

**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.*

$$V(x) = 0 \quad \text{where } 0 < x < L,$$

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

- (a) Program Numerov's method for this type of potential. (hint: start 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)**

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

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

**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 $[-X_{\max}, X_{\max}]$ 改为 $[0, L]$ , 并将 $V_{\text{pot}}$ 删去 (因为存储势能函数的 $V_{\text{pot}}$ 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 $[-X_{\max}, X_{\max}]$ , 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:

```
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
```

```

17
18      ! executable
19      Nx = -1
20      do while ((Nx <= 0) .or. (mod(Nx, 2) /= 0))
21          write(*,*) 'Please specify the # of slices between [0,L] (must be a even
22                      integer), Nx='
23          read(*,*) Nx
24      end do
25
26      write(*,*) 'Please give a trial energy E0, the first allowed state with E>E0
27                  will be calculated! E0='
28      read(*,*) E
29
30      write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'
31
32      ! memory dynamical allocation
33      allocate(x(0:Nx))
34      allocate(y(0:Nx))
35      allocate(g(0:Nx))
36      allocate(f(0:Nx))
37
38      dx = L / dble(Nx)
39      do i = 0, Nx
40          x(i) = dx * i
41          ! Vpot(i) = .5d0 * x(i)**2
42      end do
43
44      do while (.true.)
45          ! set boundary condition: y(0) and y(1)
46          y(0) = 0.d0
47          y(1) = 1.d-4
48
49          do i = 0, Nx
50              g(i) = 2.d0 * E
51              f(i) = 1.d0 + g(i) / 12.d0 * dx**2
52          end do
53
54          ! Numerov's method
55          do i = 1, Nx - 1
56              y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i
57                          + 1)
58          end do
59
60          ! renormalize y(x)
61          call Simpson(Nx + 1, y(:), dx, ySquareSum)

```

```

59      y = y / sqrt(ySquareSum)
60
61      ! output to screen
62      write(*, '(i10,2f20.8)') looptime, E, y(Nx)
63
64      ! change E
65      if (abs(y(Nx)) < eps) then      ! if precision requirement is satisfied
66          ! output to file
67          open(1, file = '1-wavefunction.txt', status = 'unknown')
68          do i = 0, Nx
69              write(1, '(2f20.8)') x(i), y(i)
70          end do
71          close(1)
72          open(2, file = '1-energy.txt', status = 'unknown')
73          write(2, '(f20.8)') E
74          close(2)
75          exit
76      else
77          if (.not. yNSignChange) then      ! if the sign of yN has not changed
78              E = E + dE
79              if (looptime == 1) then      ! if this is the first loop
80                  yNSign = (y(Nx) >= 0)
81                  yprev1 = y(Nx)
82              else
83                  if ((y(Nx) >= 0) .neqv. (yNSign)) then      ! if the sign of yN
84                      changed this time
85                      yNSignChange = .true.
86                      E = E - dE
87                      Eprev1 = E
88                      Eprev2 = E - dE
89                      E = E - dE / 2.d0
90                  end if
91                  yprev2 = yprev1
92                  yprev1 = y(Nx)
93              end if
94          else      ! once the sign of yN has changed
95              if (abs(yprev1) <= abs(yprev2)) then
96                  Eprev2 = E
97                  ! Eprev1 = Eprev1
98                  E = (Eprev1 + Eprev2) / 2.d0
99                  yprev2 = y(Nx)
100                  ! yprev1 = yprev1
101              else
102                  ! Eprev2 = Eprev2
103                  Eprev1 = E

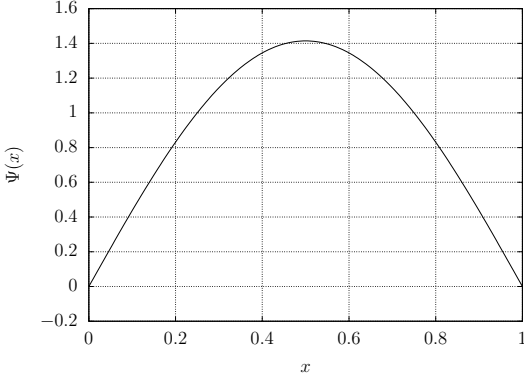
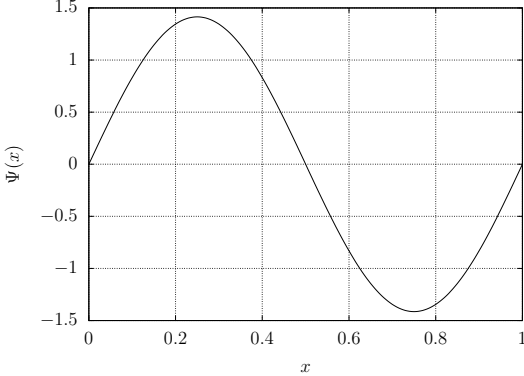
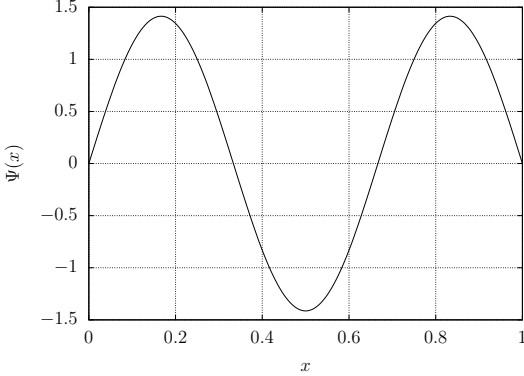
```

```

103             E = (Eprev1 + Eprev2) / 2.d0
104             ! yprev2 = yprev2
105             yprev1 = y(Nx)
106         end if
107     end if
108 end if
109
110     looptime = looptime + 1
111 end do
112 end program main
113
114 subroutine Simpson(N, y, dx, ySquareSum)
115     ! calculate the integral of the square of the wavefunction from -Xmax to Xmax
116     ! with compound simpson formula
117
118     implicit none
119     integer :: N
120     real(8), intent(in) :: y(N), dx
121     real(8), intent(out) :: ySquareSum
122
123     ! local vars
124     integer :: i
125     real(8) :: Work(N)
126
127     if (mod(N,2) == 0) then
128         ! Simpson does not work for integral of array with even number of elements
129         write(*, *) 'Array with even elements, Simpson does not know how to work!'
130     end if
131
132     do i = 1, N
133         Work(i) = y(i)**2
134     end do
135
136     ySquareSum = Work(1) + Work(N)
137     do i = 2, N - 1, 2
138         ySquareSum = ySquareSum + 4.d0 * Work(i)
139     end do
140     do i = 3, N - 2, 2
141         ySquareSum = ySquareSum + 2.d0 * Work(i)
142     end do
143     ySquareSum = ySquareSum * dx / 3.d0
144 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	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	<div>[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）</div>	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

$$\Psi(x) = \sum_k c_k \Psi_k(x). \quad (3)$$

我们定义两个基矢的内积为

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

系统的能量可表为

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

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

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

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

$$E' - E = \frac{\sum_k (c_k^* \delta_q H_{kq} + \delta_q^* c_k H_{qk})}{\sum_{kk'} c_k^* c_{k'} W_{kk'}} - E \frac{\sum_k (c_k^* \delta_q W_{kq} + \delta_q^* c_k W_{qk})}{\sum_{kk'} c_k^* c_{k'} W_{kk'}} = 0, \quad (7)$$

$$\implies \sum_k \delta_q^* c_k H_{qk} - E \sum_k \delta_q c_k W_{qk} + \text{c.c.} = 0, \quad (8)$$

$$\implies \sum_k c_k H_{qk} = E \sum_k c_k W_{q,k}. \quad (9)$$

考虑到 $q$ 取值的任意性, 上式对 $q = 1, 2, \dots, N$ 都成立:

$$\sum_k c_k H_{qk} = E \sum_k c_k W_{q,k}, \quad q = 1, 2, \dots. \quad (10)$$

上式可以化为矩阵形式:

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

$$\Rightarrow H\mathbf{c} = E\mathbf{W}\mathbf{c}. \quad (12)$$

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)**

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

**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

```



```

28
29  ! define the basis
30  allocate(basis(2 * Nx + 1, 2 * Nx + 1))
31  basis = 0.d0
32  do i = 1, 2 * Nx + 1      ! order of basis function
33      if (mod(i, 2) /= 0) then
34          ! i is odd
35          do j = 1, 2 * Nx + 1      ! coordinate index
36              basis(i,j) = sqrt(1.d0 / Xmax) * cos(dble(i) * pi * x(j - Nx - 1) / 2.
37                  d0 / Xmax)
38          end do
39      else
40          ! i is even
41          do j = 1, 2 * Nx + 1      ! coordinate index
42              basis(i,j) = sqrt(1.d0 / Xmax) * sin(dble(i) * pi * x(j - Nx - 1) / 2.
43                  d0 / Xmax)
44          end do
45      end if
46  end do
47
48  ! define the Hamiltonian matrix on the basis
49  allocate(Ham(2 * Nx + 1, 2 * Nx + 1))
50  Ham = 0.d0
51  do i = 1, 2 * Nx + 1
52      do j = i, 2 * Nx + 1
53          call Simpson(2 * Nx + 1, basis(i, 1:2 * Nx + 1), basis(j, 1:2 * Nx + 1),
54              Vpot(-Nx:Nx), dx, V0)
55          Ham(i, j) = V0
56          if (i == j) then
57              Ham(i, j) = Ham(i, i) + (dble(i) * pi)**2 / 8.d0 / Xmax**2
58          end if
59          Ham(j, i) = Ham(i, j)
60      end do
61  end do
62
63  ! calculate the eigenvalues of the Hamiltonian
64  allocate(W(2 * Nx + 1))
65  LWORK = 10 * Nx
66  allocate(WORK(LWORK))
67  call DSYEV('V', 'U', 2 * Nx + 1, Ham, 2 * Nx + 1, W, WORK, LWORK, INFO)
68
69  if (INFO == 0) then
70      open(unit = 1, file = '3-eigenvalue-ExactDiagonalization.txt', status = '
71          unknown')

```

```

69      ! output results
70      do i = 1, 2 * Nx + 1
71          write(1, '(i4,f20.8)') i, W(i)
72      end do
73      close(1)
74  else
75      write(*, *) 'Diagonalization went wrong! aborting...'
76      stop
77  end if
78
79      deallocate(Ham)
80      deallocate(basis, W, WORK)
81      deallocate(x, Vpot)
82
83  end program main
84
85  subroutine Simpson(N, u, v, Vpot, dx, V0)
86      ! calculate the integral of the product of u, v, and Vpot from -Xmax to Xmax with
      ! compound simpson formula
87
88      implicit none
89      integer :: N
90      real(8), intent(in) :: u(N), v(N), Vpot(N), dx
91      real(8), intent(out) :: V0
92
93      ! local vars
94      integer :: i
95      real(8) :: Work(N)
96
97      if (mod(N, 2) == 0) then
98          ! Simpson does not work for integral of array with even number of elements
99          write(*, *) 'Array with even elements, Simpson does not know how to work!'
100     end if
101
102     do i = 1, N
103         Work(i) = Vpot(i) * u(i) * v(i)
104     end do
105
106     V0 = Work(1) + Work(N)
107     do i = 2, N - 1, 2
108         V0 = V0 + 4.d0 * Work(i)
109     end do
110     do i = 3, N - 2, 2
111         V0 = V0 + 2.d0 * Work(i)
112     end do

```

```

113     V0 = V0 * dx / 3.d0
114
115 end subroutine Simpson

```

Numerov方法: 计算步骤类似Problem 1. Fortran代码:

```

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 = 100, i, n = 1, nmax = 20
11     real(dp) :: E = 7.d0      ! trial energy, lower than the first energy level
12     real(dp) :: dx, ySquareSum
13     real(dp), allocatable :: x(:), Vpot(:), y(:), g(:), f(:)
14     logical :: yNSignChange, yNSign
15     integer :: looptime
16     real(dp) :: Eprev1, Eprev2, yprev1, yprev2
17
18     ! executable
19     ! write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'
20
21     ! memory dynamical allocation
22     allocate(x(-Nx:Nx))
23     allocate(Vpot(-Nx:Nx))
24     allocate(y(-Nx:Nx))
25     allocate(g(-Nx:Nx))
26     allocate(f(-Nx:Nx))
27
28     ! discretization and define the potential Vpot
29     dx = L / dble(Nx)
30     do i = -Nx, Nx
31         x(i) = dx * i
32         if (abs(x(i)) <= 0.5) then
33             Vpot(i) = 10.d0
34         else
35             Vpot(i) = 0.d0
36         end if
37     end do
38
39     open(2, file = '3-energy-Numerov.txt', status = 'unknown')

```

```

40  do while (n <= nmax)
41      looptime = 1
42      yNSignChange = .false.
43      do while (.true.)
44          ! set boundary condition: y(-Nx) and y(-Nx+1)
45          y(-Nx) = 0.d0
46          y(-Nx + 1) = 1.d-4
47
48          do i = -Nx, Nx
49              g(i) = 2.d0 * (E - Vpot(i))
50              f(i) = 1.d0 + g(i) / 12.d0 * dx**2
51          end do
52
53          ! Numerov's method
54          do i = -Nx + 1, Nx - 1
55              y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i
                    + 1)
56          end do
57
58          ! renormalize y(x)
59          call Simpson(2 * Nx + 1, y(:), dx, ySquareSum)
60          y = y / sqrt(ySquareSum)
61
62          ! change E
63          if (abs(y(Nx)) < eps) then      ! if precision requirement is satisfied
64              write(2, '(i4,f20.8)') n, E
65              exit
66          else
67              if (.not. yNSignChange) then      ! if the sign of yN has not changed
68                  E = E + dE
69                  if (looptime == 1) then      ! if this is the first loop
70                      yNSign = (y(Nx) >= 0)
71                      yprev1 = y(Nx)
72                  else
73                      if ((y(Nx) >= 0) .neqv. (yNSign)) then      ! if the sign of yN
                          changed this time
74                          yNSignChange = .true.
75                          E = E - dE
76                          Eprev1 = E
77                          Eprev2 = E - dE
78                          E = E - dE / 2.d0
79                      end if
80                      yprev2 = yprev1
81                      yprev1 = y(Nx)
82                  end if

```

```

83         else      ! once the sign of yN has changed
84             if (abs(yprev1) <= abs(yprev2)) then
85                 Eprev2 = E
86                 ! Eprev1 = Eprev1
87                 E = (Eprev1 + Eprev2) / 2.d0
88                 yprev2 = y(Nx)
89                 ! yprev1 = yprev1
90             else
91                 ! Eprev2 = Eprev2
92                 Eprev1 = E
93                 E = (Eprev1 + Eprev2) / 2.d0
94                 ! yprev2 = yprev2
95                 yprev1 = y(Nx)
96             end if
97         end if
98     end if
99     looptime = looptime + 1
100 end do
101 E = E + 2 * dE
102 n = n + 1
103 end do
104 close(2)
105 end program main
106
107 subroutine Simpson(N, y, dx, ySquareSum)
108     ! calculate the integral of the square of the wavefunction from -Xmax to Xmax with
109     ! compound simpson formula
110
111     implicit none
112     integer :: N
113     real(8), intent(in) :: y(N), dx
114     real(8), intent(out) :: ySquareSum
115
116     ! local vars
117     integer :: i
118     real(8) :: Work(N)
119
120     if (mod(N,2) == 0) then
121         ! Simpson does not work for integral of array with even number of elements
122         write(*, *) 'Array with even elements, Simpson does not know how to work!'
123     end if
124
125     do i = 1,N
126         Work(i) = y(i)**2
127     end do

```

```

127
128   ySquareSum = Work(1) + Work(N)
129   do i = 2, N - 1, 2
130       ySquareSum = ySquareSum + 4.d0 * Work(i)
131   end do
132   do i = 3, N - 2, 2
133       ySquareSum = ySquareSum + 2.d0 * Work(i)
134   end do
135   ySquareSum = ySquareSum * dx / 3.d0
136
137 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.  $\square$

**Problem 4 Score:** \_\_\_\_\_. Double well, quantum tunneling, instantons.

Consider a double well system with the following Hamiltonian,

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

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

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

方便起见, 设 $\hbar$ ,  $m$ ,  $\omega_0$ ,  $x_0$ 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 $[-5, 5]$ .

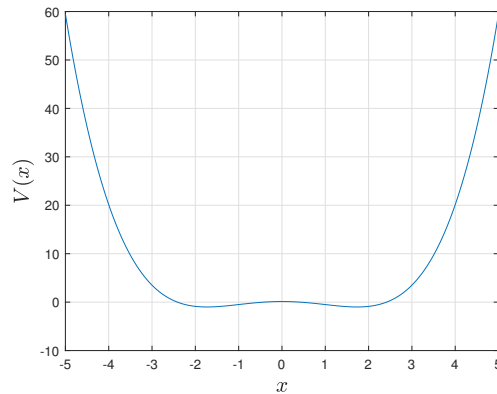


图 1: 双势阱的势场.

**Numerov方法:** Fortran代码:

```

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) :: Xmax
10    integer :: Nx, i
11    real(dp) :: E      ! trial energy
12    real(dp) :: dx, ySquareSum
13    real(dp), allocatable :: x(:), Vpot(:), y(:), g(:), f(:)
14    logical :: yNSignChange = .false., yNSign
15    integer :: looptime = 1
16    real(dp) :: Eprev1, Eprev2, yprev1, yprev2
17
18    ! executable
19    Xmax = -1.d0
20    do while (Xmax < 0.d0)
21        write(*,*) 'Please indicate the range of x, [-Xmax, Xmax]'
```

```

22      read(*,*) Xmax
23  end do
24
25  Nx = -1
26  do while ((Nx <= 0) .or. (mod(Nx, 2) /= 0))
27      write(*,*) 'Please specify the # of slices between [0,Xmax], Nx='
28      read(*,*) Nx
29  end do
30
31  write(*,*) 'Please give a trial energy E0, the first allowed state with E>E0 will
    be calculated! E0='
32  read(*,*) E
33
34  write(*, '(a10,a20,a20)') 'Iteration', 'Energy', 'Boundary value'
35
36  ! memory dynamical allocation
37  allocate(x(-Nx:Nx))
38  allocate(Vpot(-Nx:Nx))
39  allocate(y(-Nx:Nx))
40  allocate(g(-Nx:Nx))
41  allocate(f(-Nx:Nx))
42
43  dx = Xmax / dble(Nx)
44  do i = -Nx, Nx
45      x(i) = dx * i
46      Vpot(i) = .5d0 * ((x(i)**2 - 1.d0)**2 / 4.d0 - x(i)**2)
47  end do
48
49  do while (.true.)
50      ! set boundary condition: y(0) and y(1)
51      y(-Nx) = 0.d0
52      y(-Nx + 1) = 1.d-4
53
54      do i = -Nx, Nx
55          g(i) = 2.d0 * (E - Vpot(i))
56          f(i) = 1.d0 + g(i) / 12.d0 * dx**2
57      end do
58
59      ! Numerov's method
60      do i = -Nx + 1, Nx - 1
61          y(i + 1) = ((12.d0 - 10.d0 * f(i)) * y(i) - f(i - 1) * y(i - 1)) / f(i + 1)
62      end do
63
64      ! renormalize y(x)
65      call Simpson(Nx + 1, y(:), dx, ySquareSum)

```



```

66     y = y / sqrt(ySquareSum)
67
68     ! output to screen
69     write(*, '(i10,2f20.8)') looptime, E, y(Nx)
70
71     ! change E
72     if (abs(y(Nx)) < eps) then      ! if precision requirement is satisfied
73         ! output to file
74         open(1, file = '4-wavefunction-Numerov.txt', status = 'unknown')
75         do i = -Nx, Nx
76             write(1, '(2f20.8)') x(i), y(i)
77         end do
78         close(1)
79         open(2, file = '4-energy-Numerov.txt', status = 'unknown')
80         write(2, '(f20.8)') E
81         close(2)
82         exit
83     else
84         if (.not. yNSignChange) then    ! if the sign of yN has not changed
85             E = E + dE
86             if (looptime == 1) then      ! if this is the first loop
87                 yNSign = (y(Nx) >= 0)
88                 yprev1 = y(Nx)
89             else
90                 if ((y(Nx) >= 0) .neqv. (yNSign)) then    ! if the sign of yN
91                     ! changed this time
92                     yNSignChange = .true.
93                     E = E - dE
94                     Eprev1 = E
95                     Eprev2 = E - dE
96                     E = E - dE / 2.d0
97                 end if
98                 yprev2 = yprev1
99                 yprev1 = y(Nx)
100             end if
101         else    ! once the sign of yN has changed
102             if (abs(yprev1) <= abs(yprev2)) then
103                 Eprev2 = E
104                 ! Eprev1 = Eprev1
105                 E = (Eprev1 + Eprev2) / 2.d0
106                 yprev2 = y(Nx)
107                 ! yprev1 = yprev1
108             else
109                 ! Eprev2 = Eprev2
110                 Eprev1 = E

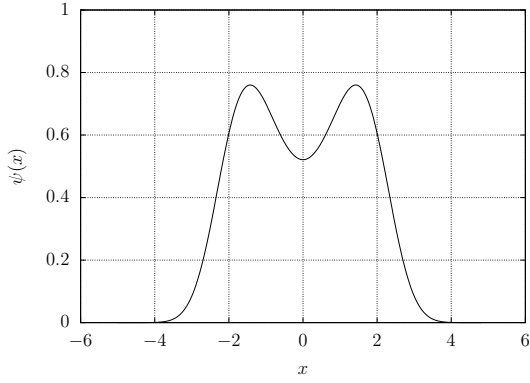
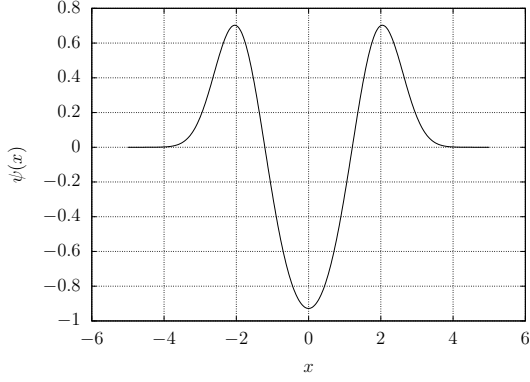
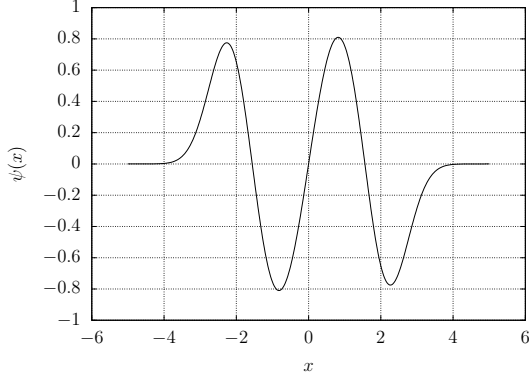
```

```

110             E = (Eprev1 + Eprev2) / 2.d0
111             ! yprev2 = yprev2
112             yprev1 = y(Nx)
113         end if
114     end if
115 end if
116
117     looptime = looptime + 1
118 end do
119 end program main
120
121 subroutine Simpson(N, y, dx, ySquareSum)
122     ! calculate the integral of the square of the wavefunction from -Xmax to Xmax with
123     ! compound simpson formula
124
125     implicit none
126     integer :: N
127     real(8), intent(in) :: y(N), dx
128     real(8), intent(out) :: ySquareSum
129
130     ! local vars
131     integer :: i
132     real(8) :: Work(N)
133
134     if (mod(N,2) == 0) then
135         ! Simpson does not work for integral of array with even number of elements
136         write(*, *) 'Array with even elements, Simpson does not know how to work!'
137     end if
138
139     do i = 1, N
140         Work(i) = y(i)**2
141     end do
142
143     ySquareSum = Work(1) + Work(N)
144     do i = 2, N - 1, 2
145         ySquareSum = ySquareSum + 4.d0 * Work(i)
146     end do
147     do i = 3, N - 2, 2
148         ySquareSum = ySquareSum + 2.d0 * Work(i)
149     end do
150     ySquareSum = ySquareSum * dx / 3.d0
151 end subroutine Simpson

```

部分计算结果如表.

能量	波函数图像
-0.32227314	
0.83123120	
1.71356141	

矩阵对角化方法：以频率为 $\omega_0$ 的量子谐振子的特征波函数

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

为基矢，其中厄米多项式

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

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

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

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

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

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

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:

```

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, LWORK, INFO
10    real(dp) :: Xmax, dx, V0
11    real(dp), allocatable :: x(:), Vpot(:), Ham(:, :), basis(:, :), W(:), WORK(:)
12
13    integer :: LWORK1, INFO1
14    real(dp), allocatable :: x1(:), Vpot1(:), Ham1(:, :), W1(:), WORK1(:)
15
16    ! executable
17    Xmax = -1.d0
18    do while (Xmax < 0.d0)
19        write(*,*) 'Please indicate the range of x, [-Xmax, Xmax]; Xmax='
20        read(*,*) Xmax
21    end do
22
23    Nx = -1
24    do while (Nx <= 0)
25        write(*,*) 'Please specify the # of slices between [0, Xmax], Nx='
26        read(*,*) Nx
27    end do
28
29    ! memory dynamical allocation
30    allocate(x(-Nx:Nx))
31    allocate(Vpot(-Nx:Nx))

```

```

32  allocate(x1(-Nx - 1:Nx + 1))
33  allocate(Vpot1(-Nx - 1:Nx + 1))
34
35  dx = Xmax / dble(Nx)
36  do i = -Nx, Nx
37      x(i) = dx * dble(i)
38      Vpot(i) = .5d0 * ((x(i)**2 - 1.d0)**2 / 4.d0 - x(i)**2)
39  end do
40
41  ! calculate the wavefunction of 1D HO
42  do i = -Nx - 1, Nx + 1
43      x1(i) = dx * dble(i)
44      Vpot1(i) = .5d0 * x1(i)**2
45  end do
46
47  allocate(Ham1(2 * Nx + 1, 2 * Nx + 1))
48  Ham1 = 0.d0
49  do i = 1, 2 * Nx + 1
50      Ham1(i,i) = Vpot1(i - Nx) + 1.d0 / dx**2
51      if (i > 1) Ham1(i,i - 1) = -.5d0 / dx**2
52      if (i < 2 * Nx + 1) Ham1(i,i + 1) = -.5d0 / dx**2
53  end do
54
55  allocate(W1(2 * Nx + 1))
56  LWORK1 = 10 * (Nx + 1)
57  allocate(WORK1(LWORK1))
58  call DSYEV('V','U',2 * Nx + 1, Ham1, 2 * Nx + 1, W1, WORK1, LWORK1, INFO1)
59
60  allocate(basis(2 * Nx + 1, 2 * Nx + 1))
61  if (INFO1 == 0) then
62      do i = 1, 2 * Nx + 1
63          do j = 1, 2 * Nx + 1
64              basis(i,j) = Ham1(j,i)
65          end do
66      end do
67  else
68      write(*, *) 'Diagonalization went wrong! aborting...'
69      stop
70  end if
71
72  allocate(Ham(2 * Nx + 1, 2 * Nx + 1))
73  Ham = 0.d0
74  do i = 1, 2 * Nx + 1
75      do j = i, 2 * Nx + 1
76          call Simpson(2 * Nx + 1, basis(i, 1:2 * Nx + 1), basis(j, 1:2 * Nx + 1),

```

```

      Vpot(-Nx:Nx) = .5d0, dx, V0)
77      Ham(i, j) = V0
78      if (i == j) then
79          Ham(i, j) = Ham(i, j) + (i - .5d0)
80      end if
81      Ham(j, i) = Ham(i, j)
82  end do
83 end do

      allocate(W(2 * Nx + 1))
86 LWORK = 10 * Nx
87 allocate(WORK(LWORK))
88 call DSYEV('V', 'U', 2 * Nx + 1, Ham, 2 * Nx + 1, W, WORK, LWORK, INFO)
89
90 if (INFO == 0) then
91     open(unit = 1, file = '4-energy-ExactDiagonalization.txt', status = 'unknown')
92     open(unit = 2, file = '4-wavefunction-ExactDiagonalization.txt', status = '
        unknown')
93
94     ! multiply the coefficient (which is stored in Ham now) with the basis function
        to get the wave function
95     Ham = matmul(transpose(basis), Ham)
96
97     ! output results
98     do i = 1, 2 * Nx + 1
99         write(1, '(i4,f20.12)') i, W(i)
100        write(2, '(f12.8)', advance = 'no') x(i - Nx - 1)
101        do j = 1, 2 * Nx + 1
102            write(2, '(f12.6)', advance = 'no') Ham(i, j)
103        end do
104        write(2, *)
105    end do
106    close(1)
107    close(2)
108 else
109     write(*, *) 'Diagonalization went wrong! aborting...'
110     stop
111 end if

112
113 deallocate(Ham)
114 deallocate(basis, W, WORK)
115 deallocate(x, Vpot)
116
117 end program main
118

```

```

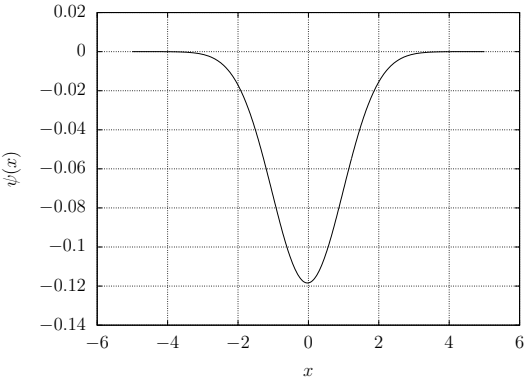
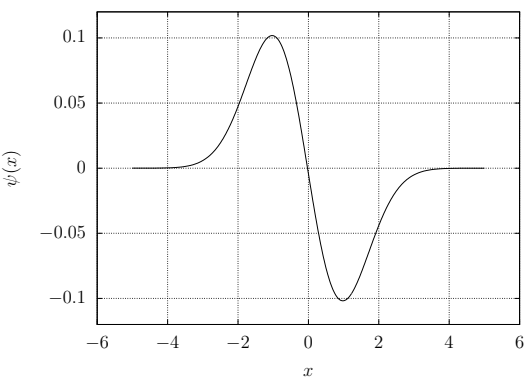
119 subroutine Simpson(N, u, v, Vpot, dx, V0)
120   ! calculate the integral of the product of u, v, and Vpot from -Xmax to Xmax with
      compound simpson formula
121
122   implicit none
123   integer :: N
124   real(8), intent(in) :: u(N), v(N), Vpot(N), dx
125   real(8), intent(out) :: V0
126
127   ! local vars
128   integer :: i
129   real(8) :: Work(N)
130
131   if (mod(N, 2) == 0) then
132     ! Simpson does not work for integral of array with even number of elements
133     write(*, *) 'Array with even elements, Simpson does not know how to work!'
134   end if
135
136   do i = 1, N
137     Work(i) = Vpot(i) * u(i) * v(i)
138   end do
139
140   V0 = Work(1) + Work(N)
141   do i = 2, N - 1, 2
142     V0 = V0 + 4.d0 * Work(i)
143   end do
144   do i = 3, N - 2, 2
145     V0 = V0 + 2.d0 * Work(i)
146   end do
147   V0 = V0 * dx / 3.d0
148
149 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	