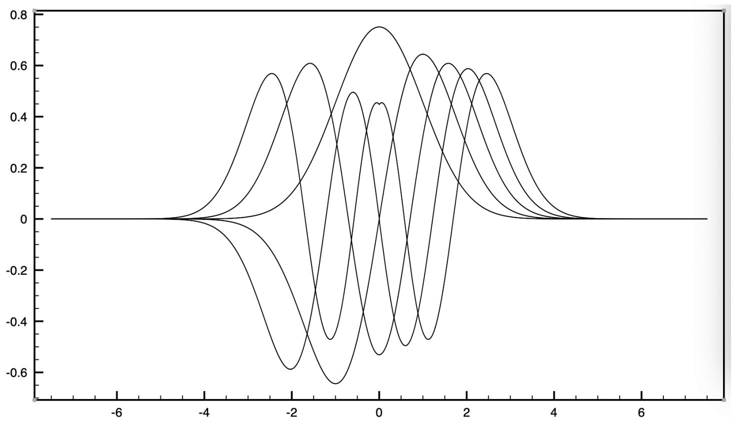

Figure 2: $\psi_n$ 1-D Harmonic Oscillator

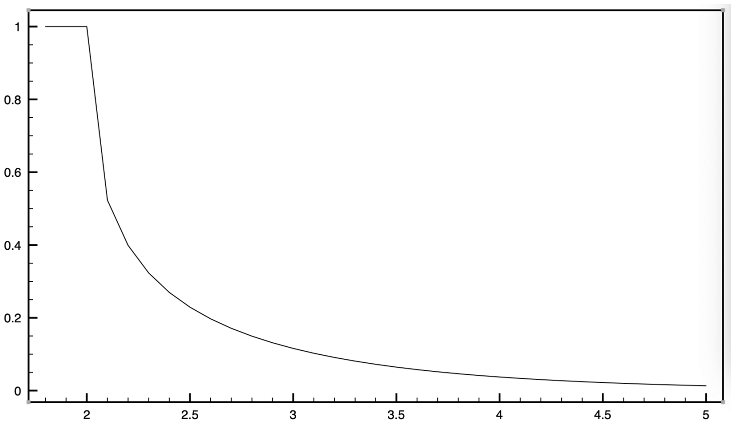

Figure 3: Energy and Reflection Coeeficients

## 3.4 Electric Potential

We were unable to have successful outputs from the differential equation solvers. To work properly we think the reverse Runge-Kutta method needs to be implemented as the potential at $r = 0$ is most likely a singularity. Using a potential of zero and a very small value for the potential derivative at $r = \infty$ as initial conditions.