

一维定态薛定谔方程本征值求解的三种方法 比较

俞炀

2020.6.9

摘要

文章以求解一维定态薛定谔方程为例，介绍了打靶法 ode45、Cowling 算法、矩阵法、伪扩散方法三种不同的求解本征问题的数值算法。其中对于一维平底势阱与一维二次势阱本征值与本征函数进行了详细求解作图。文章从求解定态薛定谔方程开始，分析比对了三种方法的优劣，并对其适应的范围做出了分析，从而对于求解类似问题时采用何种较优手段有较为准确的认识。

关键词： 定态薛定谔方程 打靶法 矩阵法 伪扩散法

1 定态薛定谔方程的导出

1.1 方程通解

由量子力学的学习，我们可知含时的薛定谔方程可以表述成：

$$i\hbar \frac{\partial \Psi}{\partial t} = \hat{H}\Psi \quad (1)$$

其中， $\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V$ 为哈密顿算符。

对于该方程，我们通过分离时空变量，得到定态薛定谔方程：

设 $\Psi(\vec{r}, t) = \Psi(\vec{r})T(t)$ ，代入方程 1 中，对于两边同时除以 Ψ ，可得：

$$\frac{1}{T}i\hbar T'(t) = \frac{1}{\Psi}\hat{H}\Psi(\vec{r}) \quad (2)$$

因为方程 2 左侧只是时间 t 的函数，右侧只是空间位置的函数，可知此等式等于一常数。令此常数为 E ，我们得到一个关于时间的常微分方程和一

个只和空间有关的偏微分方程:

$$T' + i\frac{E}{\hbar}T = 0 \quad (3)$$

$$\hat{H}\Psi = E\Psi \quad (4)$$

式 3 可轻松解出: $T = \exp(-i\frac{E}{\hbar}t)$ 。式 4 为我们所需求解的定态薛定谔方程。其中 E 为体系的能量, 其为一系列离散值, 即哈密顿算符在给定边界条件下的本征值。

在一维情况下, 式 4 可以化为:

$$(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V)\Psi = E\Psi \quad (5)$$

1.2 条件设定

- 关于自变量 x , 其取值范围我们定为 $x \in [-1, 1]$ (在某些情况下取值范围为 $[0, 1]$, 在 x 取值为 $[-1, 1]$ 时, 将会具体标出)。
- 对势能项进行约化, 即令 $V = V_0\nu(x)$

经过简单计算, 我们可得到约化形式的一维定态薛定谔方程:

$$(-\frac{1}{\gamma^2}\frac{d^2}{dx^2} + \nu(x) - \varepsilon)\Psi = 0 \quad (6)$$

其中 $\gamma = (\frac{2ma^2V_0}{\hbar^2})(\frac{1}{2})$, $\varepsilon = \frac{E}{V_0}$ 。

1.3 方势阱的薛定谔方程解析求解

求解方势阱时, 我们令 $\nu(x) = \nu_0 = 0$, 此时, 薛定谔方程变为:

$$\frac{d^2\Psi}{dx^2} + k^2\Psi = 0 \quad (7)$$

其中: $k^2 = \gamma^2\varepsilon$ 。

这是一个常系数二阶线性常微分方程。其通解为:

$$\Psi_n = \begin{cases} A_n \cos(\frac{n\pi x}{2}), & n = \text{奇数}, \\ A_n \sin(\frac{n\pi x}{2}), & n = \text{偶数} \end{cases} \quad (8)$$

由此, 我们得到其本征值为 $k_n = \frac{n\pi}{2}$, 相应的本正能量 $\varepsilon_n = \frac{1}{\gamma^2}(\frac{n\pi}{2})^2$ 。

1.4 二次势的薛定谔方程的解析求解

与量子力学中相对照，可知二次势与一维谐振子势相似：

$$V(x) = \frac{1}{2}m\omega^2x^2 \quad (9)$$

其势函数的解析解为厄密多项式，由于其过于繁琐，在此不再罗列；其能量本征值为：

$$E = \frac{1}{2}\hbar\omega + n\hbar\omega \quad (10)$$

在求解不同形式的二次势的情况下，只需要将一维谐振子势进行平移伸缩即可。

此处求解的二次势区间为 $x \in [-1, 1]$ ，势函数为 $V(x) = 2500(\frac{x^2}{2} - 1)$ 。令 $\frac{\hbar}{2m} = 1$ ，我们可得薛定谔方程为：

$$\frac{d^2}{dx^2}\Psi + [E - 2500(\frac{x^2}{2} - 1)\Psi] = 0 \quad (11)$$

令式 11 中 $E' = E + 2500$ ，代入谐振子薛定谔方程中，可得：

$$\frac{1}{2}m\omega^2 = 1250 \quad (12)$$

解得： $\omega = \sqrt{\frac{2500}{m}} = \frac{50\sqrt{2}}{\hbar}$

代入谐振子能量本征值，可得：

$$E' = (\frac{1}{2} + n)\hbar\omega = (\frac{1}{2} + n) \times 50\sqrt{2} \approx 70.710678 \times (\frac{1}{2} + n) \quad (13)$$

根据 $E' = E + 2500$ 可得：

$$E_n = (\frac{1}{2} + n) \times 70.710678 - 2500 \quad (14)$$

由上式 14 可知，我们所求得地二次势基态能量本征值数值大小应为：

$$E_1 = -2464.6447$$

2 打靶法

2.1 简介

打靶法，即：猜一个本征值，将其代入方程直接积分，并观察得到的端点值是否满足相应的边界条件，如果不满足就改变猜测的本征值继续进行

同样的过程。此过程与枪手打靶类似，因此被形象地称为打靶法。另一方面，我们对方程进行积分得到的端点值依赖于方程的本征值，因此，端点值可以看成是本征值的函数。对于端点值连续依赖本征值的情形，此问题即转化成为方程求根问题。这个时候为了稳定，我们通常选择最简单的搜索法来进行这一个过程。因此，打靶法拆开来看其实就是一个求解方程的过程与方程求根的结合。

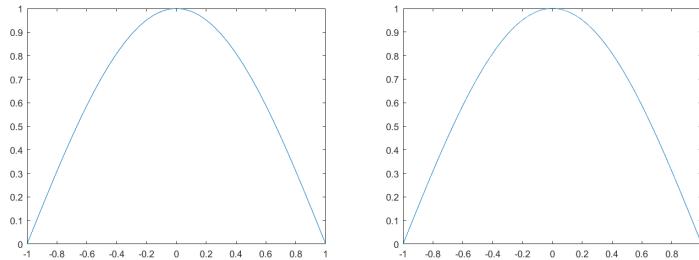
在使用打靶法求解薛定谔方程中方势阱、二次势阱的过程中，我们分别采用了 Numerov 算法以及使用 Matlab 自带的 ode45 算法求解微分方程。我们将以表格形式分别给出方势阱和二次势阱的本征值，并以图象形式表现两者的本征函数。

2.2 一维薛定谔方程——方势阱求解本征值、本征函数

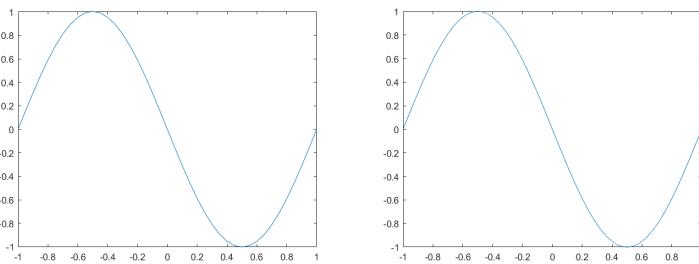
2.2.1 本征值

初始能量	本征能量 (ode45)	本正能量 (Numerov)	量子数	解析能量本征值
2.4	2.4674	2.4674	1	2.4674
9.8	9.8691	9.8696	2	9.8696
22.1	22.1950	22.2066	3	22.2066

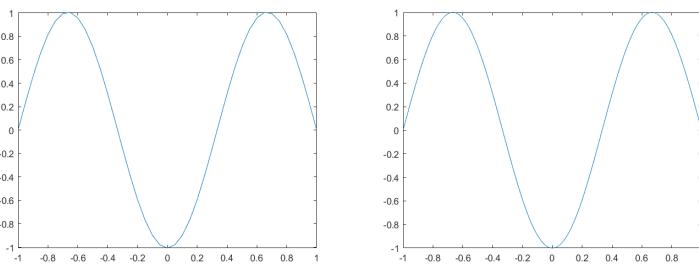
2.2.2 本征函数 (图像)



(a) $n=1$ 时 ode45 求解得到的本征函数 (b) $n=1$ 时 Numerov 求解得到的本征函数



(c) $n=2$ 时 ode45 求解得到的本征函数 (d) $n=2$ 时 Numerov 求解得到的本征函数



(e) $n=3$ 时 ode45 求解得到的本征函数 (f) $n=3$ 时 Numerov 求解得到的本征函数

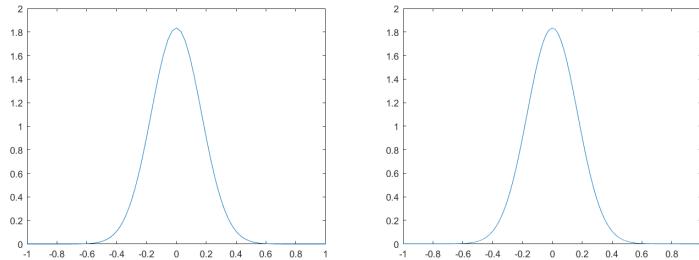
图 1: 一维平底势阱下量子数分别为 1、2、3 时使用 ode45 与 numerov 算法求解得出的势函数

2.3 一维薛定谔方程——二次势阱求解本征值、本征函数

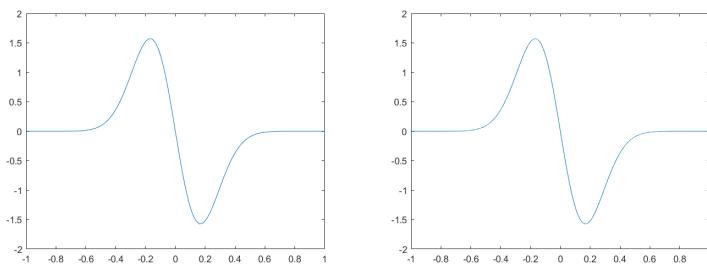
2.3.1 本征值

初始能量	本征能量 (ode45)	本正能量 (Numerove)	量子数	解析能量本征值
-2465	-2.4646e+03	-2.4646e+03	1	-2464.6447
-2394	-2394	-2.3939e+03	2	-2393.9340
-2324	-2324	-2.3232e+03	3	-2323.2893

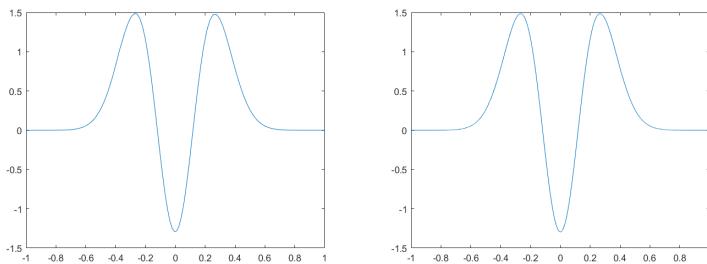
2.3.2 本征函数 (图像)



(a) $n=1$ 时 ode45 求解得到的本征函数 (b) $n=1$ 时 Numerov 求解得到的本征函数



(c) $n=2$ 时 ode45 求解得到的本征函数 (d) $n=2$ 时 Numerov 求解得到的本征函数



(e) $n=3$ 时 ode45 求解得到的本征函数 (f) $n=3$ 时 Numerov 求解得到的本征函数

图 2: 一维二次势阱下量子数分别为 1、2、3 时使用 ode45 与 numerov 算法求解得出的势函数

3 分析

- a 一维方势阱由于此方程不存在转折点，因此无论是 ode45 还是 numerove 算法均采用从 x 的一边求解至另一端；一维二次势阱中 ode45 是从 x 的两端开始求解的，而 numerove 算法在此出现了问题，因此依然采用从一边求解至另一边，通过比较另一边边值条件与计算值之间的正负进行步长的调整。由本征值可得出，无论是哪一种势，Numerove 算法的计算值均优于 ode45 算法的计算值；
- b 由于一维二次势阱本征值之间相差巨大，因此如果猜测值与本征值相差很远，将会出现运算时间极长的情况；
- c 无论是 ode45 还是 numerove 算法，两者均不能计算多个本征值，输入一个猜测值只能计算出一个本征值。

4 矩阵法

4.1 简介

矩阵方法的基本思想就是偏微分方程的差分方法。当我们把偏微分方程中的微商用差分代替之后，我们就会得到一个差分方程组。本来无穷多个点的情况被我们用有限个点来代替。对于线性方程的情况，我们得到的方程也将是线性方程组，此时偏微分算子即化为了矩阵。式 4 的定态薛定谔方程本质上即为一个本征方程，它告诉我们系统的能量是哈密顿算符的本征态值。现在我们用差分方法来处理哈密顿算符：

$$\hat{H} = -\frac{1}{\gamma^2} \frac{d^2}{dx^2} + \nu(x) = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ \dots & \dots & \dots & \dots & \dots \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} + \begin{bmatrix} \nu_1 & & & & \\ & \nu_2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \nu_n \end{bmatrix} \quad (15)$$

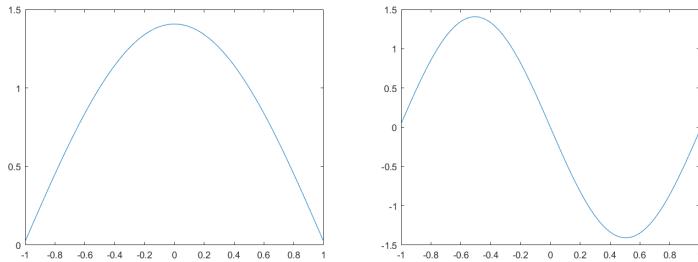
此时，哈密顿算符就变成了一个矩阵，所以问题自然而然的就化为了求这个矩阵的本征值问题。我们通过逆名迭代法求解其本征值与本征向量。以下是在矩阵法下得到的一维方势阱、一维二次势阱下的本征值和本征函数：

4.2 一维薛定谔方程——方势阱求解本征值、本征函数

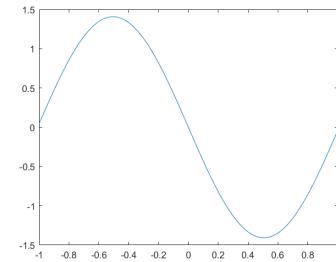
4.2.1 本征值

本征能量	量子数	解析能量本征值
9.8696	1	9.8696
39.4784	2	39.4784
88.8264	3	88.8264
398741.8874	201	398741.8874

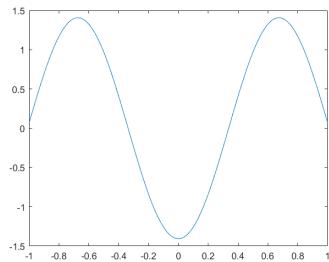
4.2.2 本征函数 (图像)



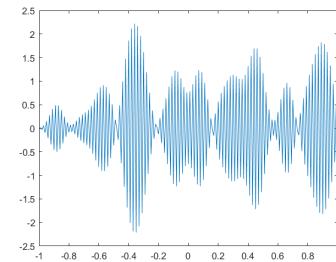
(a) $n=1$ 时矩阵法求解得到的本征函数



(b) $n=2$ 时矩阵法求解得到的本征函数



(c) $n=3$ 时矩阵法求解得到的本征函数



(d) $n=201$ 时矩阵法求解得到的本征函数

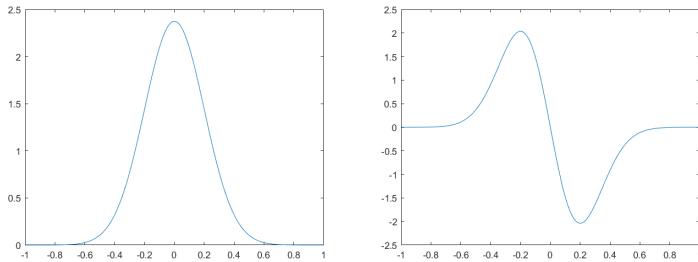
图 3: 一维平底势阱下量子数分别为 1、2、3、201 时使用矩阵法求解得出的势函数

4.3 一维薛定谔方程——二次势阱求解本征值、本征函数

4.3.1 本征值

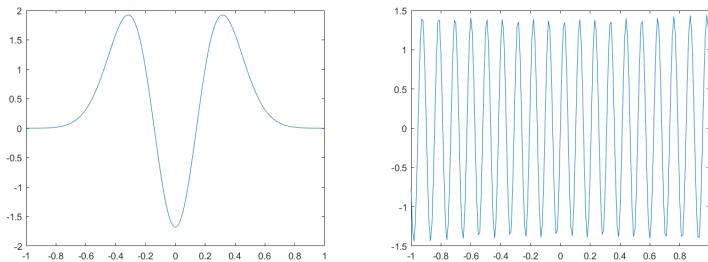
本征能量	量子数	解析能量本征值
-2400.0156	1	-2464.6447
-2200.0781	2	-2393.9340
-2000.2031	3	-2323.2893
11928.4770	201	11748.2016

4.3.2 本征函数(图像)



(a) $n=1$ 时矩阵法求解得到的本征函数

(b) $n=2$ 时矩阵法求解得到的本征函数



(c) $n=3$ 时矩阵法求解得到的本征函数

(d) $n=201$ 时矩阵法求解得到的本征函数

图 4: 一维二次势阱下量子数分别为 1、2、3、201 时使用矩阵法求解得出的势函数

4.4 分析

- a 由矩阵法解出的一维方势阱下的薛定谔方程的本征值与本征函数与解析解符合得非常好，到 n=201 时依然保持着非常稳定的状态，且运行速度较快；
- b 由矩阵法解出得一维二次势阱下得薛定谔方程的本征值与解析解有较大出入，绝对误差达到了几十；
- c 由于逆名迭代法需要有一组本征值的猜测解，此处是根据其本征值解析解的数值给出的猜测解，因此运行较快，在实际情况中求解本征值的过程将减缓许多。

5 伪扩散法

5.1 简介

由含时薛定谔方程：

$$\hat{H}\Psi = E\Psi \quad (16)$$

式中 E 为本征值。此时我们考虑如下扩散方程??

$$\frac{\partial\Psi}{\partial\tau} = -\hat{H}\Psi \quad (17)$$

分离变量，可得时间部分：

$$T' + ET = 0 \quad (18)$$

空间部分与原先的定态薛定谔方程相同。

方程 18 的解为：

$$T = \exp^{-E\tau} \quad (19)$$

由式 19 可知，只要本征值，这个本征态就将随时间指数衰减，而且衰减速度正好是相应的本征值。由于哈密顿算符是一个厄米算符，它的本征态构成一组完备正交基。于是任何一个波函数可以在这组基上展开。之后，我们可以想见，对于任何一个初始的波函数，在这样一个随时间演化的过程之后，本征值最小的分量那个将在最后的结果占据主导地位。而其他成分均被过滤。由此可知，我们可以轻松获得定态薛定谔方程的第一本征态。伪扩散

法的求解可运用欧拉法或后项欧拉法求解。下图分别为由方势阱与二次势阱求得的波函数：

5.2 一维薛定谔方程——方势阱求解第一本征态本征函数图像

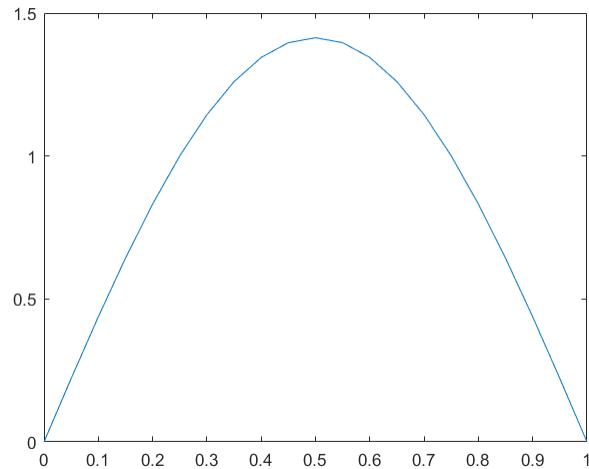


图 5：一维薛定谔方程方势阱第一本征态本征函数图像

其求得的本征值为：9.8493，与解析解求得的本征值 9.8696 有一定的差距；

5.3 一维薛定谔方程——二次势阱求解第一本征态本征函数图像

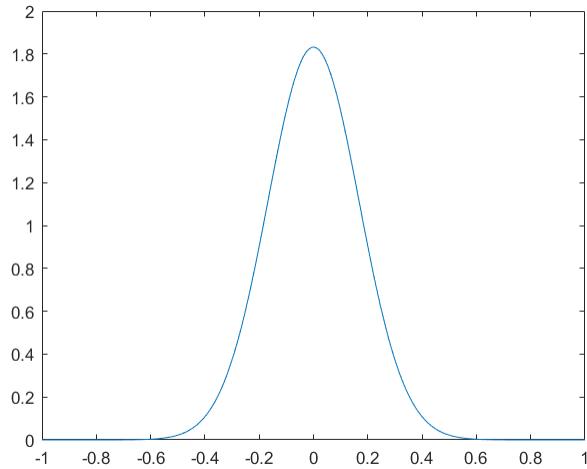


图 6: 一维薛定谔方程二次势阱第一本征态本征函数图像

其求得的本征值为:-2491.1614, 同样,其与解析解得出的本征值-2464.6447有一定差距。

6 总结

打靶法 打靶法适合求解已大致知晓本征值范围的情形。其原理依然是方程求根。其对于对参数敏感的方程不方便处理，无法保证其收敛性；另外，由于在处理实际问题是不清楚其本征态，在遍历的过程中，需要花费较多的时间；其优点是，所求的本征值与本征态相对精准；

矩阵法 矩阵法适合求解大量本征值的问题。其运算速度较快，能一次性求出从基态到几百激发态的本征值与本征函数；可是其精度与求解的势函数有关，在一些情况下其求解到很后面依然保持稳定，而在一些情况下后面的值非常不稳定。

伪扩散法 伪扩散法无论是从精确度，运算数量都位于前两者中间，其缺点也存在，如需计算第一激发态上的本征函数与本征值，其需从基态开始计算并剔除基态、第一激发态……

综上：打靶法、矩阵法、伪扩散法各有其优势与劣势，在不同问题的求解上表现出了不同的特性。在实际解决问题的过程中，我们需要对具体问题具体分析，寻找适合我们求解的方法。

A ode45 求解一维方势阱问题

```
%函数文件
function dydx = test2(x,y)
global e1 V0
dydx = [y(2);(-e1+V0)*y(1)]; 

%运行文件
clc;
clear all;
global e1 V0;
tol = 1e-6; de = 0.0001; e1 = 22.1; V0 = 0;

[x1, u1] = ode45(@test2, [-1,1], [0 0.0001]);
%从左往右积分
f0 = u1(length(x1),1)-0;
%判断斜率是否相等
while (abs(de)>tol)
    e1 = e1 +de;
    [x1, u1] = ode45(@test2, [-1,1], [0 0.0001]);
    f1 = u1(length(x1),1)-0;
    if (f0*f1<=0)
        e1 = e1-de;
        de = de ./ 2;
    end
end
sum = 0;
for i = 1: length(x1)-1
    h = x1(i+1)-x1(i);
```

```

sum = (0.5.*(ul(i)+ul(i+1))).^2.*h+sum;
end
for i = 1:length(x1)
ul(i) = ul(i)./sum.^(1/2);
end
e1

plot(x1,ul(:,1))
axis([-1 1 -inf inf])

```

B ode45 求解一二次方势阱问题

```

%函数文件
function dydx = test(x,y)
global e1
dydx = [y(2);(-e1+50^2.*(x.^2./2-1))*y(1)];

%运行文件
clc; clear all;
global e1;
tol = 1e-6; de = 0.0001; e1 = -2324;
xturn = 0;
%xturn = -sqrt(2*(e1+1));
[x1, u1] = ...
ode45(@test, [-1, xturn], [0 0.0001]);
%从左往右积分
[x2, ur] = ...
ode45(@test, [1, xturn], [0 0.0001]);
%从右往左积分
temp1 = ur(length(ur),1); %归一化第一步
ur(:, :) = ul(length(ul),1)./...
temp1.*ur(:, :); %归一化第二步

```

```

f0 = ul(length(x1),2) - ...
ur(length(x2),2);
%判断斜率是否相等
while (abs(de)>tol)
e1 = e1 +de;
xturn = -sqrt(abs(2*(e1/2500+1)));
[x1,ul] = ode45(@test,[-1,xturn],[0 0.0001]);
[x2,ur] = ode45(@test,[1,xturn],[0 0.0001]);
temp2 = ur(length(ur),1);
%归一化第一步
ur (:,:) = ul(length(x1),1)./temp2.*ur (:,:);
%归一化第二步
f1 = ul(length(x1),2)-ur(length(x2),2);
if (f0.*f1<0||f0.*f1==0)
e1 = e1-de;
de = de./2;
end
end
for k = 1:length(x2)
ur_fixed(k,: ) = ...
ur(length(x2)-k+1,: );
x2_fixed(k) = x2(length(x2)-k+1);
end
u_tot = [ul(:,1);ur_fixed(:,1)];
x_tot = [x1 ; x2_fixed'];

sum = 0;
for s = 1:length(x_tot)-1
h(s) = x_tot(s+1)-x_tot(s);
sum = (0.5.* (u_tot(s)+u_tot(s+1)))...
.^2.*h(s)+sum;
end
for r = 1:length(x_tot)

```

```

u_tot( r ) = u_tot( r )./sum.^^(1/2);
end

e1
%x1(length(ul))
%x2(length(ur))

plot(x1,ul(:,1),x2,ur(:,1));
figure

```

C Numerove 算法求解一维方势阱问题

```

%函数文件
function U = numerove(N,H,Q,S,U)

G = H*H/12.0;
for i = 2:1:N-1
    C0 = 1.0+G.*Q(i-1);
    C1 = 2.0-10.0.*G.*Q(i);
    C2 = 1.0+G.*Q(i+1);
    D = G.*(S(i+1)+S(i-1)+10.0*S(i));
    UTMP = C1*U(i)-C0*U(i-1)+D;
    U(i+1) = UTMP/C2;
end

%运行文件
clc; clear all;
global e1 V0;
tol = 1e-6; de = 0.0001; e1 = 9.8; V0 = 0;
h = 0.0001;
x = -1:h:-1;
n1 = length(1:-h:-1);
x1 = 1:-h:-1;

```

```

u1(1) = 0;
u1(2) = 0.0001;
Q1 = zeros(1,n1);
S1 = ones(1,n1).*(e1-V0);
u1 = numerove(n1,-h,S1,Q1,u1);%从右往左积分
f0 = u1(n1)-0;
while (abs(de)>tol)
    e1 = e1 +de;
    S1 = ones(1,n1).*(e1-V0);
    u1 = numerove(n1,-h,S1,Q1,u1);%从右往左积分
    f1 = u1(n1) - 0;
    if (f0.*f1<0)
        e1 = e1-de;
        de = de./2;
    end
    end
sum = 0;
for i = 1: length(x1)-1
    sum = (0.5.* (u1(i)+u1(i+1))).^2.*h+sum;
end
for i = 1:length(x1)
    u1(i) = u1(i)./sum.^ (1/2);
end
e1
u1(1:length(x1)) = u1(length(x1):-1:1)

plot(x1,u1(:))
axis([-1 1 -inf inf])

```

D Numerove 算法解决一维二次势阱问题

```

%函数文件
function U = numerove(N,H,Q,S,U)

```

```

G = H*H/12.0;
for i = 2:1:N-1
C0 = 1.0+G.*Q(i-1);
C1 = 2.0-10.0.*G.*Q(i);
C2 = 1.0+G.*Q(i+1);
D = G.*(S(i+1)+S(i-1)+10.0*S(i));
UTMP = C1*U(i)-C0*U(i-1)+D;
U(i+1) = UTMP/C2;
end

```

%运行文件

E 段落

```

%具体代码内容
clc;clear all;
global e1 V0;
tol = 1e-6; de = 0.0001;
e1 = -2324; V0 = 0;
h = 0.0001;
n1 = length(-1:h:1);
x1 = -1:h:1;
u1(1) = 0;
u1(2) = 0.0001;
Q1 = zeros(1,n1);
S1 = ones(1,n1).*...
(e1-50.^2.*((x1.^2./2-1)));
u1 = ...
numerove(n1,h,S1,Q1,u1);
%从右往左积分
f0 = u1(n1)-0;
istep = 0;

```

```

while (abs(de)>tol)
e1 = e1 +de;
Q1 = zeros(1,n1);
S1 = ones(1,n1).*...
(e1-50.^2.* (x1.^2./2-1));
u1 = numerove(n1,h,S1,Q1,u1);
%从右往左积分
f1 = u1(n1) - 0;
if (f0.*f1<=0)
e1 = e1-de;
de = de ./ 2;
end
istep = istep + 1;
if (istep >= 187)

end
%plot(x1,u1), pause(0.05);
end
sum = 0;
for i = 1: length(x1)-1
sum = (0.5.* (u1(i)+u1(i+1))).^2.*h+sum;
end
for i = 1:length(x1)
u1(i) = u1(i)./sum.^ (1/2);
end
e1

plot(x1,u1)

```

F 矩阵法解决一维方势阱问题

%函数文件

```

function [lambda ,x] = invpowerit (A,x,s,k)
As = A-s.*eye( size(A));
for j = 1:k
u = x/norm(x);
x = As\u;
lambda = u'*x;
end
lambda = (1./lambda)+s;

%运行文件
clear all;clc;
n = 201;
h = 1/(n-1);
H = zeros(n,n);
for i = 2:n-1
for j = 1:n
if(i==j)
H(i,j) = -2;
H(i-1,j) = 1;
H(i+1,j) = 1;
end
end
end
H(1,1) = -2;
H(n,n) = -2;
H(2,1) = 1;
H(n-1,n) = 1;

H = H*(-1/h^2);

for i = 1:n
energy(i) = i.^2.*pi.^2;
s(i) = 0.999999.*energy(i);

```

```
end
```

```
k = 100;  
x = zeros([n n]);  
save = rand(n,1);  
for i = 1:n  
[lambda(i),x(:,i)] = ...  
invpowerit(H,save,s(i),k);  
sum_of_square(i) = 0;  
for j = 1:n-1  
sum_of_square(i) = sum_of_square(i)...  
+x(j,i).^2.*h;  
end  
for m = 1:n  
x(m,i) = x(m,i)./...  
sum_of_square(i).^(1/2);  
end  
end
```

```
figure  
plot(linspace(-1,1,n),x(:,1));  
figure  
plot(linspace(-1,1,n),x(:,2));  
figure  
plot(linspace(-1,1,n),x(:,3));  
figure  
plot(linspace(-1,1,n),x(:,n));
```

G 矩阵法解决一维二次势阱问题

%函数文件

```

function [lambda ,x] = invpowerit (A,x,s,k)
As = A-s.*eye(size(A));
for j = 1:k
u = x/norm(x);
x = As\u;
lambda = u'*x;
end
lambda = (1./lambda)+s;

%运行文件
clear all;clc;
n = 201;
h = 1/(n-1);
H = zeros(n,n);
V = zeros(n,n);
x = zeros([n n]);
for i = 2:n-1
for j = 1:n
if (i==j)
H(i,j) = -2;
H(i-1,j) = 1;
H(i+1,j) = 1;
end
end
end
H(1,1) = -2;
H(n,n) = -2;
H(2,1) = 1;
H(n-1,n) = 1;
for i = 1:n
q(i) = 2.*((i-1).*h-0.5);
V(i,i) = 2500.*(q(i).^2-1);
end

```

```

H = H*(-1/h^2)+V;

for i = 1:n
energy(i) = -2535.5+70.75.*i;
s(i) = energy(i)-0.00000001;
end
```

```
k = 100;
```

```

save = rand(n,1);
for i = 1:n
[lambda(i),x(:,i)] = ...
invpowerit(H,save,s(i),k);
sum_of_square(i) = 0;
for j = 1:n-1
sum_of_square(i) = ...
sum_of_square(i)+x(j,i).^2.*h;
end
for m = 1:n
x(m,i) = ...
x(m,i)./sum_of_square(i).^(1/2);
end
end

plot(q(:,x(:,1));
figure
plot(q(:,x(:,4));
plot(q(:,x(:,7));
```

H 伪扩散法解决一维方势阱问题

```
clc; clear all
```

```

N = 21;
h = 1./(N-1);
dt = 0.00051;
P = dt ./ (h.^2);
Tspan = 220;
phi = zeros(N,Tspan);

for i = 2:N-1
    x(i) = (i-1).*h;
    phi(i,1) = x(i).*(1-x(i));
end

x(1) = 0;
x(N) = 1;

for i = 1:Tspan;
    for j = 2:N-1
        phi(j,i+1) = phi(j,i)+P.*...
            (phi(j-1,i)+phi(j+1,i)...
            -2.*phi(j,i));
    end
    sum_of_square(i) = 0;

    for j = 1:N-1
        sum_of_square(i) = sum_of_square(i)...
            +phi(j,i).^2.*h;
    end

for m = 1:N
    phi(m,i) = (phi(m,i)).^2./...
        sum_of_square(i)).^(1/2);
end

```

```
plot(x, phi(:, i)), pause(0.05)
```

```
E = 0;  
for k = 2:N  
E = E+(phi(k, i)-phi(k-1, i)).^2;  
end  
energy_value_num(i) = E./h;  
end
```

I 伪扩散法解决一维二次势阱问题

```
%函数文件  
function [x] = chase(a, b, c, f)  
  
N = length(f);  
  
x = zeros(1, N);  
y = zeros(1, N);  
d = zeros(1, N);  
u = zeros(1, N);  
  
d(1) = b(1);  
for i = 1:N-1  
u(i) = c(i)./d(i);  
d(i+1) = b(i+1)-a(i+1).*u(i);  
end  
  
y(1) = f(1)./d(1);  
for i = 2:N  
y(i) = (f(i)-a(i).*y(i-1))./d(i);  
end
```

```

x(N) = y(N);
for i = N-1:-1:1
x(i) = y(i)-u(i).*x(i+1);
end

%运行文件
clc;clear all
N = 1001;
h = 2./(N-1);
dt = 0.0001;
P = dt./((h.^2));
Tspan = 1000;
phi = zeros(N,Tspan);
V(1) = 0;
V(N) = 0;
x = -1:h:1;

for i = 2:N-1
V(i) = -2500.*((0.5*x(i).^2-1));
phi(i,1) = (x(i)+1).*(1-x(i));
end
phi([1, end], :) = 0.0;

%x(1) = -1;
%x(N) = 1;

a = zeros(1,N-2) - P; a(1) = 0.0;
c = zeros(1,N-2) - P; c(end) = 0.0;
b = zeros(1,N-2) + 1.0 +...
2*P - dt.*V(2:1:(N-1));
for i = 1:Tspan-1
phi(2:1:(N-1), i+1) = ...

```

```
chase(a, b, c, phi(2:1:(N-1), i));
sum_of_square = sum(phi(:, i+1).^2.0)*h;
```

```
phi(:, i+1) =...
phi(:, i+1)./sqrt(sum_of_square);
```

```
E = 0;
for k = 2:N
E = E + 0.5.* (phi(k, i) - phi(k-1, i)).^2./h ...
+h.*V(k).*phi(k, i).^2;
end
energy_value_num(i) = E;
end
plot(x, phi(:, Tspan));
```