Assignment 1
One-dimensional Schrödinger equation

Due Time : 23:59, March 15, 2020 (Sunday)

Name : 陈 稼 霖
Student ID : 45875852
Score : _____

Problem 1 Score: _____. Suppose there is a single particle of mass m confined to $0 < x < L$ with potential $V = 0$ bounded by infinite high potential barrier, *i.e.*

$$V(x) = 0 \quad \text{where } 0 < x < L, \qquad\qquad V(x) = \infty \quad \text{where } x \geq L; x \leq 0$$

(a) Program Numerov's method for this type of potential. (hint: stat with the 1D harmonic oscillator code and update the potential function.) (**2 points**)

(b) Determine the first three lowest energies and the corresponding wave function. (**1 points**)

(c) Compare your results to the exact solution analytically obtained in quantum mechanics. (**1 points**)

$$E_n = \frac{(n\pi\hbar)^2}{2mL^2} \tag{1}$$

$$\begin{aligned} \Psi_n(x) =& \sqrt{\frac{2}{L}}\sin(n\pi x/L) && 0 < x < L \\ =& 0 && x \leq 0, x \geq L \end{aligned} \tag{2}$$

Solution: (a) 由于势能V在$[0, L]$范围以外为无穷大，因此波函数在$[0, L]$范围以外均为零，而仅在$[0, L]$范围内不为零，且由连续性条件在两个端点$x = 0$和$x = L$处为零. 为用Numerov方法求解在这一势场下粒子的薛定谔方程，只需在原来求解谐振子方程的代码的基础上，将波函数自变量`x`的取值范围由`[-Xmax,Xmax]`改为`[0,L]`，并将`Vpot`删去（因为存储势能函数的`Vpot`在要计算的`[0,L]`范围内为零）. 简单起见，设$\hbar$，L，m在国际单位制下的数值均为1. 计算步骤（步骤后括号内数字为对应代码行号）：

- 给定所要计算的区域范围$[-Xmax, Xmax]$，离散化的数量Nx，以及能量E的尝试值（19–26）；
- 给定开始两点的波函数值作为迭代初始值（44–45）；
- 计算各处的函数值$g(x_i)$和$s(x_i)$，从而用numerov方法迭代得到各点的波函数值（47-55）；
- 归一化计算得到的波函数（58-59）；
- 检查是否$y_N = 0$，若是，则将结果写入文件并退出程序，否则重新选定能量E的值，再次计算（65-110）.

Fortran代码：

```fortran
1  program main
2      ! solve the Schrodinger equation of a particle in 1-dimension infinite square
           potential well with Numerov's method
3
4      implicit none
5      integer, parameter :: dp = selected_real_kind(8)
6      real(dp), parameter :: eps = 1.d-5, dE = 1.d-2
7
8      ! local vars
9      real(dp) :: L = 1.d0
10     integer :: Nx, i
11     real(dp) :: E     ! trial energy
12     real(dp) :: dx, ySquareSum
13     real(dp), allocatable :: x(:), y(:), g(:), f(:)
14     logical :: yNSignChange = .false., yNSign
15     integer :: looptime = 1
16     real(dp) :: Eprev1, Eprev2, yprev1, yprev2
```

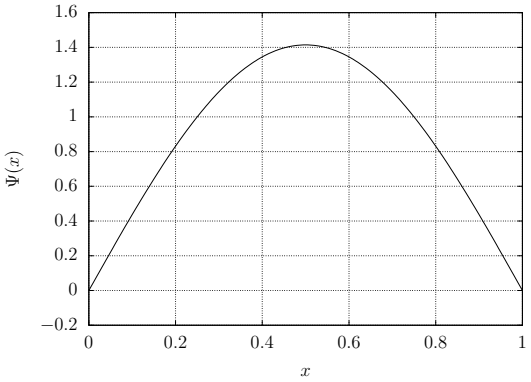
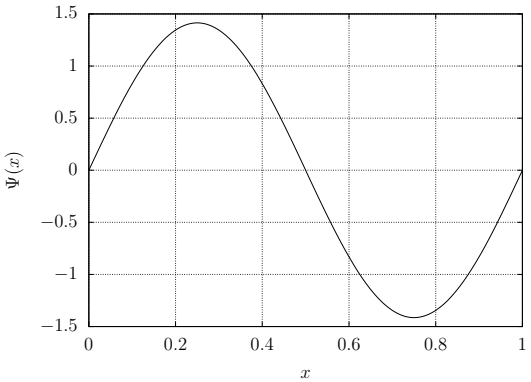
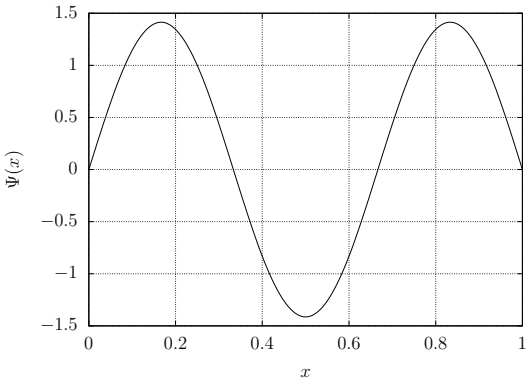
```fortran

    ! executable
    Nx = -1
    do while ((Nx <= 0) .or. (mod(Nx, 2) /= 0))
        write(*,*) 'Please specify the # of slices between [0,L] (must be a even integer), Nx='
        read(*,*) Nx
    end do

    write(*,*) 'Please give a trial energy E0, the first allowed state with E>E0 will be calculated! E0='
    read(*,*) E

    write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'

    ! memory dynamical allocation
    allocate(x(0:Nx))
    allocate(y(0:Nx))
    allocate(g(0:Nx))
    allocate(f(0:Nx))

    dx = L / dble(Nx)
    do i = 0, Nx
        x(i) = dx * i
        ! Vpot(i) = .5d0 * x(i)**2
    end do

    do while (.true.)
        ! set boundary condition: y(0) and y(1)
        y(0) = 0.d0
        y(1) = 1.d-4

        do i = 0, Nx
            g(i) = 2.d0 * E
            f(i) = 1.d0 + g(i) / 12.d0 * dx**2
        end do

        ! Numerov's method
        do i = 1, Nx - 1
            y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i + 1)
        end do

        ! renormalize y(x)
        call Simpson(Nx + 1, y(:), dx, ySquareSum)
```

```
        y = y / sqrt(ySquareSum)

        ! output to screen
        write(*,'(i10,2f20.8)') looptime, E, y(Nx)

        ! change E
        if (abs(y(Nx)) < eps) then    ! if precision requirement is satisfied
            ! output to file
            open(1, file = '1-wavefunction.txt', status = 'unknown')
            do i = 0, Nx
                write(1, '(2f20.8)') x(i), y(i)
            end do
            close(1)
            open(2, file = '1-energy.txt', status = 'unknown')
            write(2, '(f20.8)') E
            close(2)
            exit
        else
            if (.not. yNSignChange) then    ! if the sign of yN has not changed
                E = E + dE
                if (looptime == 1) then    ! if this is the first loop
                    yNSign = (y(Nx) >= 0)
                    yprev1 = y(Nx)
                else
                    if ((y(Nx) >=0) .neqv. (yNSign)) then    ! if the sign of yN changed this time
                        yNSignChange = .true.
                        E = E - dE
                        Eprev1 = E
                        Eprev2 = E - dE
                        E = E - dE / 2.d0
                    end if
                    yprev2 = yprev1
                    yprev1 = y(Nx)
                end if
            else    ! once the sign of yN has changed
                if (abs(yprev1) <= abs(yprev2)) then
                    Eprev2 = E
                    ! Eprev1 = Eprev1
                    E = (Eprev1 + Eprev2) / 2.d0
                    yprev2 = y(Nx)
                    ! yprev1 = yprev1
                else
                    ! Eprev2 = Eprev2
                    Eprev1 = E
```

```
                    E = (Eprev1 + Eprev2) / 2.d0
                    ! yprev2 = yprev2
                    yprev1 = y(Nx)
                end if
            end if
        end if

        looptime = looptime + 1
    end do
end program main

subroutine Simpson(N, y, dx, ySquareSum)
    ! calculate the integral of the square of the wavefunction from -Xmax to Xmax
        with compound simpson formula

    implicit none
    integer :: N
    real(8), intent(in) :: y(N), dx
    real(8), intent(out) :: ySquareSum

    ! local vars
    integer :: i
    real(8) :: Work(N)

    if (mod(N,2) == 0) then
    ! Simpson does not work for integral of array with even number of elements
        write(*, *) 'Array with even elements, Simpson does not know how to work!'
    end if

    do i = 1,N
        Work(i) = y(i)**2
    end do

    ySquareSum = Work(1) + Work(N)
    do i = 2, N - 1, 2
        ySquareSum = ySquareSum + 4.d0 * Work(i)
    end do
    do i = 3, N - 2, 2
        ySquareSum = ySquareSum + 2.d0 * Work(i)
    end do
    ySquareSum = ySquareSum * dx / 3.d0

end subroutine Simpson
```

(b) 用以上代码计算最低的三个能级能量和对应的波函数，结果如表1所示.

表 1: 计算结果：一维无限深方势阱最低的三个能级能量和对应的波函数.

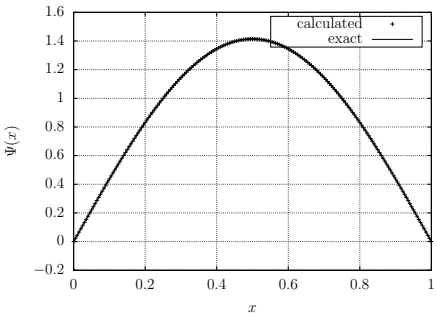
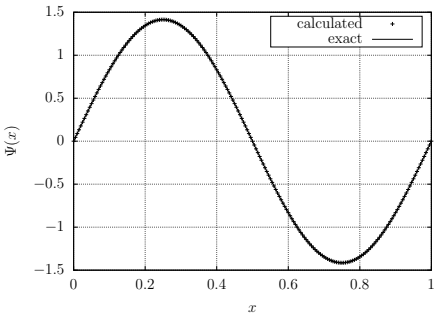
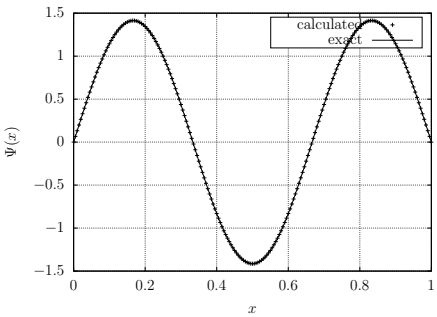
能级序号	能量 (国际单位制)	$[0, L]$范围内波函数图像 （$[0, L]$范围外波函数均为零）
1	4.93480469	
2	19.73921875	
3	44.41328125	

(c) 我们通过一下步骤比较Numerov方法计算结果与其精确值：

- 计算各能级能量的精确值；
- 计算各能级能量计算值相对于其精确值的误差；
- 计算各点的波函数精确值（计算所用节点与(a)中所用节点相同），将波函数计算值和精确解的绘制在同一张图中比较；
- 计算波函数计算值相对于其精确值的误差，这里两条曲线之间的误差定义为各点计算值与当地精确值的欧几里得距离平均值.

比较结果如表2.

表 2: 比较：计算结果和精确值

能级序号 n	能量E_n（国际单位制）			波函数$\Psi_n(x)$	
	计算值	精确值 $\frac{(n\pi\hbar)^2}{2mL^2}$	相对误差	$[0,L]$范围内函数图像（“+”代表计算所得函数值，实线代表精确波函数曲线，$[0,L]$范围外波函数均为零）	误差
1	4.93480469	4.93480220	0.00005%		$3.74129353 \times 10^{-7}$
2	19.73921875	19.73920880	0.00005%		$7.31393035 \times 10^{-7}$
3	44.41328125	44.41321980	0.00014%		$3.00985075 \times 10^{-6}$

无论从误差的数值还是波函数的图像上，均可见Numerov方法计算所得结果十分精确.

□

Problem 2 Score: _____. Prove the energy of the matrix Schrödinger equation on non-orthogonal basis states satisfies the variation principle as well. (**3 points**)

证明. 在一组非正交的基上，系统的波函数可展开为

$$\Psi(x)=\sum_k c_k\Psi_k(x). \tag{3}$$

我们定义两个基矢的内积为

$$\int_{-\infty}^{+\infty}\Psi_k^*(x)\Psi_{k'}(x)\,dx=W_{kk'}. \tag{4}$$

系统的能量可表为

$$E=\frac{\langle\Psi(x)|H|\Psi(x)\rangle}{\langle\Psi(x)|\Psi(x)\rangle}=\frac{\sum_{kk'}c_k^*c_{k'}H_{kk'}}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}. \tag{5}$$

假设波函数展开式中第q项的系数发生微小改变：$c_q\to c_q+\delta_q$，对应地，该系数的复共轭变为：$c_q^*\to c_q^*+\delta_q^*$. 系统能量变为

$$\begin{aligned}
&\text{(分子和分母分别展开并忽略高阶小量)}\\
E'=&\frac{\sum_{kk'}c_k^*c_{k'}H_{kk'}+\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}+\sum_k(c_k^*\delta_qW_{kq}+\delta_q^*c_kW_{qk})}\\
&\text{(分母关于}\delta_q\text{和}\delta_q^*\text{应用泰勒展开，仅保留一阶项)}\\
\approx&\frac{\sum_{kk'}c_k^*c_{k'}H_{kk'}+\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\times\left[1-\frac{\sum_k(\delta_qc_k^*W_{kq}+\delta_q^*c_kW_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\right]\\
=&\left[\frac{\sum_{kk'}c_k^*c_{k'}H_{kk'}}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}+\frac{\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\right]\times\left[1-\frac{\sum_k(\delta_qc_k^*W_{kq}+\delta_q^*c_kW_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\right]\\
&\text{(代入式5)}\\
=&\left[E+\frac{\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\right]\times\left[1-\frac{\sum_k(\delta_qc_k^*W_{kq}+\delta_q^*c_kW_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}\right]\\
&\text{(再次展开，仅保留一阶项)}\\
=&E+\frac{\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}-E\frac{\sum_k(c_k^*\delta_qW_{kq}+\delta_q^*c_kW_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}.
\end{aligned} \tag{6}$$

因为系统能量是一个稳定值，故在发生$c_k\to c_k+\delta_k$的微小变化时，系统能量变化值应为零：

$$E'-E=\frac{\sum_k(c_k^*\delta_qH_{kq}+\delta_q^*c_kH_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}-E\frac{\sum_k(c_k^*\delta_qW_{kq}+\delta_q^*c_kW_{qk})}{\sum_{kk'}c_k^*c_{k'}W_{kk'}}=0, \tag{7}$$

$$\Longrightarrow\sum_k\delta_q^*c_kH_{qk}-E\sum_k\delta_q^*c_kW_{qk}+\text{c.c.}=0, \tag{8}$$

$$\Longrightarrow\sum_k c_kH_{qk}=E\sum_k c_kW_{q,k}. \tag{9}$$

考虑到q取值的任意性，上式对$q=1,2,\cdots,N$都成立：

$$\sum_k c_kH_{qk}=E\sum_k c_kW_{q,k},\quad q=1,2,\cdots. \tag{10}$$

上式可以化为矩阵形式：

$$\begin{pmatrix}H_{11}&H_{12}&\cdots&H_{1N}\\H_{21}&H_{22}&\cdots&H_{2N}\\\vdots&\vdots&\ddots&\vdots\\H_{N1}&H_{N2}&\cdots&H_{NN}\end{pmatrix}\begin{pmatrix}c_1\\c_2\\\vdots\\c_N\end{pmatrix}=E\begin{pmatrix}W_{11}&W_{12}&\cdots&W_{1N}\\W_{21}&W_{22}&\cdots&W_{2N}\\\vdots&\vdots&\ddots&\vdots\\W_{N1}&W_{N2}&\cdots&W_{NN}\end{pmatrix}\begin{pmatrix}c_1\\c_2\\\vdots\\c_N\end{pmatrix}, \tag{11}$$

$$\Longrightarrow H\boldsymbol{c} = EW\boldsymbol{c}. \tag{12}$$

因此由非正交基上的薛定谔矩阵式计算出的能量满足变分原理. □

Problem 3 Score: _____. Complete the skeleton program for the matrix diagonalization of the Schrödinger equation for the following potential function and compare this method to the Numerov's approach in terms of the ground energy accuracy with same $N = 100$ and $x_{\max} = 1.0$, where N is the total number of discretized points in $[0, x_{\max}]$. (**5 points**)

$$V(x) = 10.0 \text{ for } |x| < 0.5 \text{ and } V(x) = \infty \text{ for } |x| \geq 1.0; \text{ otherwise } V(x) = 0.0 \tag{13}$$

Solution: 矩阵对角化方法：计算步骤

- 定义基矢（30–44）；
- 定义在基矢上展开的哈密顿（47–58）；
- 计算哈密顿矩阵的特征值和特征向量（61-64）；
- 输出结果到文件并退出（66–77）.

Fortran代码：

```
1  program main
2      ! solve the Schrodinger equation of a particle in an infinite potential well with
           central barrier with Exact Diagonalization
3
4      implicit none
5      integer, parameter :: dp = selected_real_kind(8)
6      real(dp), parameter :: pi = acos(-1.d0)
7
8      ! local vars
9      integer :: i, j, Nx = 100, LWORK, INFO
10     real(dp) :: Xmax = 1.d0, dx, V0
11     real(dp), allocatable :: x(:), Vpot(:), Ham(:,:), basis(:,:), W(:), WORK(:)
12
13     ! executable
14     ! memory dynamical allocation
15     allocate(x(-Nx:Nx))
16     allocate(Vpot(-Nx:Nx))
17
18     ! discretization and define the potential Vpot
19     dx = Xmax / dble(Nx)
20     do i = -Nx, Nx
21         x(i) = dx * dble(i)
22         if (abs(x(i)) <= .5d0) then
23             Vpot(i) = 10.0d0
24         else
25             Vpot(i) = 0.d0
26         end if
27     end do
```

```fortran

    ! define the basis
    allocate(basis(2 * Nx + 1, 2 * Nx +1))
    basis = 0.d0
    do i = 1, 2 * Nx + 1     ! order of basis function
        if (mod(i, 2) /= 0) then
            ! i is odd
            do j = 1, 2 * Nx + 1     ! coordinate index
                basis(i,j) = sqrt(1.d0 / Xmax) * cos(dble(i) * pi * x(j - Nx - 1) / 2.d0 / Xmax)
            end do
        else
            ! i is even
            do j = 1, 2 * Nx + 1     ! coordinate index
                basis(i,j) = sqrt(1.d0 / Xmax) * sin(dble(i) * pi * x(j - Nx - 1) / 2.d0 / Xmax)
            end do
        end if
    end do

    ! define the Hamiltonian matrix on the basis
    allocate(Ham(2 * Nx + 1, 2 * Nx + 1))
    Ham = 0.d0
    do i = 1, 2 * Nx + 1
        do j = i, 2 * Nx + 1
            call Simpson(2 * Nx + 1, basis(i, 1:2 * Nx + 1), basis(j, 1:2 * Nx + 1), Vpot(-Nx:Nx), dx, V0)
            Ham(i, j) = V0
            if (i == j) then
                Ham(i, j) = Ham(i, i) + (dble(i) * pi)**2 / 8.d0 / Xmax**2
            end if
            Ham(j, i) = Ham(i, j)
        end do
    end do

    ! calculate the eigenvalues of the Hamiltonian
    allocate(W(2 * Nx + 1))
    LWORK = 10 * Nx
    allocate(WORK(LWORK))
    call DSYEV('V', 'U', 2 * Nx + 1, Ham, 2 * Nx + 1, W, WORK, LWORK, INFO)

    if (INFO == 0) then
        open(unit = 1, file = '3-eigenvalue-ExactDiagonalization.txt', status = 'unknown')

```

```
        ! output results
        do i = 1, 2 * Nx + 1
            write(1, '(i4,f20.8)') i, W(i)
        end do
        close(1)
    else
        write(*, *) 'Diagonalization went wrong! aborting ...'
        stop
    end if

    deallocate(Ham)
    deallocate(basis, W, WORK)
    deallocate(x, Vpot)

end program main

subroutine Simpson(N, u, v, Vpot, dx, V0)
    ! calculate the integral of the product of u, v, and Vpot from -Xmax to Xmax with compound simpson formula

    implicit none
    integer :: N
    real(8), intent(in) :: u(N), v(N), Vpot(N), dx
    real(8), intent(out) :: V0

    ! local vars
    integer :: i
    real(8) :: Work(N)

    if (mod(N, 2) == 0) then
    ! Simpson does not work for integral of array with even number of elements
        write(*, *) 'Array with even elements, Simpson does not know how to work!'
    end if

    do i = 1, N
        Work(i) = Vpot(i) * u(i) * v(i)
    end do

    V0 = Work(1) + Work(N)
    do i = 2, N - 1, 2
        V0 = V0 + 4.d0 * Work(i)
    end do
    do i = 3, N - 2, 2
        V0 = V0 + 2.d0 * Work(i)
    end do
```

```
    V0 = V0 * dx / 3.d0

end subroutine Simpson
```

Numerov方法：计算步骤类似Problem 1. Fortran代码：

```
program main
    ! solve the Schrodinger equation of a particle in 1-dimension infinite square
        potential well with Numerov's method

    implicit none
    integer, parameter :: dp = selected_real_kind(8)
    real(dp), parameter :: eps = 1.d-5, dE = 1.d-2

    ! local vars
    real(dp) :: L = 1.d0
    integer :: Nx = 100, i, n = 1, nmax = 20
    real(dp) :: E = 7.d0    ! trial energy, lower than the first energy level
    real(dp) :: dx, ySquareSum
    real(dp), allocatable :: x(:), Vpot(:), y(:), g(:), f(:)
    logical :: yNSignChange, yNSign
    integer :: looptime
    real(dp) :: Eprev1, Eprev2, yprev1, yprev2

    ! executable
    ! write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'

    ! memory dynamical allocation
    allocate(x(-Nx:Nx))
    allocate(Vpot(-Nx:Nx))
    allocate(y(-Nx:Nx))
    allocate(g(-Nx:Nx))
    allocate(f(-Nx:Nx))

    ! discretization and define the potential Vpot
    dx = L / dble(Nx)
    do i = -Nx, Nx
        x(i) = dx * i
        if (abs(x(i)) <= 0.5) then
            Vpot(i) = 10.d0
        else
            Vpot(i) = 0.d0
        end if
    end do

    open(2, file = '3-energy-Numerov.txt', status = 'unknown')
```

```
    do while (n <= nmax)
        looptime = 1
        yNSignChange = .false.
        do while (.true.)
            ! set boundary condition: y(-Nx) and y(-Nx+1)
            y(-Nx) = 0.d0
            y(-Nx + 1) = 1.d-4

            do i = -Nx, Nx
                g(i) = 2.d0 * (E - Vpot(i))
                f(i) = 1.d0 + g(i) / 12.d0 * dx**2
            end do

            ! Numerov's method
            do i = -Nx + 1, Nx - 1
                y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i
                    + 1)
            end do

            ! renormalize y(x)
            call Simpson(2 * Nx + 1, y(:), dx, ySquareSum)
            y = y / sqrt(ySquareSum)

            ! change E
            if (abs(y(Nx)) < eps) then     ! if precision requirement is satisfied
                write(2, '(i4,f20.8)') n, E
                exit
            else
                if (.not. yNSignChange) then     ! if the sign of yN has not changed
                    E = E + dE
                    if (looptime == 1) then     ! if this is the first loop
                        yNSign = (y(Nx) >= 0)
                        yprev1 = y(Nx)
                    else
                        if ((y(Nx) >=0) .neqv. (yNSign)) then     ! if the sign of yN
                            changed this time
                            yNSignChange = .true.
                            E = E - dE
                            Eprev1 = E
                            Eprev2 = E - dE
                            E = E - dE / 2.d0
                        end if
                        yprev2 = yprev1
                        yprev1 = y(Nx)
                    end if
```

```
                else    ! once the sign of yN has changed
                    if (abs(yprev1) <= abs(yprev2)) then
                        Eprev2 = E
                        ! Eprev1 = Eprev1
                        E = (Eprev1 + Eprev2) / 2.d0
                        yprev2 = y(Nx)
                        ! yprev1 = yprev1
                    else
                        ! Eprev2 = Eprev2
                        Eprev1 = E
                        E = (Eprev1 + Eprev2) / 2.d0
                        ! yprev2 = yprev2
                        yprev1 = y(Nx)
                    end if
                end if
            end if
            looptime = looptime + 1
        end do
        E = E + 2 * dE
        n = n + 1
    end do
    close(2)
end program main

subroutine Simpson(N, y, dx, ySquareSum)
    ! calculate the integral of the square of the wavefunction from -Xmax to Xmax with compound simpson formula

    implicit none
    integer :: N
    real(8), intent(in) :: y(N), dx
    real(8), intent(out) :: ySquareSum

    ! local vars
    integer :: i
    real(8) :: Work(N)

    if (mod(N,2) == 0) then
    ! Simpson does not work for integral of array with even number of elements
        write(*, *) 'Array with even elements, Simpson does not know how to work!'
    end if

    do i = 1,N
        Work(i) = y(i)**2
    end do
```

```
    ySquareSum = Work(1) + Work(N)
    do i = 2, N - 1, 2
        ySquareSum = ySquareSum + 4.d0 * Work(i)
    end do
    do i = 3, N - 2, 2
        ySquareSum = ySquareSum + 2.d0 * Work(i)
    end do
    ySquareSum = ySquareSum * dx / 3.d0

end subroutine Simpson
```

两种方法计算得到的能量比较如表3.

表 3: 比较：矩阵对角化方法和Numerov方法计算结果

能级序号	能级能量		绝对误差	相对误差
	矩阵对角化方法计算结果	Numerov方法计算结果		
1	7.81312095	7.83992187	−0.02680092	−0.3%
2	8.81009856	8.84062500	−0.03052644	−0.3%
3	16.18368246	16.18597656	−0.00229410	−0.01%
4	25.60784699	25.61183594	−0.00398895	−0.02%
5	36.60173045	36.62683594	−0.02510549	−0.07%
6	49.34665885	49.37871094	−0.03205209	−0.06%
7	65.16362139	65.17472656	−0.01110517	−0.02%
8	84.19210278	84.19269531	−0.00059253	−0.0007%
9	105.36771842	105.38785156	−0.02013314	−0.02%
10	128.38595112	128.41910156	−0.03315044	−0.03%
⋮	⋮	⋮	⋮	⋮
91	10221.18193839	10031.67441424	189.50752415	2%
92	10447.04544281	10244.33566424	202.70977857	2%
93	10675.46551869	10458.81816424	216.64735445	2%
94	10906.03738031	10674.97566425	231.06171606	2%
95	11138.96335161	10892.68816425	246.27518746	2%
96	11374.79087745	11112.03066426	262.76021319	2%
97	11613.27318336	11333.05316426	280.22001910	2%
98	11853.50482697	11555.60941427	297.89541270	3%
99	12095.48511091	11779.57066427	315.91444664	3%
100	12343.67260074	12005.00066428	338.67193646	3%

从表3可见，对于较低能级的能量，矩阵对角化方法的计算结果与Numerov方法十分接近，而对于高能级的能量，矩阵对角化方法的计算结果与Numerov方法偏差较大. 因此会有较大误差. □

Problem 4 Score: _____. Double well, quantum tunneling, instantons.

Consider a double well system with the following Hamiltonian,

$$H = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{m\omega_0^2}{2}\left[\frac{(x^2 - x_0^2)^2}{4x_0^2} - x^2\right]. \tag{14}$$

Try to determine the ground state wave function and energy of this system by using both Numerov and matrix diagonalization approaches. (Hint: for the latter, one can take eigenstates of the harmonic oscillator as a basis function) (**8 points**)

Solution: 双势阱的势场为

$$V(x) = \frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2} - x^2\right]. \tag{15}$$

方便起见，设$\hbar$，m，ω_0，x_0的数值均为1. 双势阱的势场如图1. 由于势能在$x=\pm 5$处已经比中央的隆起高度高出$1-2$个数量级，所以选取的模拟的范围为$[-5,5]$.

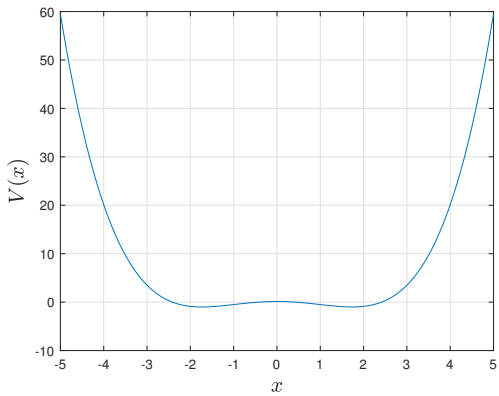


图 1: 双势阱的势场.

Numerov方法：Fortran代码：

```fortran
program main
    ! solve the Schrodinger equation of a particle in 1-dimension infinite square potential well with Numerov's method

    implicit none
    integer, parameter :: dp = selected_real_kind(8)
    real(dp), parameter :: eps = 1.d-5, dE = 1.d-2

    ! local vars
    real(dp) :: Xmax
    integer :: Nx, i
    real(dp) :: E    ! trial energy
    real(dp) :: dx, ySquareSum
    real(dp), allocatable :: x(:), Vpot(:), y(:), g(:), f(:)
    logical :: yNSignChange = .false., yNSign
    integer :: looptime = 1
    real(dp) :: Eprev1, Eprev2, yprev1, yprev2

    ! executable
    Xmax = -1.d0
    do while (Xmax < 0.d0)
        write(*,*) 'Please indicate the range of x, [-Xmax,Xmax]'
```

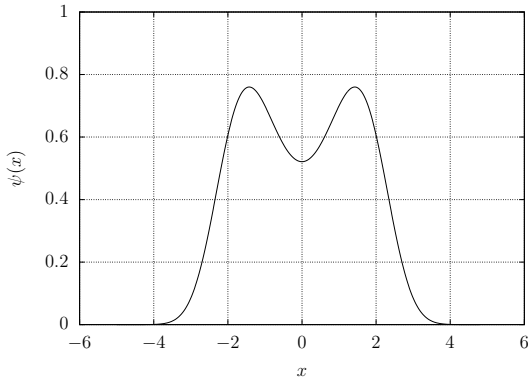
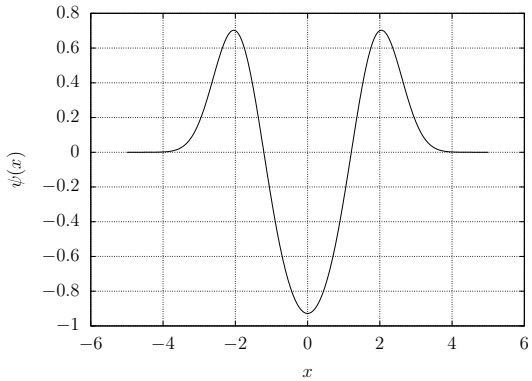
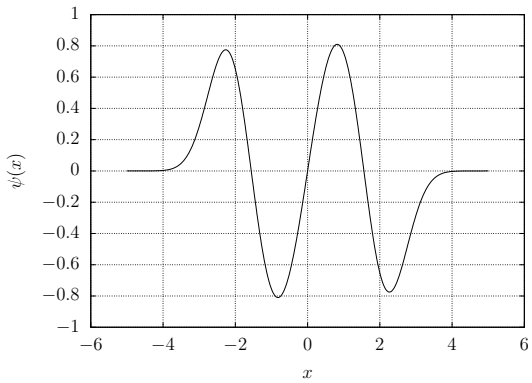
```fortran
        read(*,*) Xmax
    end do

    Nx = -1
    do while ((Nx <= 0) .or. (mod(Nx, 2) /= 0))
        write(*,*) 'Please specify the # of slices between [0,Xmax], Nx='
        read(*,*) Nx
    end do

    write(*,*) 'Please give a trial energy E0, the first allowed state with E>E0 will be calculated! E0='
    read(*,*) E

    write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'

    ! memory dynamical allocation
    allocate(x(-Nx:Nx))
    allocate(Vpot(-Nx:Nx))
    allocate(y(-Nx:Nx))
    allocate(g(-Nx:Nx))
    allocate(f(-Nx:Nx))

    dx = Xmax / dble(Nx)
    do i = -Nx, Nx
        x(i) = dx * i
        Vpot(i) = .5d0 * ((x(i)**2 - 1.d0)**2 / 4.d0 - x(i)**2)
    end do

    do while (.true.)
        ! set boundary condition: y(0) and y(1)
        y(-Nx) = 0.d0
        y(-Nx + 1) = 1.d-4

        do i = -Nx, Nx
            g(i) = 2.d0 * (E - Vpot(i))
            f(i) = 1.d0 + g(i) / 12.d0 * dx**2
        end do

        ! Numerov's method
        do i = -Nx + 1, Nx - 1
            y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i + 1)
        end do

        ! renormalize y(x)
        call Simpson(Nx + 1, y(:), dx, ySquareSum)
```

```
        y = y / sqrt(ySquareSum)

        ! output to screen
        write(*,'(i10,2f20.8)') looptime, E, y(Nx)

        ! change E
        if (abs(y(Nx)) < eps) then     ! if precision requirement is satisfied
            ! output to file
            open(1, file = '4-wavefunction-Numerov.txt', status = 'unknown')
            do i = -Nx, Nx
                write(1, '(2f20.8)') x(i), y(i)
            end do
            close(1)
            open(2, file = '4-energy-Numerov.txt', status = 'unknown')
            write(2, '(f20.8)') E
            close(2)
            exit
        else
            if (.not. yNSignChange) then     ! if the sign of yN has not changed
                E = E + dE
                if (looptime == 1) then     ! if this is the first loop
                    yNSign = (y(Nx) >= 0)
                    yprev1 = y(Nx)
                else
                    if ((y(Nx) >=0) .neqv. (yNSign)) then     ! if the sign of yN changed this time
                        yNSignChange = .true.
                        E = E - dE
                        Eprev1 = E
                        Eprev2 = E - dE
                        E = E - dE / 2.d0
                    end if
                    yprev2 = yprev1
                    yprev1 = y(Nx)
                end if
            else     ! once the sign of yN has changed
                if (abs(yprev1) <= abs(yprev2)) then
                    Eprev2 = E
                    ! Eprev1 = Eprev1
                    E = (Eprev1 + Eprev2) / 2.d0
                    yprev2 = y(Nx)
                    ! yprev1 = yprev1
                else
                    ! Eprev2 = Eprev2
                    Eprev1 = E
```

```
                    E = (Eprev1 + Eprev2) / 2.d0
                    ! yprev2 = yprev2
                    yprev1 = y(Nx)
                end if
            end if
        end if

        looptime = looptime + 1
    end do
end program main

subroutine Simpson(N, y, dx, ySquareSum)
    ! calculate the integral of the square of the wavefunction from -Xmax to Xmax with compound simpson formula

    implicit none
    integer :: N
    real(8), intent(in) :: y(N), dx
    real(8), intent(out) :: ySquareSum

    ! local vars
    integer :: i
    real(8) :: Work(N)

    if (mod(N,2) == 0) then
    ! Simpson does not work for integral of array with even number of elements
        write(*, *) 'Array with even elements, Simpson does not know how to work!'
    end if

    do i = 1,N
        Work(i) = y(i)**2
    end do

    ySquareSum = Work(1) + Work(N)
    do i = 2, N - 1, 2
        ySquareSum = ySquareSum + 4.d0 * Work(i)
    end do
    do i = 3, N - 2, 2
        ySquareSum = ySquareSum + 2.d0 * Work(i)
    end do
    ySquareSum = ySquareSum * dx / 3.d0

end subroutine Simpson
```

部分计算结果如表.

表 4: Numerov方法计算结果：双势阱最低的三个能级

能量	波函数图像
-0.32227314	
0.83123120	
1.71356141	

矩阵对角化方法：以频率为ω_0的量子谐振子的特征波函数

$$\psi_n(x) = H_n\left(\sqrt{\frac{m\omega_0}{\hbar}}x\right)e^{-\frac{m\omega_0}{2\hbar}x^2} \tag{16}$$

为基矢，其中厄米多项式

$$H_n(\xi) = \sum_{m=0}^{[n/2]}\frac{(-1)^m n!}{m!(n-2m)!}(2\xi)^{n-2m}. \tag{17}$$

我们知道，对于量子谐振子来说，它的特征波函数满足的薛定谔方程为

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi_n(x) + \frac{1}{2}m\omega_0^2x^2\psi_n(x) = E_n\psi(x), \quad E = \hbar\omega_0\left(n+\frac{1}{2}\right). \tag{18}$$

因此将双势阱中粒子的哈密顿在量子谐振子的本征波函数上展开，我们会得到

$$H_{nn'} = \langle\psi_n^*(x)|H|\psi_{n'}^*(x)\rangle$$

$$
\begin{aligned}
&=\langle\psi_n(x)|\left\{-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+\frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2}-x^2\right]\right\}|\psi_{n'}(x)\rangle\\
&=\langle\psi_n(x)|\left\{\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+\frac{1}{2}m\omega_0^2x^2\right)+\frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2}-2x^2\right]\right\}|\psi_{n'}(x)\rangle\\
&=\langle\psi_n(x)|\left[\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+\frac{1}{2}m\omega_0^2x^2\right)\right]|\psi_{n'}(x)\rangle+\langle\psi_n(x)|\frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2}-2x^2\right]|\psi_{n'}(x)\rangle\\
&=\hbar\omega_0\left(n+\frac{1}{2}\right)\delta_{nn'}+\int_{-\infty}^{+\infty}\psi_n^*(x)\frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2}-2x^2\right]\psi_{n'}(x)\,dx\\
&\approx\hbar\omega_0\left(n+\frac{1}{2}\right)\delta_{nn'}+\int_{-Xmax}^{Xmax}\psi_n^*(x)\frac{m\omega_0^2}{2}\left[\frac{(x^2-x_0^2)^2}{4x_0^2}-2x^2\right]\psi_{n'}(x)\,dx
\end{aligned}
\tag{19}
$$

综上，我们只在原有矩阵对角化代码的基础上，~~按照式(16)修改基矢组成的矩阵并~~按照式(19)修改哈密顿矩阵即可. 但是由于厄米多项式中存在阶乘，当n或m过大时，会存在溢出的问题，所以先用矩阵对角化方法计算谐振子波函数的近似值来作为基矢，然后再次运用矩阵对角化方法求解双势阱的问题. 代码Fortran：

```fortran
program main
    ! solve the Schrodinger equation of a particle in an infinite potential well with central barrier with Exact Diagonalization

    implicit none
    integer, parameter :: dp = selected_real_kind(8)
    real(dp), parameter :: pi = acos(-1.d0)

    ! local vars
    integer :: i, j, Nx, LWORK, INFO
    real(dp) :: Xmax, dx, V0
    real(dp), allocatable :: x(:), Vpot(:), Ham(:,:), basis(:,:), W(:), WORK(:)

    integer :: LWORK1, INFO1
    real(dp), allocatable :: x1(:), Vpot1(:), Ham1(:,:), W1(:), WORK1(:)

    ! executable
    Xmax = -1.d0
    do while (Xmax < 0.d0)
        write(*,*) 'Please indicate the range of x, [-Xmax,Xmax]; Xmax='
        read(*,*) Xmax
    end do

    Nx = -1
    do while (Nx <= 0)
        write(*,*) 'Please specify the # of slices between [0,Xmax], Nx='
        read(*,*) Nx
    end do

    ! memory dynamical allocation
    allocate(x(-Nx:Nx))
    allocate(Vpot(-Nx:Nx))
```

```
    allocate(x1(-Nx - 1:Nx + 1))
    allocate(Vpot1(-Nx - 1:Nx + 1))

    dx = Xmax / dble(Nx)
    do i = -Nx, Nx
        x(i) = dx * dble(i)
        Vpot(i) = .5d0 * ((x(i)**2 - 1.d0)**2 / 4.d0 - x(i)**2)
    end do

    ! calculate the wavefunction of 1D HO
    do i = -Nx - 1, Nx + 1
        x1(i) = dx * dble(i)
        Vpot1(i) = .5d0 * x1(i)**2
    end do

    allocate(Ham1(2 * Nx + 1, 2 * Nx + 1))
    Ham1 = 0.d0
    do i = 1, 2 * Nx + 1
        Ham1(i,i) = Vpot1(i - Nx) + 1.d0 / dx**2
        if (i > i) Ham1(i,i - 1) = -.5d0 / dx**2
        if (i < 2 * Nx + 1) Ham1(i,i + 1) = -.5d0 / dx**2
    end do

    allocate(W1(2 * Nx + 1))
    LWORK1 = 10 * (Nx + 1)
    allocate(WORK1(LWORK1))
    call DSYEV('V','U',2 * Nx + 1, Ham1, 2 * Nx + 1, W1, WORK1, LWORK1, INFO1)

    allocate(basis(2 * Nx + 1, 2 * Nx +1))
    if (INFO1 == 0) then
        do i = 1, 2 * Nx + 1
            do j = 1, 2 * Nx + 1
                basis(i,j) = Ham1(j,i)
            end do
        end do
    else
        write(*, *) 'Diagonalization went wrong! aborting ...'
        stop
    end if

    allocate(Ham(2 * Nx + 1, 2 * Nx + 1))
    Ham = 0.d0
    do i = 1, 2 * Nx + 1
        do j = i, 2 * Nx + 1
            call Simpson(2 * Nx + 1, basis(i, 1:2 * Nx + 1), basis(j, 1:2 * Nx + 1),
```

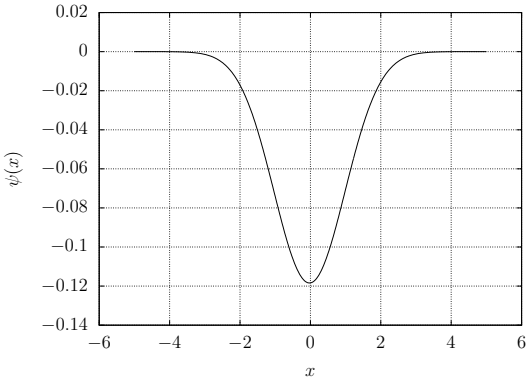
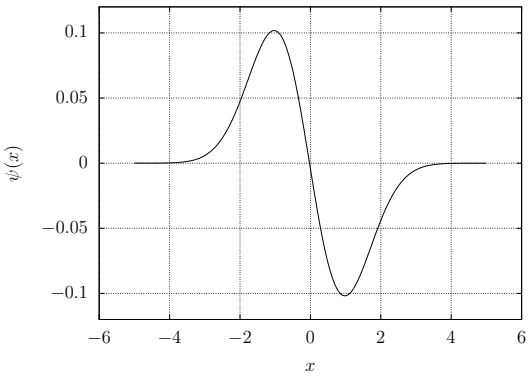
```
                Vpot(-Nx:Nx) - .5d0, dx, V0)
            Ham(i, j) = V0
            if (i == j) then
                Ham(i, j) = Ham(i, j) + (i - .5d0)
            end if
            Ham(j, i) = Ham(i, j)
        end do
    end do

    allocate(W(2 * Nx + 1))
    LWORK = 10 * Nx
    allocate(WORK(LWORK))
    call DSYEV('V', 'U', 2 * Nx + 1, Ham, 2 * Nx + 1, W, WORK, LWORK, INFO)

    if (INFO == 0) then
        open(unit = 1, file = '4-energy-ExactDiagonalization.txt', status = 'unknown')
        open(unit = 2, file = '4-wavefunction-ExactDiagonalization.txt', status = 'unknown')

        ! multiply the coefficient (which is stored in Ham now) with the basis function to get the wave function
        Ham = matmul(transpose(basis), Ham)

        ! output results
        do i = 1, 2 * Nx + 1
            write(1, '(i4,f20.12)') i, W(i)
            write(2, '(f12.8)', advance = 'no') x(i - Nx -1)
            do j = 1, 2 * Nx + 1
                write(2, '(f12.6)', advance = 'no') Ham(i,j)
            end do
            write(2, *)
        end do
        close(1)
        close(2)
    else
        write(*, *) 'Diagonalization went wrong! aborting ...'
        stop
    end if

    deallocate(Ham)
    deallocate(basis, W, WORK)
    deallocate(x, Vpot)

end program main

```

```fortran
subroutine Simpson(N, u, v, Vpot, dx, V0)
    ! calculate the integral of the product of u, v, and Vpot from -Xmax to Xmax with compound simpson formula

    implicit none
    integer :: N
    real(8), intent(in) :: u(N), v(N), Vpot(N), dx
    real(8), intent(out) :: V0

    ! local vars
    integer :: i
    real(8) :: Work(N)

    if (mod(N, 2) == 0) then
    ! Simpson does not work for integral of array with even number of elements
        write(*, *) 'Array with even elements, Simpson does not know how to work!'
    end if

    do i = 1, N
        Work(i) = Vpot(i) * u(i) * v(i)
    end do

    V0 = Work(1) + Work(N)
    do i = 2, N - 1, 2
        V0 = V0 + 4.d0 * Work(i)
    end do
    do i = 3, N - 2, 2
        V0 = V0 + 2.d0 * Work(i)
    end do
    V0 = V0 * dx / 3.d0

end subroutine Simpson
```

部分计算结果如表5.

双势阱的各能级能量与波函数与量子谐振子的十分相近，即使能量低于中央凸起，粒子依然可以越过中央凸起，体现出量子隧穿的效应. □

表 5: 矩阵对角化方法计算结果：双势阱最低的三个能级

能量	波函数图像
0.483562389916	
1.474199856714	
2.474189011020