

HOMEWORK4:薛定谔方程

A.非简谐振子

1.数值求解方法

此题为边界问题，要求满足在边界处：

$$\lim_{x \rightarrow \infty} \phi(x, t) = 0$$

此外，还要保证波函数的归一化

算法一

为了数值求解，不妨认为存在 L 足够大，使得：

$$\phi(L, t) \approx 0$$

和

$$\phi(-L, t) \approx 0$$

其中 L 足够大以覆盖波函数的主要分布范围。

这是一个本征值问题：只有当 E 是某些特殊值（本征值）时，这个边值问题才有非零的解（波函数）。

所以，我们可以使用基于割线法和龙格库塔法结合（打靶法）的思想解决问题：

1. 首先要利用量子力学的一个结论，对于势函数为偶函数的情况下，基函数、偶数级激发态的波函数为偶函数，奇数级激发态为奇函数。
2. 将问题转化为初值问题
猜测一个 E 的值，我们可以设置初值条件
 - 基态或者偶数级激发态： $\psi(0) = 1$ （或其他非零值，数值计算时不影响归一化）和 $\psi'(0) = 0$ 。
 - 奇数级激发态： $\psi(0) = 0$ 和 $\psi'(0) = 1$ （或其他非零值，数值计算时不影响归一化）
3. 使用数值方法（四阶龙格-库塔法）从 $x = 0$ 积分到 $x = L$ 。
4. 检查边界 $\psi(L)$ 是否接近零。如果结果不满足边界条件，使用割线法调整 E 的值。定义 $f(E) = \psi(L)$ ，则

$$E_{n+1} = E_n - \frac{E_n - E_{n+1}}{f(E_n) - f(E_{n-1})} f(E_n)$$

5. 对于新的E的值，回到步骤2，如此迭代，直到满足边界条件
6. 最后，对于找到的E，得出波函数，对波函数进行归一化处理。

写出相关代码进行数值计算，其中用到的割线法和龙格库塔法已在前面的作业中写过。具体代码实现详见附录。

算法二

此算法同老师上课所讲算法。

1. 选择数值解的区域。该区域应足够大，以确保相较于势函数的有效区域，边界处波函数的影响可忽略不计。（思想同算法一中的L）
2. 对最低本征值提供一个合理的猜测。猜测的初始值应该尽可能接近正确值，可以降低运算时间
3. 积分波函数。

- 从左侧将 $\phi_l(x)$ 从 $x = x_l$ 处积分到 $x = x_r + h$ ；
- 从右侧将 $\phi_l(x)$ 从 $x = x_r$ 处积分到 $x = x_l - h$ ；
- 将初始点处的波函数值设为零，并在积分起始点对波函数值进行rescale,确保 $\phi_l(x_r) = \phi_r(x_r)$ 。

4. 计算匹配函数：

$$f(\varepsilon_0) = \frac{[\varphi_r(x_r - h) - \varphi_r(x_r + h)] - [\varphi_l(x_r - h) - \varphi_l(x_r + h)]}{2h\varphi_r(x_r)}$$

5. 使用割线法找到 $f(\varepsilon_0) = 0$ 的解。通过割线法逐步调整 ε 的值，直至满足给定的容忍度。
6. 求解下一个本征值。为了找到下一个本征值，可以从一个略高于上一个本征值的起始值开始搜索。（可以通过检查波函数的节点数来判断是否有遗漏的本征能量）

算法三

也可以将 $[-L, L]$ 的位移 x 离散化，得到一个矢量 $\vec{\phi}(x)$ 。

将薛定谔方程写成差分的格式：

$$-\frac{\hbar^2}{2m} \left[\frac{\phi(x_i - \Delta x) - 2\phi(x_i) + \phi(x_i + \Delta x)}{\Delta x^2} \right] + V(x)\phi(x) = E\phi(x)$$

得到矩阵形式：

得到矩阵方程

$$\begin{bmatrix} A(x_1) & B & 0 & \cdots & 0 \\ B & A(x_2) & B & 0 & \cdots & 0 \\ 0 & B & A(x_3) & B & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & B & A(x_l) \end{bmatrix} \begin{bmatrix} \psi(x_1) \\ \psi(x_2) \\ \psi(x_3) \\ \vdots \\ \psi(x_l) \end{bmatrix} = 0$$

其中

$$A(x_i) = \frac{1}{\Delta x^2} + V(x_i) - E, B = -\frac{1}{2\Delta x^2}$$

则E为该矩阵的本征值，求本征值即可。

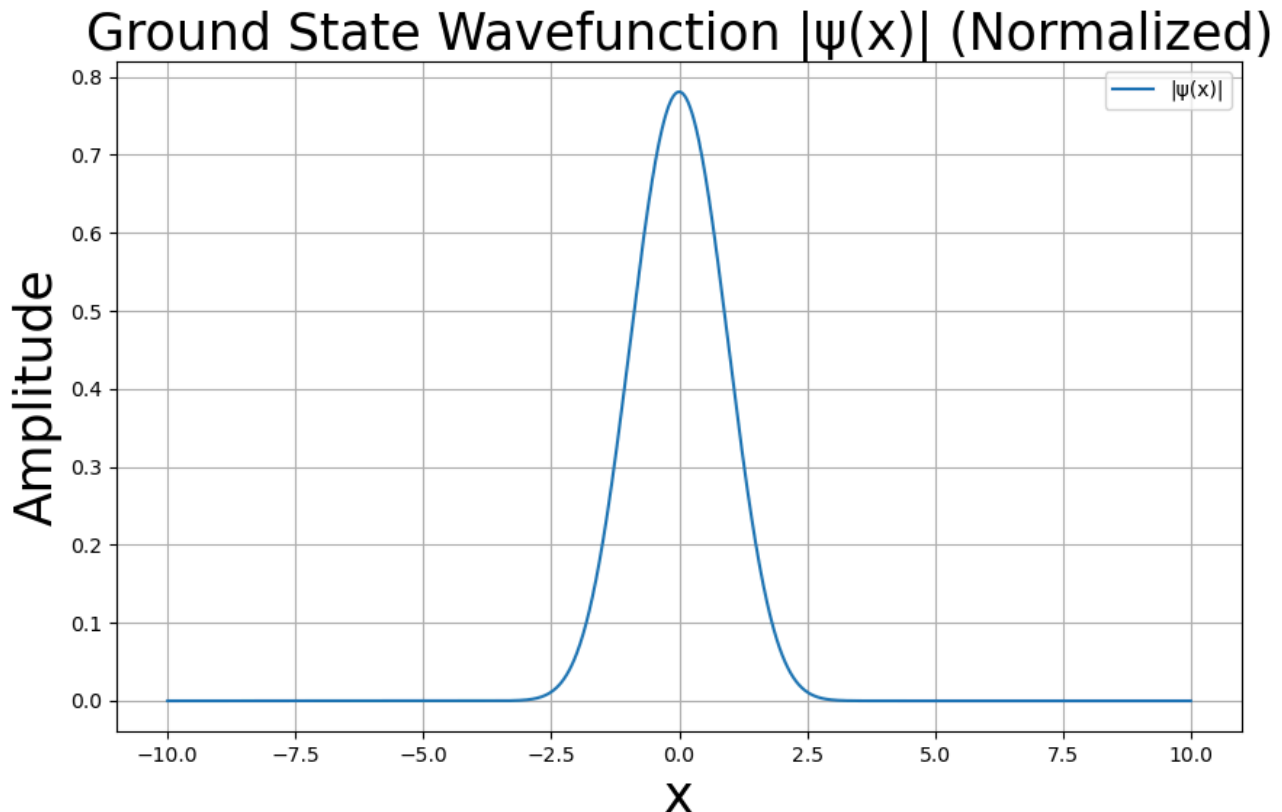
2.计算基态能量

代码见附录，得到的基态能量为0.5594:

```
[Running] python -u "e:\大二下\computational physics\homework\hw_4\code\A_chafen.py"
Ground State Energy: 0.5594

[Done] exited with code=0 in 4.295 seconds
```

得到基态波函数的绝对值的图像为：



3.求解1-4激发态的本征能量和波函数

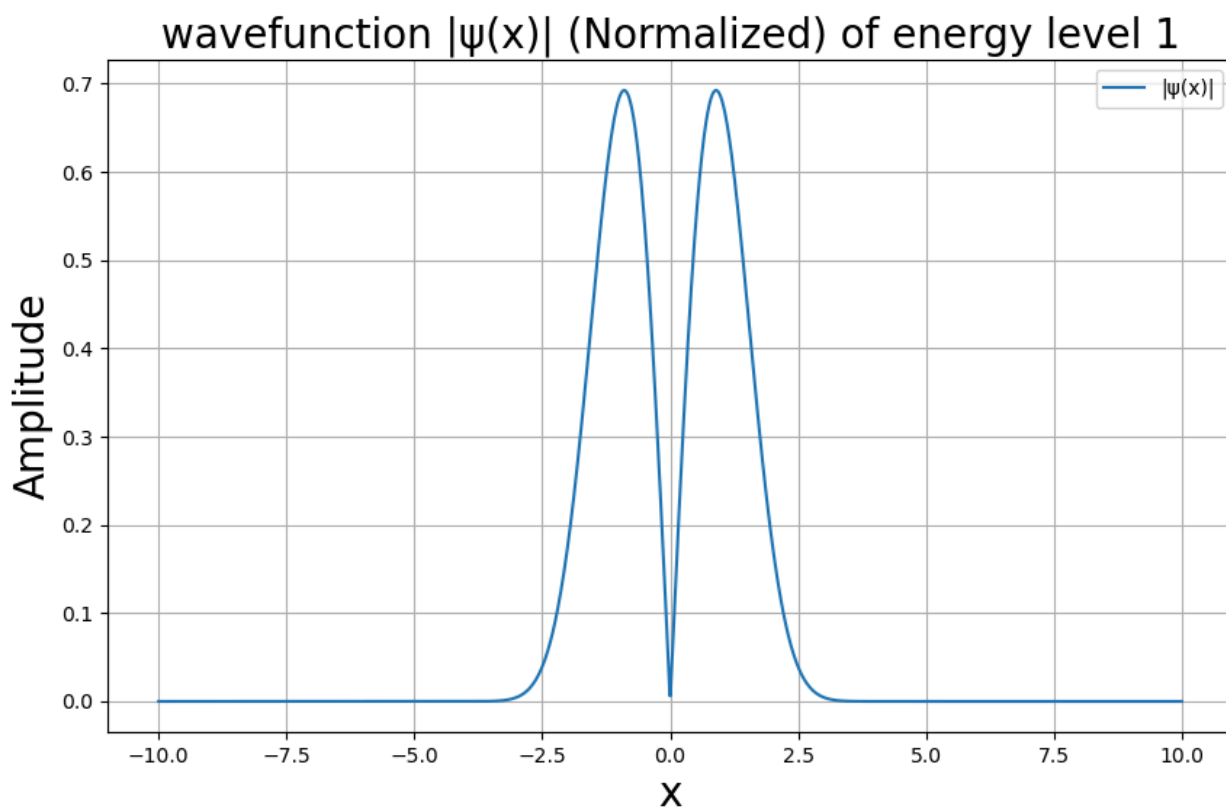
第一激发态到第四激发态的本征能量为: $E_1 = 1.7705, E_2 = 3.1403, E_3 = 4.6314, E_4 = 6.2237$

```
[Running] python -u "e:\大二下\computational physics\homework\hw_4\code\A_chafen.py"
Energy Level 1: 1.7705
Energy Level 2: 3.1403
Energy Level 3: 4.6314
Energy Level 4: 6.2237

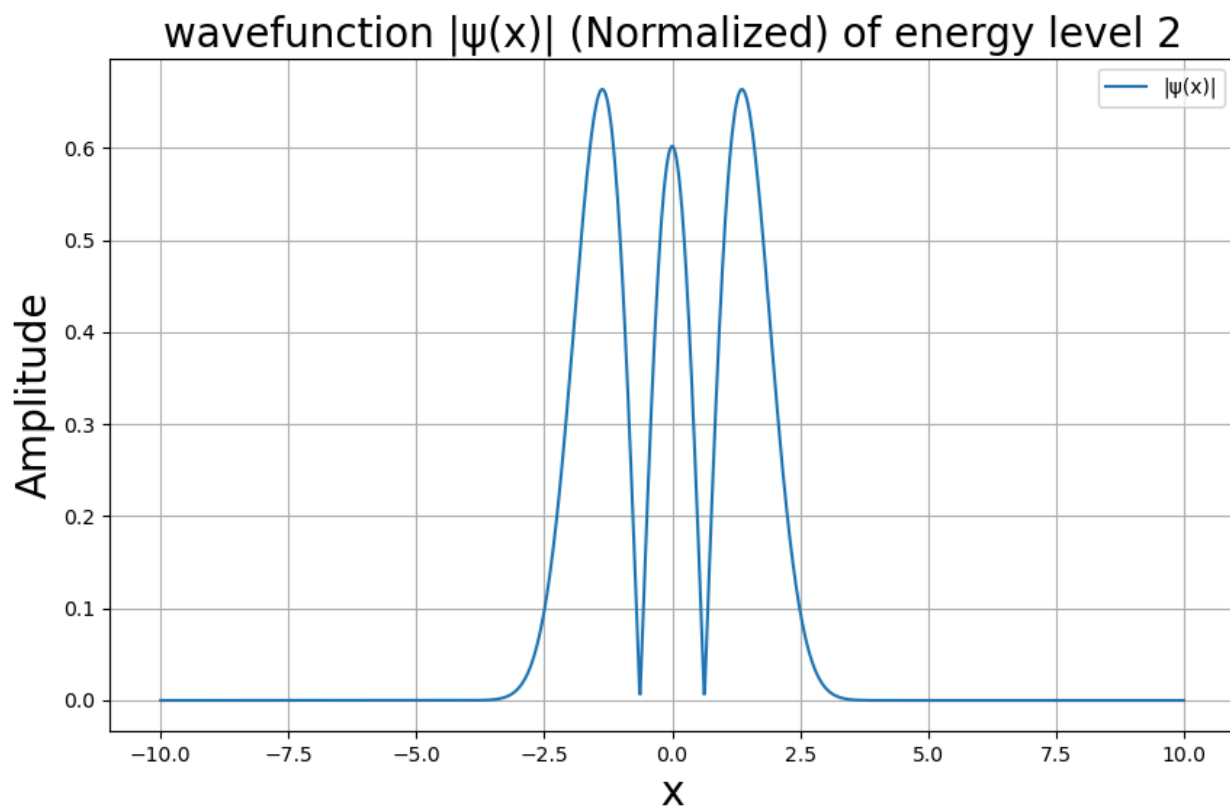
[Done] exited with code=0 in 28.733 seconds
```

波函数**绝对值**的图像为:

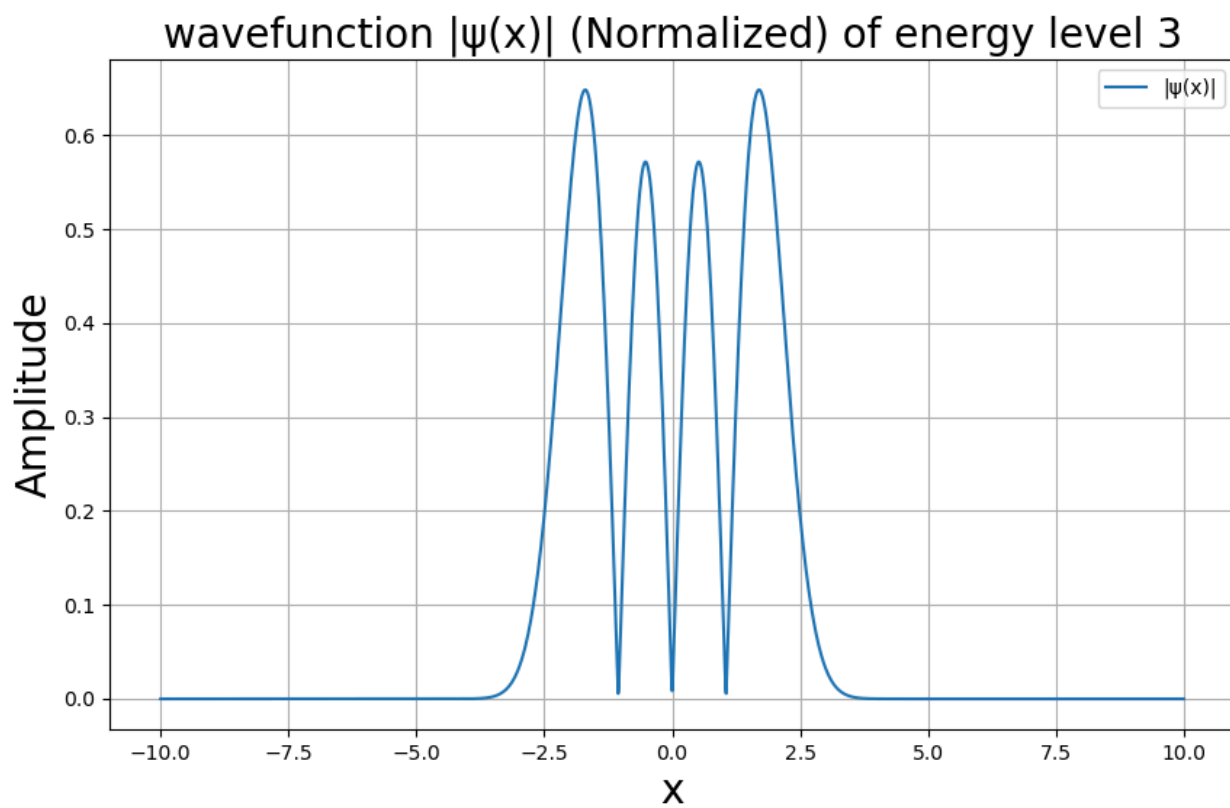
第一激发态:



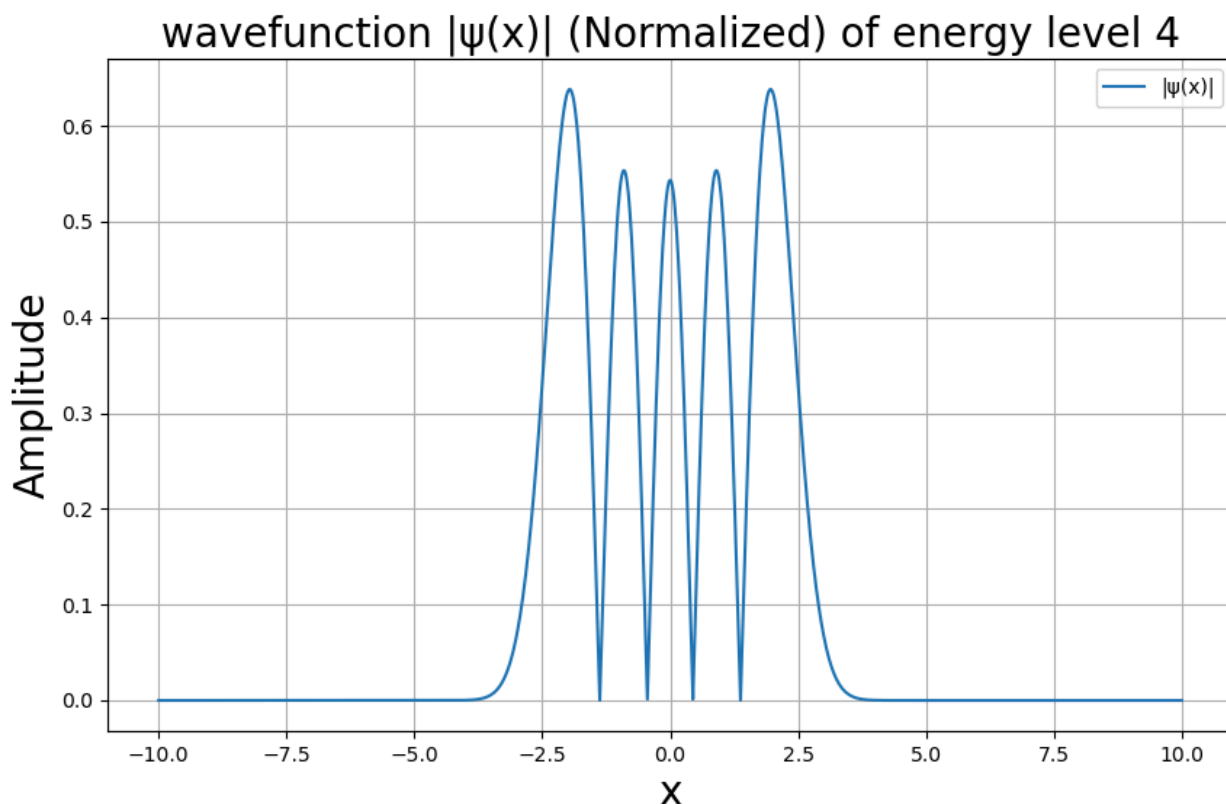
第二激发态:



第三激发态:



第四激发态:



4.和简谐振子 $v = \frac{1}{2}x^2$ 的比较

简谐振子 $v = \frac{1}{2}x^2$ 的情况下波函数是有解析解的，对于方程

$$-\frac{1}{2} \frac{d^2}{dx^2} \phi(x) + \frac{x^2}{2} \phi(x) = E \phi(x)$$

有解析解，

波函数的推导如下：

利用变系数法，波函数：

$$\phi(x) = A(x)e^{-\frac{x^2}{2}}$$

其中 $A(x) = \sum_{k=0}^{\infty} a_k x^k$

带入薛定谔方程可以得到：

$$a_{k+2} = \frac{2k + 1 - 2E}{(k + 1)(k + 2)} a_k$$

可以发现，事实上求和不能加到无穷，否则波函数不收敛，也就是存在 a_n 不为0， $a_{n+2} = 0$ 的情况：

$$a_{k+2} = \frac{2k+1-2E}{(k+1)(k+2)} a_k = 0$$

即

$$E = k + \frac{1}{2}$$

由此得到本征能量的形式：

基态能量为0.5,第一激发态能量为1.5, 以此类推。

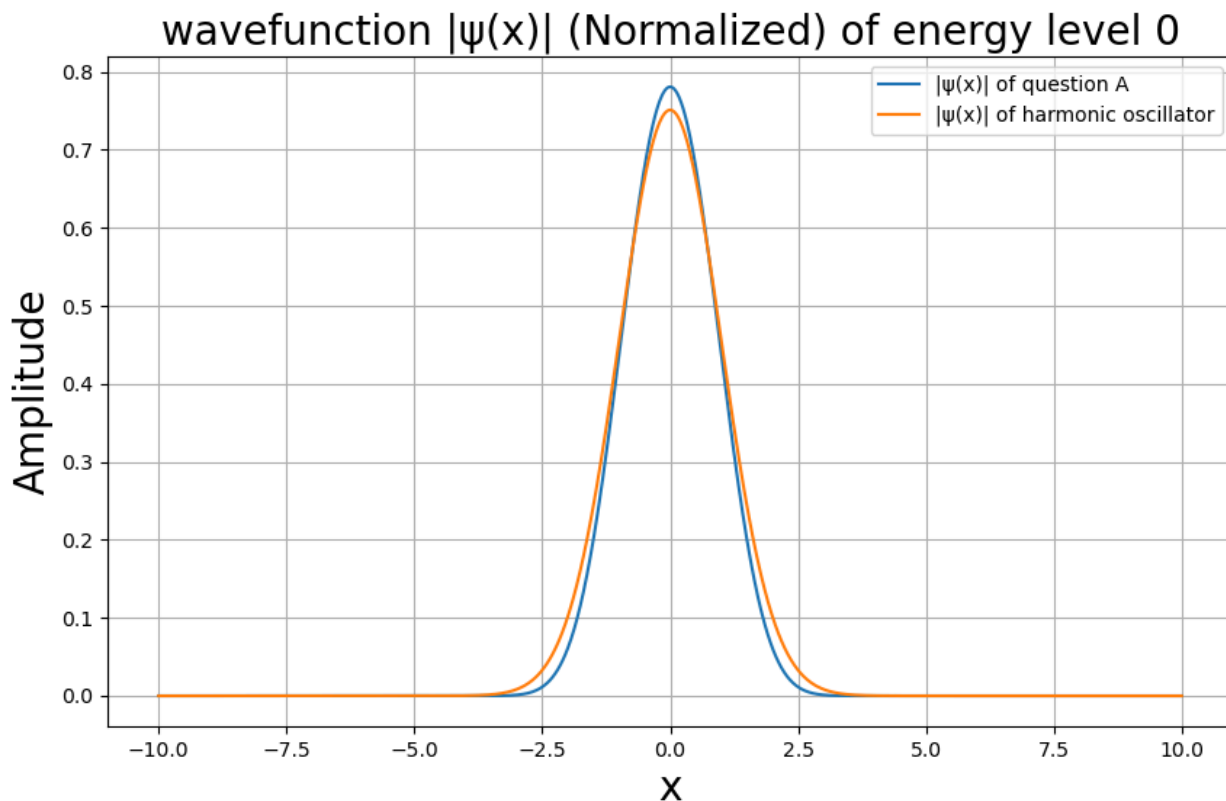
不妨取 $b_0 = 1, b + 1 = 2$,我们也得到了波函数的解析解：

$$\phi_i(x) = \sum_{k=0}^i \frac{1}{\sqrt{2^k k!} \sqrt{\pi}} e^{-\frac{x^2}{2}} A(x)$$

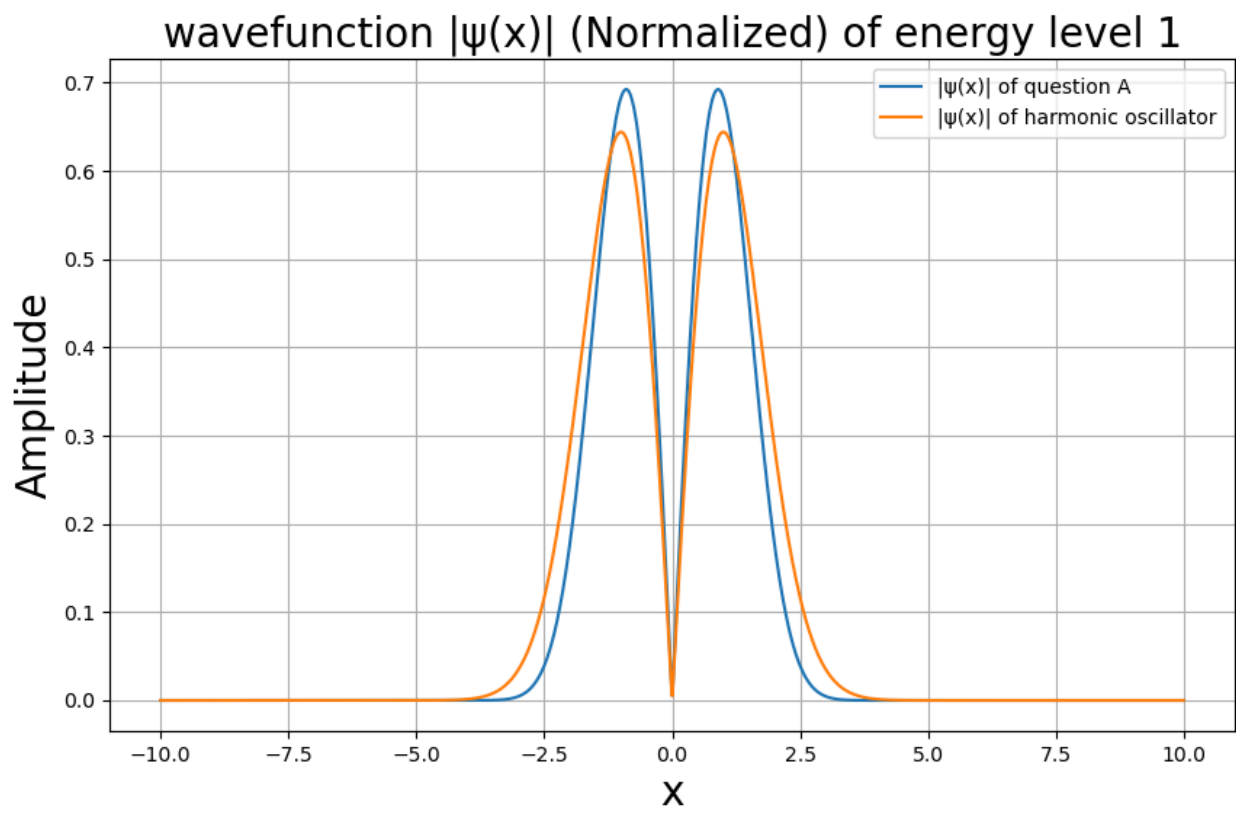
这里的A(x)的截断取决于本征能量的大小。

利用得到的解析解，画出A中所提到的势函数和简谐振子两种情况下的波函数的绝对值如图所示：

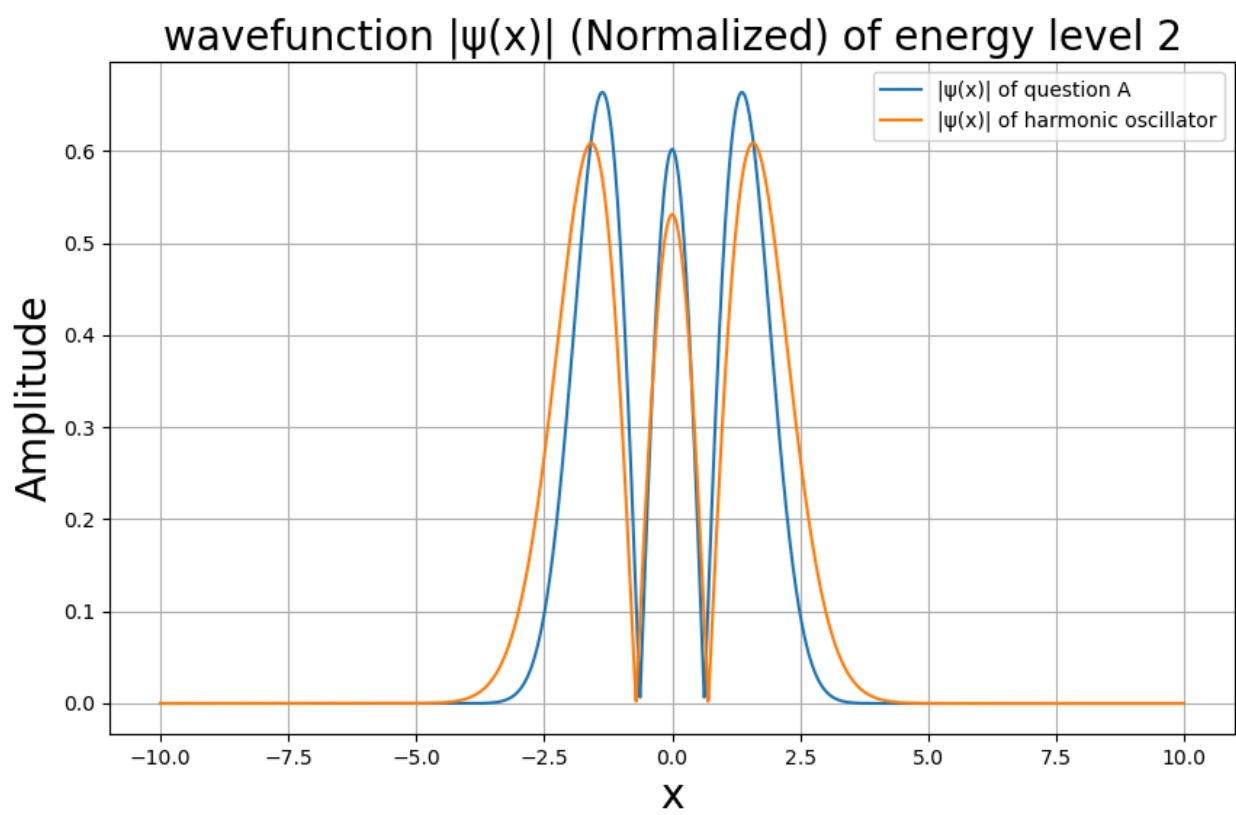
对于基态：



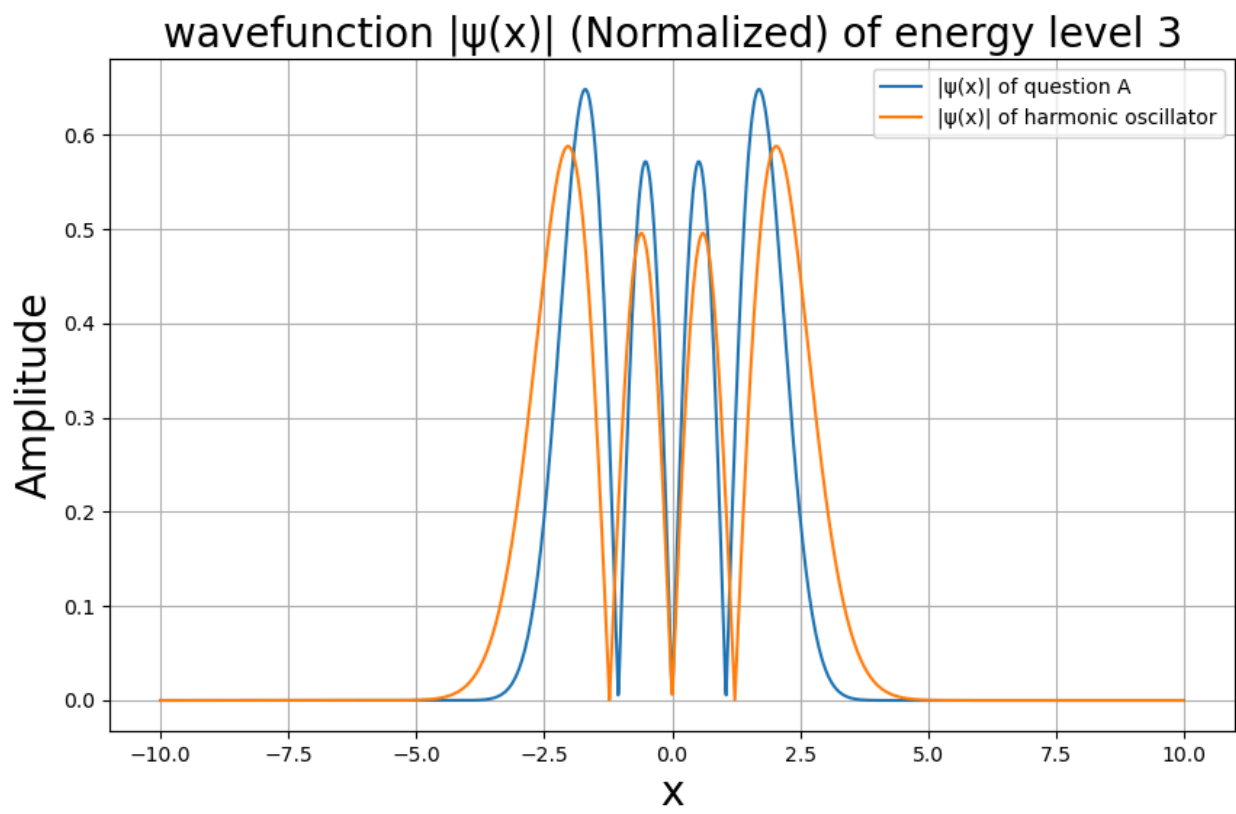
对于第一激发态：



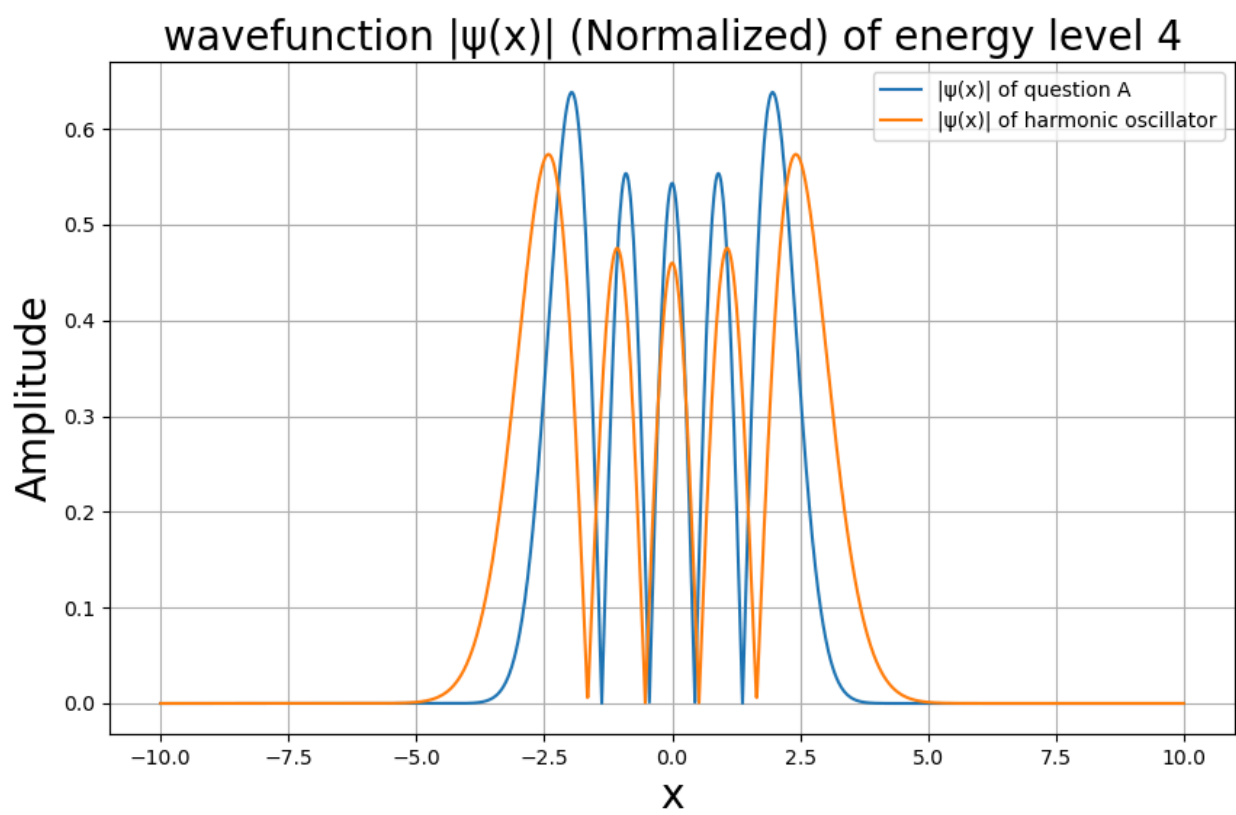
对于第二激发态：



对于第三激发态：



对于第四激发态：



我还通过前述差分的方法求出了简谐振子的本征能量作为验证：

```
[Running] python -u "e:\大二下\computational physics\homework\hw_4\code\A_chafen.py"
Energy Level 0 A:0.5594
Energy Level 0 of harmonic oscillator:0.5002
Energy Level 1 A:1.7705
Energy Level 1 of harmonic oscillator:1.5007
Energy Level 2 A:3.1403
Energy Level 2 of harmonic oscillator:2.5012
Energy Level 3 A:4.6314
Energy Level 3 of harmonic oscillator:3.5017
Energy Level 4 A:6.2237
Energy Level 4 of harmonic oscillator:4.5021

[Done] exited with code=0 in 22.379 seconds
```

由图中结果可以看出，简谐振子的本征能量确实为：

$$E_n = \frac{1}{2} + n$$

与前面理论分析符合。

二者对比可以发现：

1. 对于同为基态或者相同的激发态，A中所提到的势函数对应的本正能量更大，能级不等间隔，而且能级越大，能级间距越大。这可能是因为势函数中的 x^4 项，使得势函数比简谐振子的势函数更大且更陡峭。
2. 二者的波函数的节点数都等于对应的能级数，例如基态都只有一个节点。
3. 二者的波函数的绝对值都是关于y轴对称的，这是因为二者的势函数都是偶函数，波函数符合奇宇称或者偶宇称。
4. 对于同一个节点，A中给出的势函数的波函数的波峰位置更加向中心靠拢，波峰的高度也高于简谐振子。这是由于A中的势函数有 x^4 项，势函数更加陡峭，粒子有更大的概率分布在中心位置

5.其他方法求解

前面在（1）中已经给出多种方法及具体思路，代码在附录。这里直接分析各种方法的优劣：

算法一：打靶法（射击法）

优点：

- 对于单个本征值（尤其是基态或低阶激发态）计算效率更高。
- 直接利用了波函数的简便性（偶函数/奇函数），降低了计算的复杂度。
- 很容易实现，尤其是结合龙格库塔法的数值稳定性强。

缺点：

- 对于高阶激发状态，割线法可能需要多次迭代，且容易跳过某些本征值。
- 如果最初猜测的本征值较低，可能导致收敛缓慢或失败。
- 依赖最终条件设定，可能存在人为偏差。
- 对于非精确势场或复杂势函数，使用此方法时可能会比较麻烦。
- 适合西南低阶本征值和势简单场问题（如谐振子、方势垒等）。
- 不适用于多维势场或快速振荡的波函数。

算法二：匹配法（Matching Method）

优点：

- 本征值寻求更稳定，尤其在高层本征态时更容易找到正确的解。
- 通过精确性和匹配条件，避免了在高阶激发状态中割线法可能出现的“跳过本征值”问题。
- 使用了匹配函数 $f(E)$ ，对本征值的判定更具有洞察力，适用于复杂趋势场。

缺点：

- 数值实现比较复杂，需要两次积分并搭配割线法。
- 如果积分范围不充分（即 L 选得不够大），配合任务可能扩大。
- 初始本征值猜测较低时，可能导致更多的积分和迭代次数。
- 更适合复杂趋势场、非精确趋势场，或者高阶本征态的精确仿真。
- 对于基态也表现良好，但可能比打靶法稍微慢一些。

算法三：矩阵本征值法（Matrix Eigenvalue Method）

优点：

- 该方法将问题离散化，转化为标准的矩阵本征值问题，数值稳定性强。
- 矩阵的稀疏性可以利用高效的稀疏矩阵算法（如`scipy.sparse.linalg`）加速流程图。
- 一次可得到所有本征值和本征函数，无需再单独猜测每个本征值。
- 数值稳定性优异，特别适合计算高阶本征值。

缺点：

- 空间网格划分太粗会导致不准确，太细则增加计算量。
- 对于非常陡峭的势能函数（如深势阱），需要更密集的网格点。

B.格点上的单粒子动力学

算法解析：

该题本质上是一维相同格点的紧束缚模型。仅允许相邻格点之间的跳跃。

出于提高运算速度的考虑，可以使用哈密顿量的矩阵形式：

$$H = \begin{bmatrix} 0 & -1 & 0 & 0 & \cdots & 0 & -1 \\ -1 & 0 & -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 0 & -1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 0 & -1 \\ -1 & 0 & 0 & \cdots & 0 & -1 & 0 \end{bmatrix}$$

这里的对角线为0，次对角线为-1，表示相邻格点之间的跳跃。

由薛定谔方程

$$i \frac{\partial}{\partial t} \psi(t) = H \psi(t)$$

可得

$$\psi(t) = e^{-iHt} \psi(0)$$

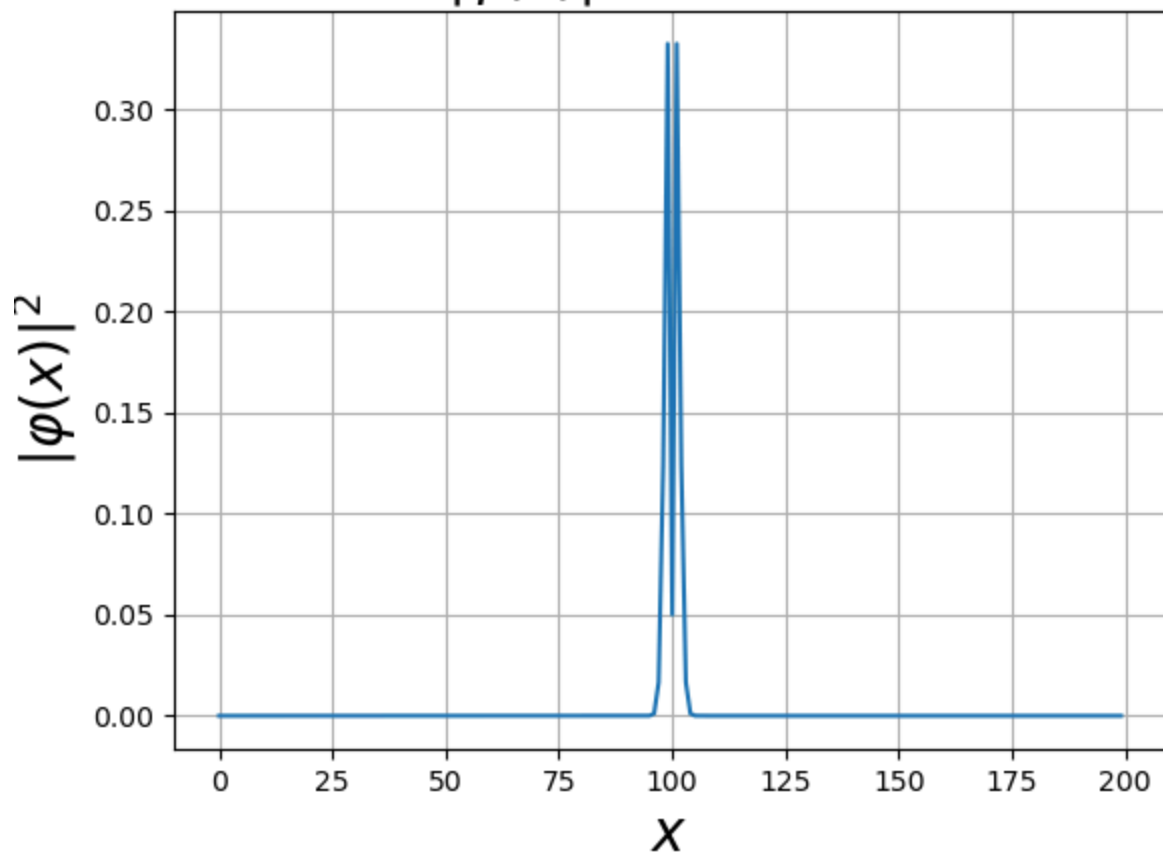
其中这里的H即为上面的矩阵形式。

1.画出几个时刻的粒子密度分布

通过前述算法，

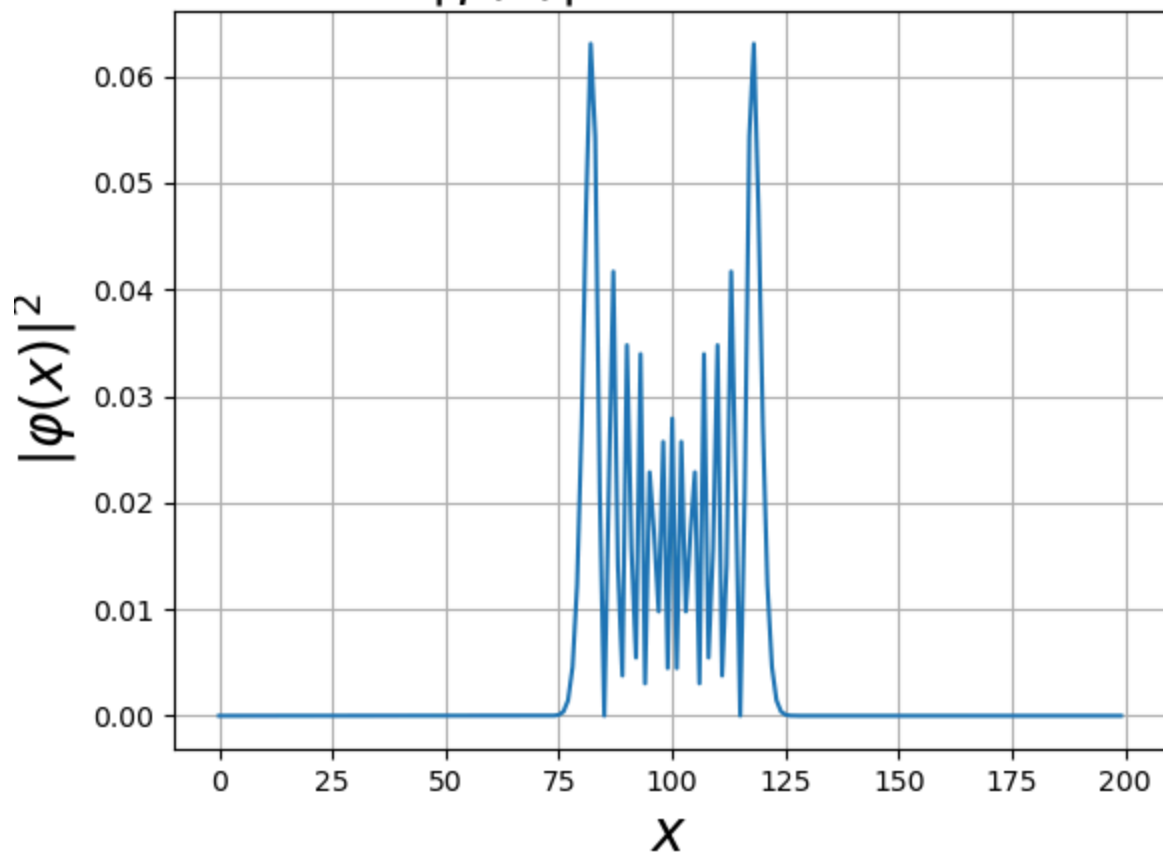
t=1:

$|\varphi(x)|^2$ at $t = 1.0$



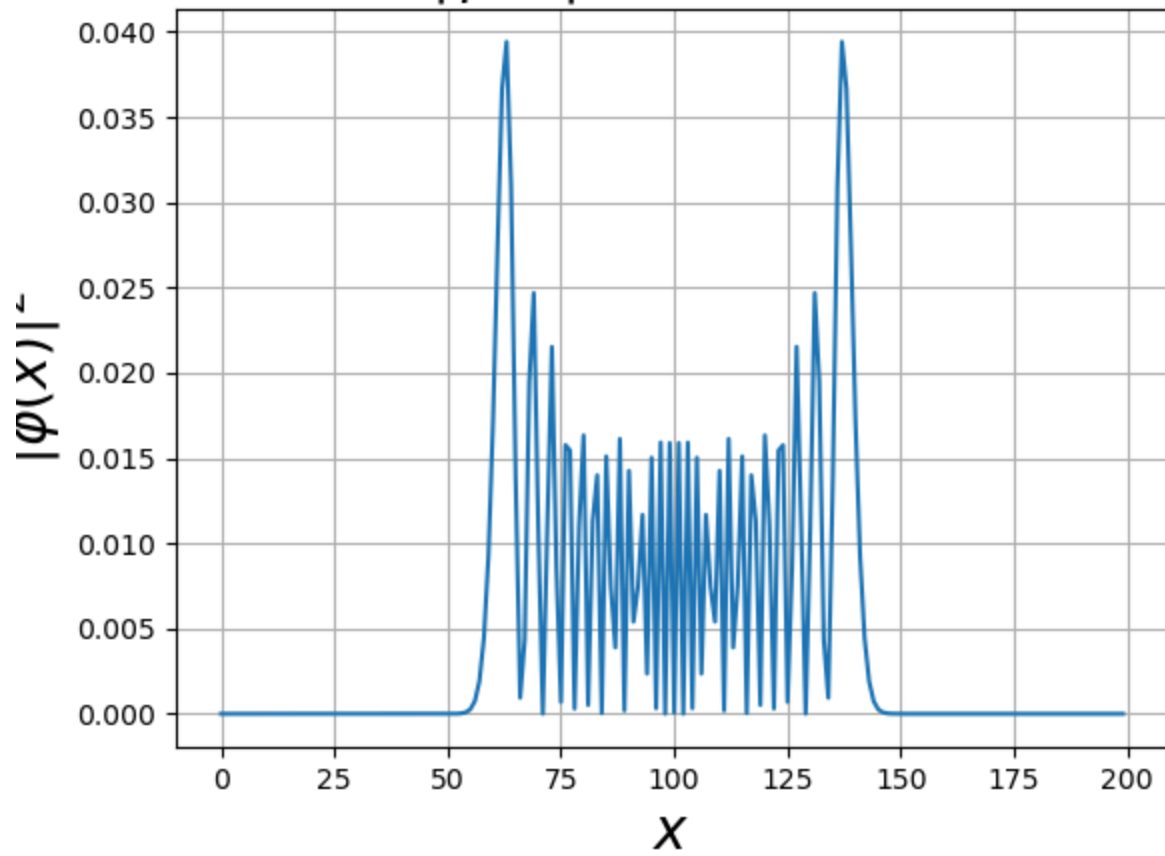
t=10:

$|\varphi(x)|^2$ at $t = 10.0$

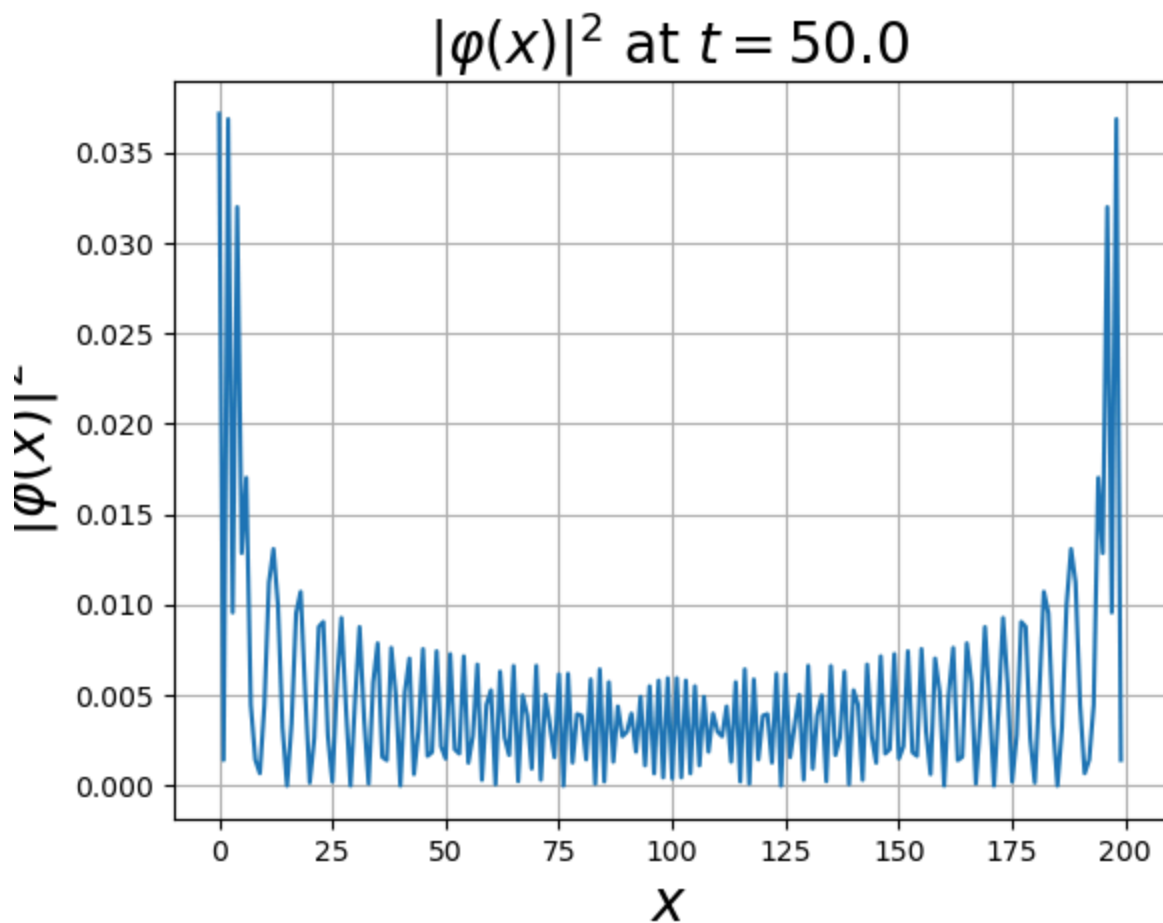


t=20:

$|\varphi(x)|^2$ at $t = 20.0$



t=50:



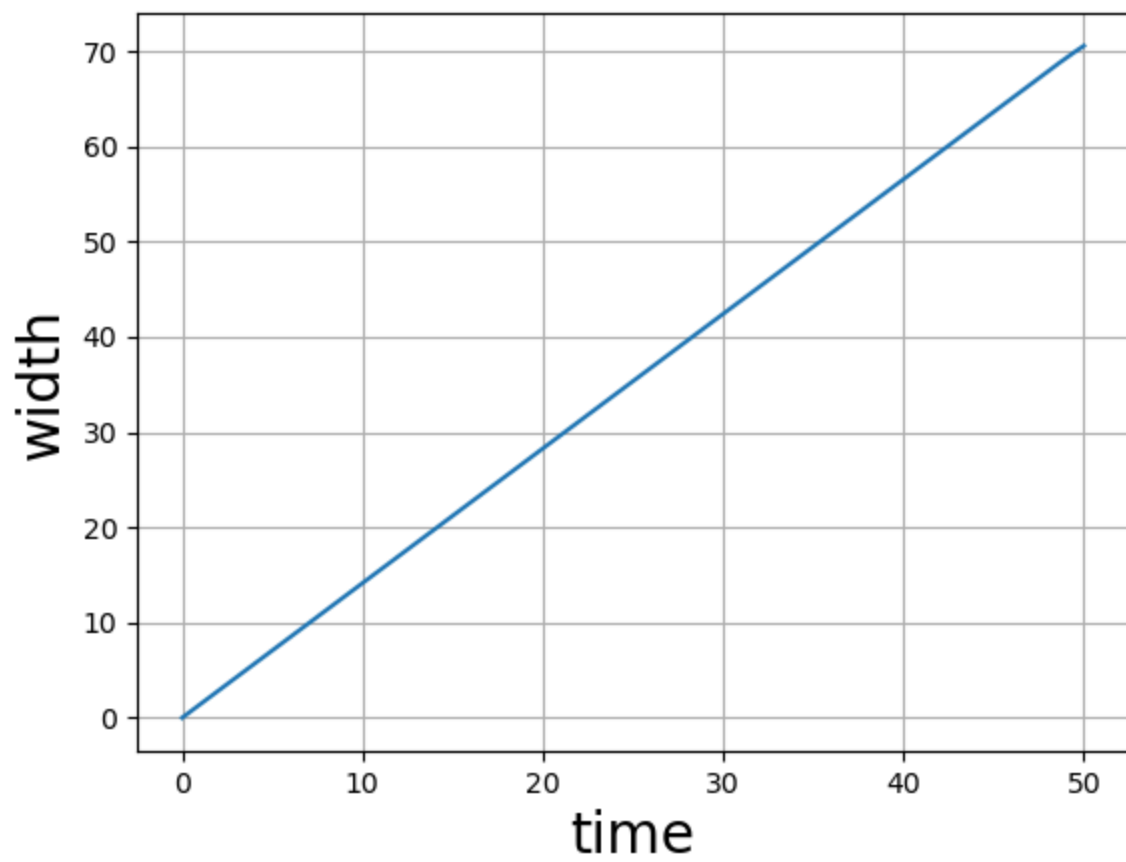
2.画出波包宽度随时间的变化

波包宽度定义为：

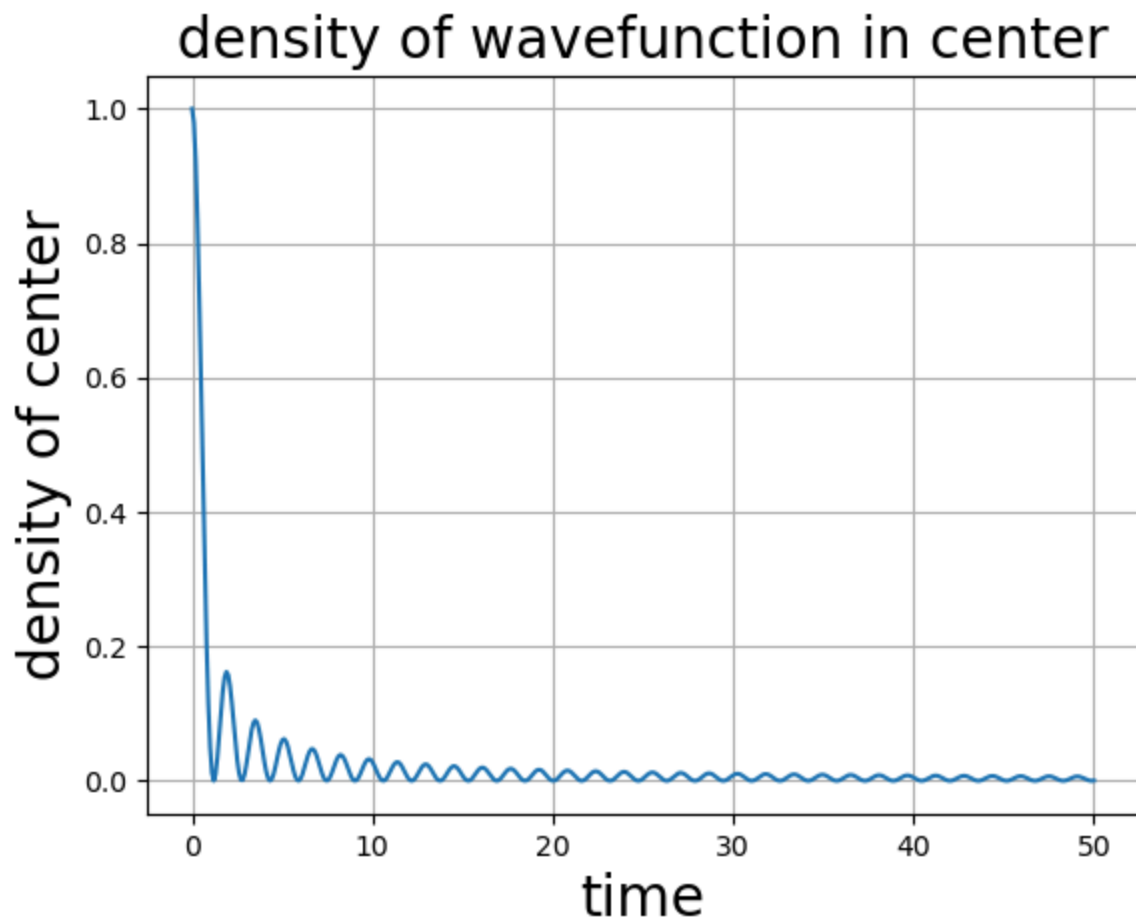
$$w(t) = \sqrt{\sum_i (i - i_0)^2 \rho_i(t)}$$

得到时间范围内的波包宽度随时间的变化为：

width of wavefunction



3.画出求解时间内起始点粒子密度随时间的变化



4.解释现象

现象一：

可以观察得到以下现象：

1. 初始时只有中心有波函数，随着时间的演化，波函数逐渐向两侧扩散，波函数的波包宽度逐渐增大
2. 在不同时刻，波函数都表现出明显的锯齿状
3. 波函数扩散的速度是均匀的，约为时间的2倍，如 $t=20$ 时刚好扩散到40和160处

量子力学的解释如下：

1. 初始时，粒子集中在某一格点，使得位置的不确定性较小，所以初始的时候动量的不确定性非常大，粒子没办法稳定在一个中心点保持不动，所以会扩散到两侧。初始时 Δx 为0，对应于很多个动量本征值的叠加，这些本征值之间发生干涉，导致波包扩散
2. 锯齿状的产生是由于初始状态的“不均匀性”引发的。如果初始时在各个格点出现的概率相同，则波函数会稳定在一个水平。而初始的时候只有中心格点有粒子，跳跃又是以1为周期的，就会产生在某一个时刻奇数点都非常高，偶数点都非常低，就出现了锯齿状

3. 波包宽度随时间线性扩散的线性在现象二中分析

特别注意前面的计算中，H矩阵的左下和右上的值都是-1，相当于认为格点是循环的，保证了没有粒子溢出。

对于L趋于正无穷的情况，即无穷长的格点，波包会无线扩散，最终各个格点处的波函数为0。这符合自由波函数的性质，因为在哈密顿量没有出现势函数，相当于各个格点处的势能相同。但是在有限的晶格下，波函数会绕回，与自身发生干涉。

现象二

可以观察得到以下现象：

波包宽度随时间演化线性增长

查阅资料可知，该情况下波包是高斯分布：

在二次能带近似下(动量 $k \approx 0$),能量色散关系为：

$$E(k) \approx -2t + tk^2.$$

在时间t的演化过程中，从薛定谔方程可得：

$$\psi(k, t) = \psi(k, 0)e^{-iE(k)t} = \psi(k, 0)e^{-i(-2t+tk^2)t}.$$

对于位置空间中的波函数 $\psi(x, t)$,由动量空间波函数的傅里叶变换得到：

$$\psi(x, t) = \int_{-\infty}^{\infty} \psi(k, 0)e^{i(kx-E(k)t)} dk.$$

将动量空间波函数代入，结合 $E(k) \approx tk^2$,得：

$$\psi(x, t) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{k^2}{2\sigma_k^2}\right) \cdot e^{i(kx-tk^2)} dk.$$

$$\psi(x, t) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \int_{-\infty}^{\infty} \exp\left[-\frac{k^2}{2\sigma_k^2} + i(kx) - i(tk^2)\right] dk.$$

整理平方项，将动量k-相关的项合并为一个平方式：

$$\psi(x, t) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \int_{-\infty}^{\infty} \exp\left[-\left(\frac{1}{2\sigma_k^2} + it\right)k^2 + i(kx)\right] dk.$$

此时积分项变为一个高斯积分（指数中 k^2 为二次项），其解为高斯分布的形式：

$$\psi(x, t) \propto \exp\left(-\frac{x^2}{2\sigma_x^2(t)}\right),$$

其中位置空间的宽度 $\sigma_x(t)$ 随时间演化。该波包在位置空间呈高斯形状。

从上述推导中可以看出，高斯波函数在动量空间中的宽度为 σ_k ，其随时间在位置空间中的传播速度与二次色散关系密切相关。当动量频谱确定后，位置空间的波包宽度 $\sigma_x(t)$ 与时间 t 的关系如下：

$$\sigma_x(t) = \frac{1}{\sigma_k} \cdot \sqrt{1 + (2t\sigma_k^2)^2}.$$

对于时间较短的情况(初期演化阶段，当 $t \rightarrow 0$)，我们可以近似忽

略 $(2t\sigma_k^2)^2$ 中的平方项，此时：

$$\sigma_x(t) \approx \frac{1}{\sigma_k} \cdot (2t\sigma_k^2) = 2t\sigma_k.$$

因此，在初期阶段，波包宽度 $\sigma_x(t)$ 与时间 t 呈现线性关系：

$$w(t) = \sigma_x(t) \propto t.$$

这也就是为什么波包宽度初期会随时间线性增长，体现经典动力学中自由粒子的传播特征。

现象三

观察可知中心格点的波函数密度有以下现象：

1. 呈现出明显的锯齿状、波浪状
2. 整体上迅速下降，最终趋于0，在 t 接近50时展现出小幅的周期性的振荡

解释如下：

1. 中心格点是对称的传播起点。粒子以相等的几率扩散到两侧，对称传播的部分波函数中间形成周期性的波动，振荡幅度逐渐减小。这是晶格离散系统特有的现象，在连续系统中不会出现同样显著的锯齿波形。
2. 初始时，粒子集中在中心格点，因此密度较高。但是，随着时间增长，整个波包开始扩展，分散到其他晶格位置，导致中心点的粒子密度显著下降。在无外场的自由传播中，晶格的对称条件保证粒子均匀扩散，最终使中心格点粒子密度趋于零。
3. 晶格的有限尺寸导致波函数传播到晶格边界（两边）并返回，从而再次进入相位叠加形成局部的强化或削弱位置
4. 总的来说就是，由于各个晶格本质上是一样的，以及波函数的归一化，所以中心晶格的波函数密度不会真的等于0。由于在一侧溢出的波函数可以在另一侧重新进入晶格系统，就会出现一个周期性的振荡，这就像向湖面投掷一颗石子，如果在湖边缘溢出的波可以重新在中心出现，那么湖面就会出现一个连绵不断的水波。

附录

A 割线法和龙格库塔法结合

```
import numpy as np

def potential(x:float) -> float:
    return 0.5 * x**2 + 0.25 * x**4

def func(phi: np.ndarray, x: float, E: float) -> np.ndarray:
    dphi = phi[1]
    ddphi = 2 * (potential(x) - E) * phi[0]
    return np.array([dphi, ddphi])

def runge_kutta_4(f, phi0: np.ndarray[float], x0: float, xf: float, dx: float, e: float) -> np.ndarray:
    x = x0
    phip = phi0.astype(float)
    length = int((xf - x0) / dx) + 1
    trajectory = np.zeros((length, 3))
    time = 0
    while x < xf:
        phi, dphi = phip[0], phip[1]
        trajectory[time, :] = np.array([x, phi, dphi])
        k1 = dx * f(phip, x, e)
        k2 = dx * f(phip + 0.5 * k1, x + 0.5 * dx, e)
        k3 = dx * f(phip + 0.5 * k2, x + 0.5 * dx, e)
        k4 = dx * f(phip + k3, x + dx, e)
        phip += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        x += dx
        time += 1
    return trajectory

def interative(E0: float, E1: float, L: float, max_iter: int, dx: float, phi0: np.ndarray, tolerance: float):
    x0 = 0
    xf = L
    for i in range(max_iter):
        phi_E0 = runge_kutta_4(func, phi0, x0, xf, dx, E0)[-1, 1] # 获取 psi(L)
        phi_E1 = runge_kutta_4(func, phi0, x0, xf, dx, E1)[-1, 1] # 获取 psi(L)

        if abs(phi_E1) < tolerance:
```

```

        return E1, runge_kutta_4(func, phi0, x0, xf, dx, E1)

    E_new = E1 - phi_E1 * (E1 - E0) / (phi_E1 - phi_E0)
    E0, E1 = E1, E_new

    raise ValueError("111")

def normalize_wavefunction(trajectory):
    x_vals = trajectory[:, 0]
    psi_vals = trajectory[:, 1]
    norm = np.sqrt(np.trapz(psi_vals**2, x_vals))
    return psi_vals / norm # 返回归一化的波函数

if __name__ == "__main__":
    # 基态, 设x=0处波函数为1, 导数为0
    phi0 = np.array([1.0, 0.0])
    E0 = 0
    E1 = 0.2
    L = 90
    max_iter = 100
    dx = 0.01
    tolerate = 1e-6
    E, trajectory = iterative(E0, E1, L, max_iter, dx, phi0, tolerate)
    psi = normalize_wavefunction(trajectory)
    print(f"ground state energy:{E}")
    print(f"param of normal:{np.trapz(psi**2, trajectory[:, 0])}")

    print("trajectory:")
    print(trajectory[500:])

```

A 差分

```
import numpy as np
import matplotlib.pyplot as plt

def matrix(L:float,dx:float)->tuple[np.ndarray,np.ndarray]:
    N = int(2*L/dx)
    x = np.linspace(-L,L,N)
    # 势能项加到对角线上
    V = 0.5 * x**2 + 0.1 * x**4
    H = np.zeros((N, N))
    for i in range(N):
        if i > 0:
            H[i, i-1] = -1 / (2 * dx**2)
            H[i, i] = V[i] + 1 / (dx**2)
        if i < N - 1:
            H[i, i+1] = -1 / (2 * dx**2)
    return H,x

def matrix_jianxie(L:float,dx:float)->tuple[np.ndarray,np.ndarray]:
    N = int(2*L/dx)
    x = np.linspace(-L,L,N)
    # 势能项加到对角线上
    V = 0.5 * x**2
    H = np.zeros((N, N))
    for i in range(N):
        if i > 0:
            H[i, i-1] = -1 / (2 * dx**2)
            H[i, i] = V[i] + 1 / (dx**2)
        if i < N - 1:
            H[i, i+1] = -1 / (2 * dx**2)
    return H,x

def eigen_and_norm(H:np.ndarray, x:np.ndarray)->tuple[np.ndarray,np.ndarray]:
    eigvals, eigvecs = np.linalg.eigh(H)
    for n in range(eigvecs.shape[1]):
        norm = np.sqrt(np.trapz(eigvecs[:, n]**2, x))
        eigvecs[:, n] /= norm
    eigvals = np.round(eigvals, 4) # 精确到1e-4
    return eigvals, eigvecs

def plot_wavefunction_abs(x:np.ndarray,wavefunction:np.ndarray,i:int)->None:
    plt.figure(figsize=(10, 6))
```

```

plt.plot(x,np.abs(wavefunction), label=f" $|\psi(x)|$ ")
plt.xlabel("x",fontsize=20)
plt.ylabel("Amplitude",fontsize=20)
plt.title(f"wavefunction  $|\psi(x)|$  (Normalized) of energy level {i}",fontsize=20)
plt.legend(loc='upper right')
plt.grid()

```

```

path = f"figure/A_chafen_{i}.png"
plt.savefig(path)
plt.show()

```

```

def plot_wavefunction_compare(x:np.ndarray,wavefunction:np.ndarray,wavefunction1:np.ndarray,i:i
plt.figure(figsize=(10, 6))
plt.plot(x,wavefunction, label=f" $|\psi(x)|$  of question A")
plt.plot(x,wavefunction1, label=f" $|\psi(x)|$  of harmonic oscillator")
plt.xlabel("x",fontsize=20)
plt.ylabel("Amplitude",fontsize=20)
plt.title(f"wavefunction  $\psi(x)$  (Normalized) of energy level {i}",fontsize=20)
plt.legend(loc='upper right')
plt.grid()

path = f"figure/A_chafen_compare_{i}.png"
plt.savefig(path)
plt.show()

```

```

if __name__ == "__main__":
    L = 10
    dx = 0.01
    H ,x= matrix(L,dx)
    eigvals,eigvecs = eigen_and_norm(H,x)
    # ground_state_energy = eigvals[0]
    # ground_state_wavefunction = eigvecs[:, 0]
    # plot_wavefunction_abs(x,ground_state_wavefunction,0)
    # print(f"Ground State Energy: {ground_state_energy}")
    # for i in range(1, 5):
    #     energy = eigvals[i]
    #     wavefunction = eigvecs[:, i]
    #     print(f"Energy Level {i}: {energy}")
    #     plot_wavefunction_abs(x,wavefunction,i)
    H1,x = matrix_jianxie(L,dx)
    eigvals1,eigvecs1 = eigen_and_norm(H1,x)
    for i in range(0, 5):

```



```
energy = eigvals[i]
energy1 = eigvals1[i]
wavefunction0 = eigvecs[:, i]
wavefunction1 = eigvecs1[:, i]
print(f"Energy Level {i} A:{energy}")
print(f"Energy Level {i} of harmonic oscillator:{energy1}")
plot_wavefunction_compare(x,wavefunction0,wavefunction1,i)
```

B 第一题

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import diags
from scipy.sparse.linalg import expm
from scipy.sparse import csr_matrix

def H_matrix(L: int) -> np.ndarray:
    # 构造哈密顿量
    H = np.zeros((L, L))
    for i in range(L):
        if i > 0:
            H[i, i-1] = -1
        if i < L - 1:
            H[i, i+1] = -1
    # 周期边界条件
    H[0, L-1] = H[L-1, 0] = -1
    return H

def phi(L: int, t: float, phi0: np.ndarray) -> np.ndarray:
    H = csr_matrix(H_matrix(L))
    U_t = expm(-1j * t * H)
    return U_t @ phi0

def plot_wavefunction(L: int, t: float, phi_wave: np.ndarray) -> None:
    # 绘制波函数
    x = np.arange(0, L)
    plt.plot(x, np.abs(phi_wave)**2)
    plt.xlabel(r'$x$', fontsize=20)
    plt.ylabel(r'$|\varphi(x)|^2$', fontsize=20) # 注意 ``\varphi`` 更接近物理学中常用的波函数符号
    plt.title(r'$|\varphi(x)|^2$ at $t = %.1f$' % t, fontsize=20)
    plt.grid(True)
    path = f"figure/B_1_{t}.png"
    plt.savefig(path)
    plt.show()

if __name__ == "__main__":
    time = [90.0]
    L=200
    phi0 = np.zeros(L, dtype=complex)
    phi0[100] = 1.0
    for t in time:
```

```
phi_t = phi(L,t,phi0)
plot_wavefunction(L,t,phi_t)
```

B 第二题

```
from B_1 import H_matrix,phi,plot_wavefunction
import numpy as np
import matplotlib.pyplot as plt

def calculate_width(phi_wave: np.ndarray, x: np.ndarray) -> float:
    # 计算波包宽度 w(t)
    rho = np.abs(phi_wave)**2 # 粒子密度
    width = np.sqrt(np.sum((x - 100)**2 * rho)) # 计算加权平均距离的平方根
    return width

if __name__ == "__main__":
    time = np.linspace(0, 50, 500)
    width = []
    L = 200
    phi0 = np.zeros(L, dtype=complex)
    phi0[100] = 1.0
    x = np.arange(0,L)
    for t in time:
        phi_t = phi(L,t,phi0)
        width.append(calculate_width(phi_t,x))
    width = np.array(width)
    plt.plot(time,width)
    plt.xlabel("time",fontsize=20)
    plt.ylabel("width",fontsize=20)
    plt.title("width of wavefunction",fontsize=20)
    plt.grid(True)
    path = f"figure/B_2.png"
    plt.savefig(path)
    plt.show()
```

B 第三题

```
from B_1 import H_matrix, phi, plot_wavefunction
import numpy as np
import matplotlib.pyplot as plt

if __name__ == '__main__':
    time = np.linspace(0, 50, 500)
    rho = []
    L=200
    phi0 = np.zeros(L, dtype=complex)
    phi0[100] = 1.0
    for t in time:
        phi_t = phi(200, t, phi0)
        rho.append(np.abs(phi_t[100])**2)
    rho = np.array(rho)
    plt.plot(time, rho)
    plt.xlabel("time", fontsize=20)
    plt.ylabel("density of center", fontsize=20)
    plt.title("density of wavefunction in center", fontsize=20)
    plt.grid(True)
    path = f"figure/B_3.png"
    plt.savefig(path)
    plt.show()
```