Physics 562 - Computational Physics

Final

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

May 15, 2014

Abstract

This paper examines three ways of solving for eigenfunctions. The first problem uses Runge-Kutta to solve for the wave functions of a one-dimensional problem with a coulomb potential. The second problem uses matrices to solve for the same problem. The third problem solves a scattering problem.

1 Problem 1

The first system is a quantum one-dimensional problem with $\hbar = 1$ and $mass = 1$. The potential is of the form

$$V(x) = \frac{1}{\sqrt{1+x^2}}. \tag{1}$$

The eigenvalues are calculated by assuming the value of the endpoints and using Runge-Kutta to solve for the wave function. Chebyshev is then used to find the energies where either the wave function or the derivative of the wave function vanish at zero. To find orthogonality, use the equation

$$\int_{-\infty}^{+\infty} \psi_n^* \psi_m \, dx = \delta_{nm}. \tag{2}$$

If $n = m$ integrating over the wave functions will give a value of 1, otherwise integrating will give a value of 0.

1.1 The Fortran95 code

Listing 1: program `ho1d.f95`

```fortran


module setup

    use NumType
    implicit none
    integer, parameter  :: n_eq = 3
    real(dp), parameter :: hbar = 1._dp,hbar2 = hbar**2,&
                           mass = 1._dp, omega = 1._dp, &
                           x0 = sqrt(hbar/(mass*omega))
    real(dp) :: energy, xmax, dx, eps, total,val1,val2
    real(dp), allocatable, dimension(:,:) :: wf, w1
    integer :: imax

end module setup

program ho1d

    use setup
    use chebyshev
    implicit none
    real(dp) :: eminx, emaxx, emin, emax, de, e0, de0
    real(dp), external :: psi0
    integer :: nch, iz, i, maxf, izz,m,n,j,k

    eminx = -1._dp
    emaxx = 0._dp

    xmax = 40.0_dp
    dx = 0.01_dp
    eps = 0.000001_dp
    de0 = 0.01_dp
    maxf = 10
    imax = abs(xmax/dx)
    allocate(wf(0:imax,2))
    allocate(w1(0:2*imax,10))
    de = 0.01_dp
    izz = 0
    nch = 5
    emin = eminx
    emax = emin+de

    print *, '================Eigenvalues==============='

    do while (emin < emaxx)

        call chebyex(psi0, nch, cheb, emin, emax)
        call chebyzero(nch,cheb,emin,emax,z0,iz0)

        do iz=1,iz0
            e0=z0(iz)
            call root_polish(psi0,e0,de0,eps,maxf)
            izz = izz + 1
            print *, izz, e0
            call wavef(e0,izz)
        end do

        emin = emax
        emax = emin + de

    end do

    print *, '============Orthogonality␣Test============'

    total = 0
    val1=0
    val2=0
    j=1
    k=1

    do j = 1,10
    do k = 1,10
    do i = 1,2*imax
        m=j
        n=k
        val1=w1(i,m)*w1(i,n)
        val2=w1(i+1,m)*w1(i+1,n)
        !Rombint/trapazoidal method of integration
        total = total + (0.01*(val1+1/2._dp*(val2-val1)))
    end do

    print '(a,1f5.1,a5,i2,a5,i2)','total=', &
         total,'i=',j,'j=',k

    total = 0

    end do
    end do



end program ho1d

function psi0(eee) result(psi)

    use setup
    implicit none
    real(dp), intent(in) :: eee
    real(dp) :: x, psi
    real(dp), dimension(n_eq) :: y

    energy = eee
    x = xmax
    y(1) = 0.00001
    y(2) = -0.00001
    y(3) = 0._dp
    do while ( x > 0._dp )
        call rk4step(x,y,-dx)
    end do
    psi = y(1)*y(2)

end function psi0

subroutine wavef(eee,iz)

    use setup
    implicit none
```

```fortran
    real(dp), intent(in) :: eee
    real(dp) :: x, parity
    real(dp), dimension(n_eq) :: y
    integer :: iz, i, imin
    energy = eee
    x = xmax
    y(1) = 0.00001
    y(2) = -0.00001
    y(3) = 0._dp
    do while ( x > 0._dp)
        call rk4step(x,y,-dx)
    end do

    x = xmax
    i = imax + 1
    y(1) = 0.00001
    y(2) = -0.00001
    y(3) = 0._dp
    do while ( x > 0._dp)
        i = i-1
        wf(i,1) = x
        wf(i,2) = y(1)
        call rk4step(x,y,-dx)
    end do

    !establish parity so the negative side of
    !graph can be plotted
    imin=i
    if( abs(y(1)) > abs(y(2)) ) then
        parity = 1
    else
        parity = -1
    end if

    ! make a master list of wavefunctions called w1.
    !Used for integration
    w1(0:imax,iz) = parity*wf(0:imax,2)/sqrt(2*y(3))
    w1(imax:2*imax,iz) = wf(0:imax,2)/sqrt(2*y(3))

    !normalize the wavefunction
    wf(0:imax,2) = wf(0:imax,2)/sqrt(2*y(3))


    !make the graphs for positive and negative sides
    do i = imax, imin, -1
        write(unit=20+iz,fmt='(2f15.5)') wf(i,1), &
            wf(i,2)
    end do

    do i = imin, imax
        write(unit=20+iz,fmt='(2f15.5)') -wf(i,1), &
            parity*wf(i,2)
    end do


end subroutine wavef
```

2 Problem 2

The next problem solves the same system as problem 1, but to solve it in the basis of the simple harmonic oscillator.

2.1 The Fortran95 code

Listing 2: `mtest.f95`

```

module setup

    use NumType
    implicit none
    integer,    parameter    :: n_basis=50,&
                     lwork=2*n_basis+1,n_eq=3
    real(dp),   parameter    :: mass=1.0_dp, hbar=1.0_dp,&
                                omega_b=4/5._dp, &
                                hbar2 = hbar**2
    real(dp), allocatable, dimension(:,:) :: wf
    real(dp) :: energy
```

```fortran

end module setup

program matrix

    use setup
    implicit none

    complex(dp) ::  x_mat(0:n_basis,0:n_basis+1), &
                    p_mat(0:n_basis,0:n_basis+1), &
                    x2_mat(0:n_basis,0:n_basis), &
                    p2_mat(0:n_basis,0:n_basis), &
                    h_mat(0:n_basis,0:n_basis), &
                    work(lwork),e(0:n_basis,0:n_basis)
    integer :: n,m, info, i, j,imax, parity, imin, iz
    real(dp) :: rwork(3*(n_basis-2)), w_eigen(n_basis+1)
    real(dp) :: x,xmax,dx, y(n_eq)


    x_mat = 0._dp
    p_mat = 0._dp

    do n=0,n_basis-1
        x_mat(n,n+1) = sqrt(hbar/(2*mass*omega_b))*&
        sqrt(n+1._dp)
        x_mat(n+1,n) = x_mat(n,n+1)
    end do
    x_mat(n_basis,n_basis+1)=sqrt(hbar/(2*mass*omega_b))&
    *sqrt(n_basis+1._dp) !add last point


    do n=0,n_basis-1
        p_mat(n,n+1) = -iic*sqrt(mass*hbar*omega_b/2)&
        *sqrt(n+1._dp)
        p_mat(n+1,n) = -p_mat(n,n+1)
    end do
    p_mat(n_basis,n_basis+1)=-sqrt(mass*hbar*omega_b/2)&
    *sqrt(n_basis+1._dp) !add last point



    h_mat(0:n_basis,0:n_basis) = 1/(2*mass) * &
        matmul(p_mat(0:n_basis,0:n_basis+1),&
        conjg(transpose(p_mat(0:n_basis,0:n_basis+1))))+&
        1/sqrt(1 + &
        matmul(x_mat(0:n_basis,0:n_basis+1),&
        transpose(x_mat(0:n_basis,0:n_basis+1))))


    e(1:n_basis,1:n_basis)=h_mat(1:n_basis,1:n_basis)/10

    call zheev('n','u',n_basis+1,e,n_basis+1,&
    w_eigen,work,lwork,rwork,info)


    print *, '-------␣Orthogonality␣-------␣'

        do j = 1,50
            do i = 1,50
                print *, i, j, dot_product(e(1:50,i) &
                ,e(1:50,j) )
            end do
        end do


    print *, '===First␣Ten␣Energy␣States==='
    do i=1,10
    print *, w_eigen(i)
    end do

    print *, 'First␣ten␣entries␣for␣the␣Ten␣Eigenvectors'
    do i=1,10
    print '(10f8.2)', dble(e(1:10,i))
    end do

    !Plot Results

    xmax=40
    dx=0.01
    imax = abs(xmax/dx)

    allocate(wf(0:imax,2))

    !only plot the first 10 intead of all 50
!     iz = size(w_eigen)
    iz = 10

    !plot the wavefunctions for the different eigenvalues
    do j=1,iz

    energy = w_eigen(j)

    x=xmax

    y(1) = 0.00001
    y(2) = -0.00001
    y(3) = 0

    i = imax+1
    do while(x > 0)
        i = i -1
        wf(i,1) = x
        wf(i,2) = y(1)
        call rk4step(x,y,-dx)
    end do

    imin=i
    if( abs(y(1)) > abs(y(2)) ) then
        parity = 1
    else
        parity = -1
    end if

    wf(0:imax,2) = wf(0:imax,2)/sqrt(2*y(3))


        do i = imax, imin, -1
            write(unit=20+j,fmt='(2f15.5)') wf(i,1), &
                wf(i,2)
        end do

        do i = imin, imax
            write(unit=20+j,fmt='(2f15.5)') -wf(i,1), &
                parity*wf(i,2)
        end do
    end do


end program matrix
```

3 Problem 3

The thirds system solves for a quantum scattering system. The wave function vanishes at zero and as $r \to \infty$ the wave function becomes

$$\psi(r) \sim cos(kr - l\frac{\pi}{2} + \delta_l), \tag{3}$$

where

$$k = \sqrt{\frac{2mE}{\hbar^2}}. \tag{4}$$

The $l = 0$ phase shift are calculated for $E = 1, 5, 10, \& 20$. The bound state energies for $l = 0$ are also computed. This problem required two different codes. One that is run with `ho1d` to calculate the phase shift and one that is run with `ho2d` to calculate the bound state energies.

3.1 The Fortran95 code

Listing 3: `ho1d10.f95`

```


module setup

    use NumType
    implicit none
    integer, parameter :: n_eq = 3
    real(dp), parameter :: hbar2=1._dp, &
```

```fortran
               mass=1.0_dp,  xm=mass*mass/(mass+mass)
    integer :: l

    real(dp) :: energy, xmax, dx, eps, xmid
    real(dp), allocatable, dimension(:,:) :: wf
    integer :: imax

end module setup

program ho1d

    use setup
    use chebyshev
    implicit none
    real(dp) :: dmin,dmax,de,delta0,psi, energy_array(4)
    real(dp), external :: psi0
    integer :: nch, iz, i, maxf

    l=0
    energy_array(1)=1._dp
    energy_array(2)=5._dp
    energy_array(3)=10._dp
    energy_array(4)=20._dp

    xmax=10._dp
    dx=0.001_dp
    eps=0.0000001_dp
    maxf=20
    dmin=0._dp
    dmax=pi

    imax=nint(xmax/dx)+1
    allocate(wf(0:imax,2))

    nch=6

    do i = 1,4
    energy = energy_array(i)
    print *,'for␣Energy=', nint(energy)
    call chebyex(psi0,nch,cheb,dmin,dmax)
    call chebyzero(nch,cheb,dmin,dmax,z0,iz0)

    de=0.01_dp

    do iz=1,iz0
        delta0=z0(iz)
        call root_polish(psi0,delta0,de,eps,maxf)
        psi=psi0(delta0)
        print *, '␣Delta=', delta0
        call wavefunction(delta0,i)
    end do

    end do

end program ho1d

function psi0(delta) result(psi)

    use setup
    implicit none
    real(dp), intent(in) :: delta
    real(dp) :: x, psi, k
    real(dp), dimension(n_eq) :: y

    k = sqrt(2*xm/hbar2*energy)

    x=xmax
    y(1) = sin(k*x - l*pi/2 +delta)
    y(2) = k*cos(k*x -l*pi/2 +delta)
    do while ( x>xmid)
        call rk4step(x,y,-dx)
    end do
    psi=y(2)

end function psi0

subroutine wavefunction(delta,index)

    use setup
    implicit none
```

```fortran
    real(dp), intent(in) :: delta
    integer, intent(in) :: index
    real(dp) :: x, k, y12,y11
    real(dp), dimension(n_eq) :: y
    integer :: n, i

    k=sqrt(2*xm/hbar2*energy)

    x=xmax
    y(1)=sin(k*x-l*pi/2+delta)
    y(2)=k*cos(k*x-l*pi/2+delta)
    do while (x>xmid)
        n=nint(x/dx)
        wf(n,1)=x
        wf(n,2)=y(1)
        call rk4step(x,y,-dx)
    end do
    y12=y(1)

    wf(0,1)=0._dp
    wf(0,2)=0._dp
    x=dx
    y(1)=x**(l+1)
    y(2)=(l+1)*x**l
    do while (x<=xmid)
        n=nint(x/dx)
        wf(n,1)=x
        wf(n,2)=y(1)
        call rk4step(x,y,dx)
    end do
    y11=y(1)

    wf(0:n,2)=y12/y11*wf(0:n,2)
    n=nint(xmax/dx)
    do i = 0,n
        write(10+index,*) wf(i,1),wf(i,2)
    end do

end subroutine wavefunction
```

Listing 4: `ho1d10_energy.f95`

```


module setup

    use NumType
    implicit none
    integer, parameter :: n_eq = 3
    real(dp), parameter :: hbar2=1._dp, &
            mass=1.0_dp, xm=mass*mass/(mass+mass)
    integer :: l

    real(dp) :: energy, xmax, dx, eps, xmid
    real(dp), allocatable, dimension(:,:) :: wf
    integer :: imax

end module setup

program bound

    use setup
    use chebyshev
    implicit none
    real(dp) :: eminx,emin,emaxx,emax,deltae, &
        de,e0,psi
    real(dp), external :: psi0
    integer :: nch, izz, i, maxf, n, nstep

    l=0

    xmax=5.0_dp
    dx=0.001_dp
    eps=0.0001_dp
    maxf=20
    eminx=-100._dp
    emaxx=0._dp
    de=0.2_dp
    nstep=5
    nch=5

```

```fortran
    imax=nint(xmax/dx)+1
    allocate(wf(0:imax,2))

    e0=eminx
    do
        psi=psi0(e0)
        if(psi /= xmax .or. e0 > emaxx) exit
        e0=e0+de
    end do

    eminx=e0
!     print *, eminx
    deltae=(emaxx-eminx)/nstep
!     print *, deltae

    izz = 0

    print *, 'The bound state energy are '
    do n=1,nstep

        emin = eminx+(n-1)*deltae
        emax = eminx+n*deltae
!         print *, emin, emax
        call chebyex(psi0,nch,cheb,emin,emax)
        call chebyzero(nch,cheb,emin,emax,z0,iz0)
!         print *, z0(1:iz0)

        de=0.1_dp
        do i=1,iz0
            e0=z0(i)
            call root_polish(psi0,e0,de,eps,maxf)
            psi=psi0(e0)
            izz=izz+1
            print *, izz,'E=',e0
            call wavef(e0,izz)
        end do

    end do

end program bound
```

```

function psi0(eee) result(psi)

    use setup
    implicit none
    real(dp), intent(in) :: eee
    real(dp) :: x, psi, k
    real(dp), dimension(n_eq) :: y

    energy = eee
    k = sqrt(2*xm/hbar2*(-energy))

    x=dx
    y(1)=x**(l+1)
    y(2)=(l+1)*x**l
    y(3)=0._dp
    do while (x <= xmax .and. y(2) > 0._dp)
        call rk4step(x,y,dx)
    end do

    xmid=x
    if ( xmid >= xmax ) then
        psi=xmax
        return
    end if

    x=xmax
    y(1) = exp(-k*x)
    y(2) = -k*y(1)
    y(3) = 0._dp
    do while (x > xmid)
        call rk4step(x,y,-dx)
    end do
    psi=y(2)

!     print *, e,psi

end function psi0

subroutine wavef(eee,iz)
```

```fortran

    use setup
    implicit none
    real(dp), intent(in) :: eee
    real(dp) :: x, psi,k,y12,y32,y11,y31,yy
    real(dp), dimension(n_eq) :: y
    integer :: iz, n, i

    energy = eee
    k=sqrt(2*xm/hbar2*(-energy))

    x=xmax
    y(1)=exp(-k*x)
    y(2)=-k*y(1)
    y(3) = 0._dp
    do while (x>xmid)
        n=nint(x/dx)
        wf(n,1)=x
        wf(n,2)=y(1)
        call rk4step(x,y,-dx)
    end do
    y12=y(1)
    y32=-y(3)

    wf(0,1)=0._dp
    wf(0,2)=0._dp
    x=dx
    y(1)=x**(l+1)
    y(2)=(l+1)*x**l
    y(3)=0._dp
    do while ( x <= xmid )
        n=nint(x/dx)
        wf(n,1)=x
        wf(n,2)=y(1)
        call rk4step(x,y,dx)
    end do
    y11=y(1)
    y31=y(3)

    wf(0:n,2)=y12/y11*wf(0:n,2)
    y31=(y12/y11)**2* y31
    yy=y31+y32
    n=nint(xmax/dx)
    wf(0:n,2)=wf(0:n,2)/sqrt(yy)
    do i = 0,n
        write(30*(l+1)+iz,*) wf(i,1),-wf(i,2)
    end do

end subroutine wavef
```

4 Results

4.1 Problem 1

The results from problem 1 give the following energies,

$$\begin{aligned}
E_0 &= -0.669777\\
E_1 &= -0.274891\\
E_2 &= -0.151454\\
E_3 &= -0.092679\\
E_4 &= -0.063526\\
E_5 &= -0.045477\\
E_6 &= -0.034374\\
E_7 &= -0.025277\\
E_8 &= -0.016334\\
E_9 &= -0.005216
\end{aligned}$$

Notice that as you go up in energy values, the spacing between energy levels gets smaller. The first five wave functions plotted in Fig **??**. The eigenstates are also orthogonal as the relationship in Eq. **??** holds true.

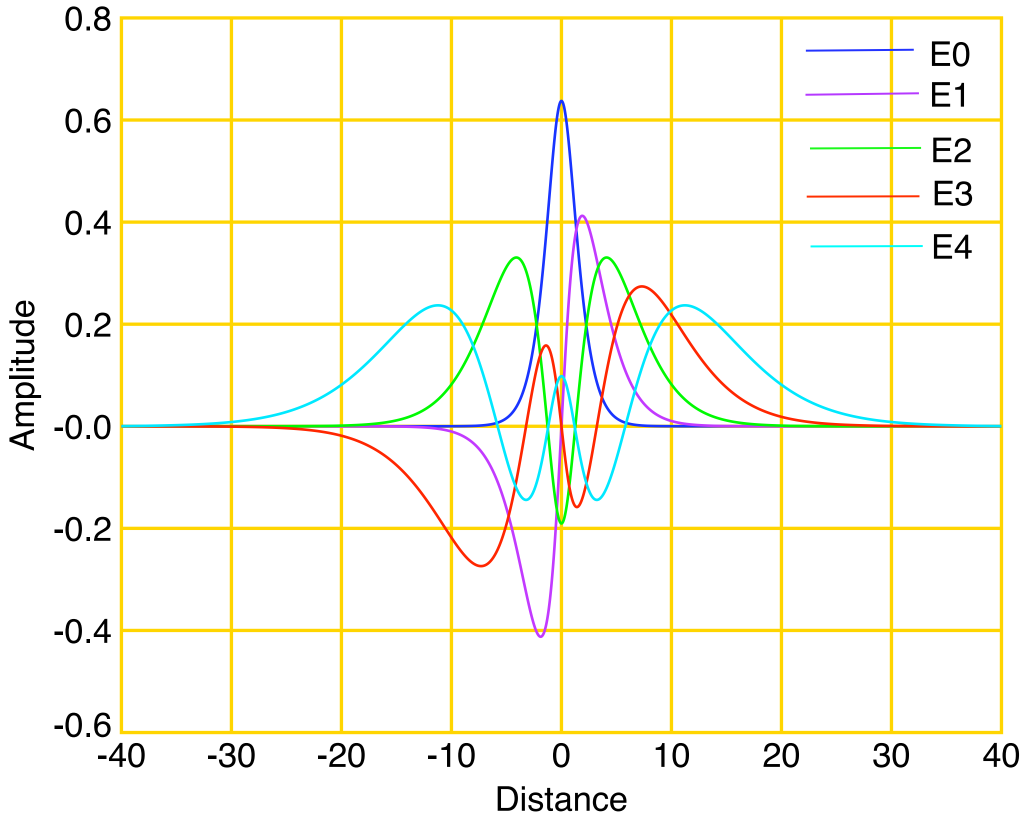

Figure 1: The first five wave functions for problem 1

4.2 Problem 2

The eigenvalues for problem 2 come out to be

$$E_0 = -0.22382$$
$$E_1 = -0.19183$$
$$E_2 = -0.18975$$
$$E_3 = -0.16201$$
$$E_4 = -0.15147$$
$$E_5 = -0.10753$$
$$E_6 = -0.10724$$
$$E_7 = -0.06342$$
$$E_8 = -0.05584$$
$$E_9 = -0.005342$$

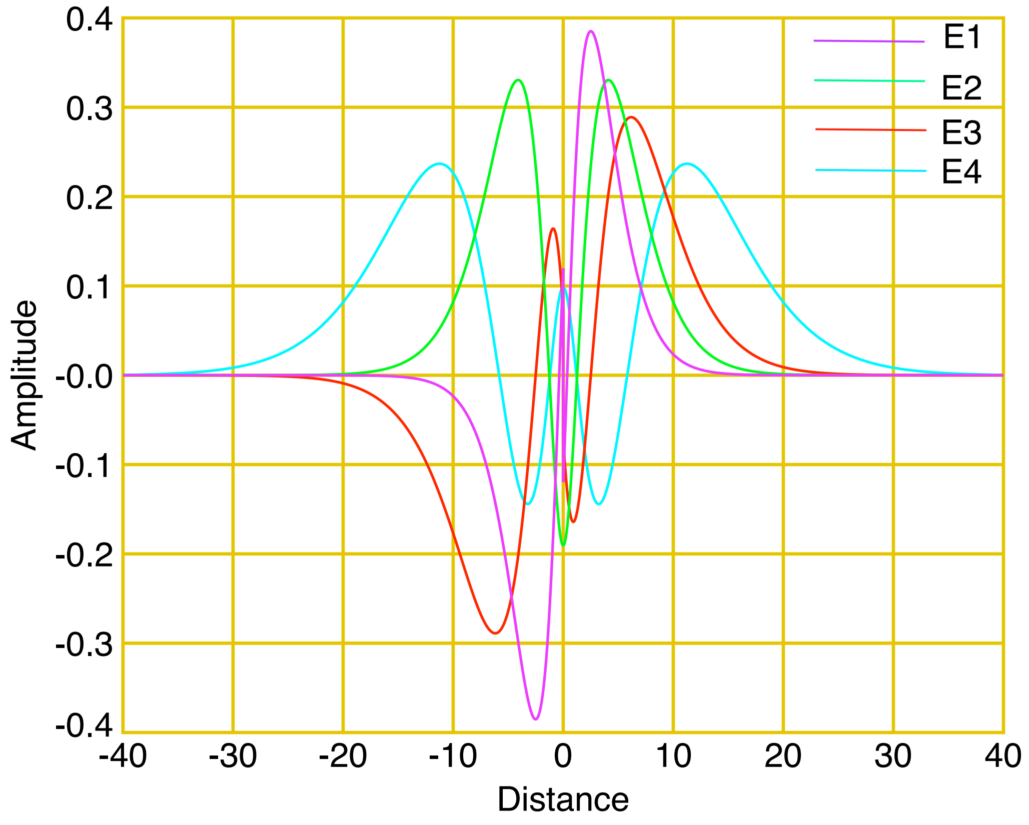

Figure 2: The wave functions for problem 2

Most of these values do not agree with the energies from problem 1. Some of them are close, such as E_9 and E_2, but the others are not. Also, the values are not orthogonal so they are probably not right. Fig. **??** shows four plots generated with the code for problem two. Notice that the wave functions are similar to the ones in Fig. **??**.

4.3 Problem 3

For problem 3, the phase shifts for their respective energies are

$$E = 1, \delta = 3.05219$$
$$E = 5, \delta = 2.55553$$
$$E = 10, \delta = 1.90526$$
$$E = 20, \delta = 1.18828$$

A completely free particle has a phase shift $\delta_l = 0$. The results show that as the energy increases, δ_l approaches zero. The reason for this is that as a particle has a higher energy, the more easily it can escape from a potential well. The scattering wave functions are plotted in Fig. **??**. The bound states for energies for $l = 0$ are

$$E_0 = -26.7309$$
$$E_1 = -13.6180$$
$$E_2 = -3.29810$$

The bound state wave functions are plotted in Fig. **??**

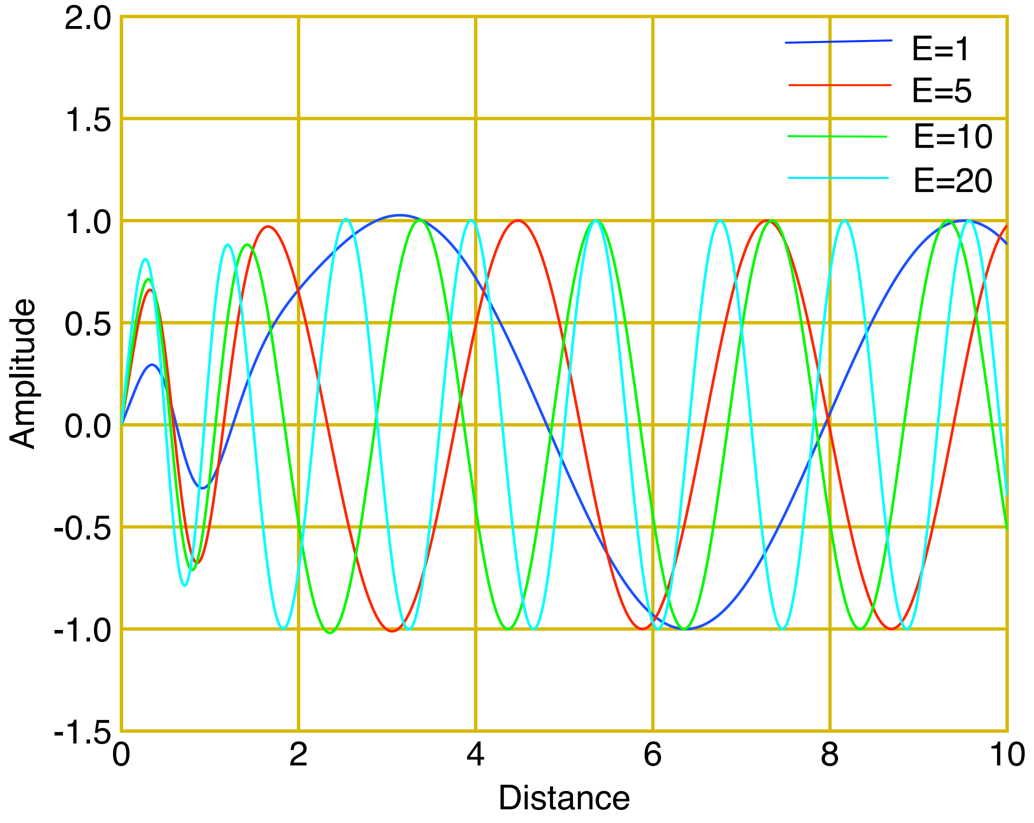

Figure 3: The scattering functions for problem 3

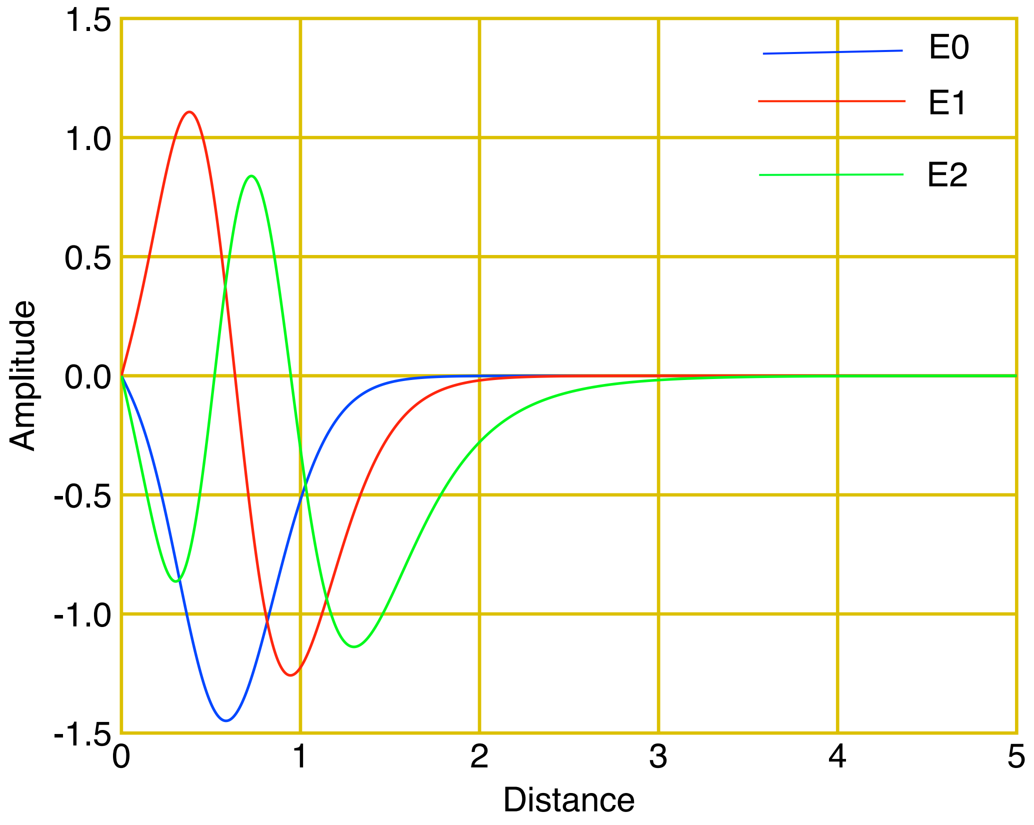

Figure 4: The bound functions for problem 3

References

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