1. 使用 Euler method, midpoint method, RK4 method, Euler-trapezoidal method 求解单摆的运动。单摆的运动方程及总能量为（将单摆的最低点设为重力势能的零点）：

$$\theta + \frac{l}{g}\ddot{\theta} = 0, E = \frac{mgl}{2}\theta^2 + \frac{ml^2}{2}\dot{\theta}$$

无量纲化，令$\frac{l}{g} = 1, \frac{ml^2}{2} = 1$，原方程化为：

$$\theta + \ddot{\theta} = 0, E = \theta^2 + \dot{\theta}^2$$

定义$\theta = y_1, \frac{d\theta}{dt} = y_2$，我们可以得到：

$$\frac{dy_1}{dt} = y_2, \frac{dy_2}{dt} = -y_1$$

定义$Euler(T, h)$函数来实现 Euler method，T为我们要计算的时间范围，h为 Euler method 中的步长 step size。

$N = int(T/h)$为迭代的总次数。初始化四个列表$t_points, y1_points, y2_points, energy$用来存放计算出的时间$t$值、$y_1 = \theta$值、$y_2 = \dot{\theta}$值和能量$E$值。初值设置为：$y = [y_1, y_2] = [\frac{\pi}{40}, 0]$，即初始角度$\theta_0 = 4.5°$，初始角速度$\dot{\theta}_0 = 0$。

在函数内部通过 for 循环完成迭代。先往三个列表里添加t, y_1, y_2值，然后再添加能量

$$E = \theta^2 + \dot{\theta}^2$$

在循环的末尾修改y：

$$y = \left[y_1 + h \cdot \frac{dy_1}{dt}, y_2 + h \cdot \frac{dy_2}{dt}\right] = [y_1 + hy_2, y_2 - hy_1]$$

然后进入下一个循环。

在所有循环结束后，还需要把最后一次迭代得到的点添加入列表。然后绘制$\theta - t$图和$E - t$图。

```python
def Euler(T, h):
# We set l/g = 1, ml^2/2 = mgl/2 = 1
# θ = y1, dθ/dt = y2
# dy1/dt = y2, dy2/dt = -y1
    N = int(T / h)
    t_points = []
    y1_points = []
    y2_points = []
    energy = []
    y = [pi/40, 0]

    for i in range(N):
        t_points.append(h * i)
        y1_points.append(y[0])
        y2_points.append(y[1])
        energy.append(y[0]**2 + y[1]**2)
        y = [y[0] + y[1]*h, y[1] - y[0]*h]

    t_points.append(h * N)
    y1_points.append(y[0])
    y2_points.append(y[1])
    energy.append(y[0]**2 + y[1]**2)

    plt.plot(t_points, y1_points)
    plt.show()
    plt.plot(t_points, energy)
    plt.show()

    return t_points, y1_points, y2_points
```

其他的函数：$midpoint(T,h), RK4(T,h), Euler_trapezoidal(T,h)$分别定义了 midpoint method, RK4 method 以及 Euler-trapezoidal method。函数的主体部分几乎完全一致，只修改了迭代中的计算新的y的部分。

```python
def midpoint(T, h):
    N = int(T / h)
    t_points = []
    y1_points = []
    y2_points = []
    energy = []
    y = [pi/40, 0]

    for i in range(N):
        t_points.append(h * i)
        y1_points.append(y[0])
        y2_points.append(y[1])
        energy.append(y[0]**2 + y[1]**2)
        y_temp = [y[0] + (y[1]*h)/2, y[1] - (y[0]*h)/2]
        y = [y[0] + y_temp[1]*h, y[1] - y_temp[0]*h]

    t_points.append(h * N)
    y1_points.append(y[0])
    y2_points.append(y[1])
    energy.append(y[0]**2 + y[1]**2)

    plt.plot(t_points, y1_points)
    plt.show()
    plt.plot(t_points, energy)
    plt.show()

    return t_points, y1_points, y2_points
```

```python
def RK4(T, h):
    N = int(T / h)
    t_points = []
    y1_points = []
    y2_points = []
    energy = []
    y = [pi/40, 0]

    for i in range(N):
        t_points.append(h * i)
        y1_points.append(y[0])
        y2_points.append(y[1])
        energy.append(y[0]**2 + y[1]**2)

        k1 = [y[1], -y[0]]
        k2 = [y[1] + (h/2)*k1[1], -(y[0] + (h/2)*k1[0])]
        k3 = [y[1] + (h/2)*k2[1], -(y[0] + (h/2)*k2[0])]
        k4 = [y[1] + h*k3[1], -(y[0] + h*k3[0])]

        temp = [(a + 2*b + 2*c + d) / 6 for a, b, c, d in zip(k1, k2, k3, k4)]
        y = [y[0] + temp[0]*h, y[1] + temp[1]*h]

    t_points.append(h * N)
    y1_points.append(y[0])
    y2_points.append(y[1])
    energy.append(y[0]**2 + y[1]**2)

    plt.plot(t_points, y1_points)
    plt.show()
    plt.plot(t_points, energy)
    plt.show()

    return t_points, y1_points, y2_points
```

对于$midpoint(T,h)$：

$$y_{temp} = \left[y_1 + \frac{h}{2}\cdot\frac{dy_1}{dt}, y_2 + \frac{h}{2}\cdot\frac{dy_2}{dt}\right] = \left[y_1 + \frac{h}{2}\cdot y_2, y_2 - \frac{h}{2}\cdot y_1\right]$$

$$y = [y_1 + hy_{temp2}, y_2 - hy_{temp1}]$$

对于$RK4(T,h)$：

$$k_1 = [y_2, -y_1], k_2 = [y_2 + \frac{h}{2}\cdot k_{12}, -\left(y_1 + \frac{h}{2}\cdot k_{11}\right)]$$

$$k_3 = [y_2 + \frac{h}{2}\cdot k_{22}, -\left(y_1 + \frac{h}{2}\cdot k_{21}\right)], k_4 = [y_2 + h\cdot k_{32}, -(y_1 + h\cdot k_{31})]$$

$$temp = \Delta y = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), y = [y_1 + \Delta y_1, y_2 + \Delta y_2]$$

对于$Euler_trapezoidal(T,h)$，除了和$Euler(T,h)$一样的常规的 Euler method，即 predictor part，还加了一段代码执行 Trapezoidal rule。

```python
        # Trapezoidal rule:
        for j in range(iteration):
            y_derivative = [(y[1] + y_temp[1]) / 2, -(y[0] + y_temp[0]) / 2]
            y_temp = [y[0] + y_derivative[0]*h, y[1] + y_derivative[1]*h]
        y = y_temp
```

$$y_{derivative} = \frac{dy}{dx} = \left[\frac{y_2 + y_2'}{2}, -\frac{y_1 + y_1'}{2}\right]$$

$$y = [y_1 + h\frac{dy_1}{dx}, y_2 + h\frac{dy_2}{dx}]$$

在每个循环中再进行三次 Trapezoidal rule 迭代，这就是 corrector part。

这样就完成了四个函数的编写，在$main()$函数中分别测试四个函数，选取一致的参数$T = 50, h = 0.001$。

```python
def main():
    T = 50
    h = 0.001
    Euler(T, h)
    midpoint(T, h)
    RK4(T, h)
    Euler_trapezoidal(T, h)
```

得到的结果如图所示：

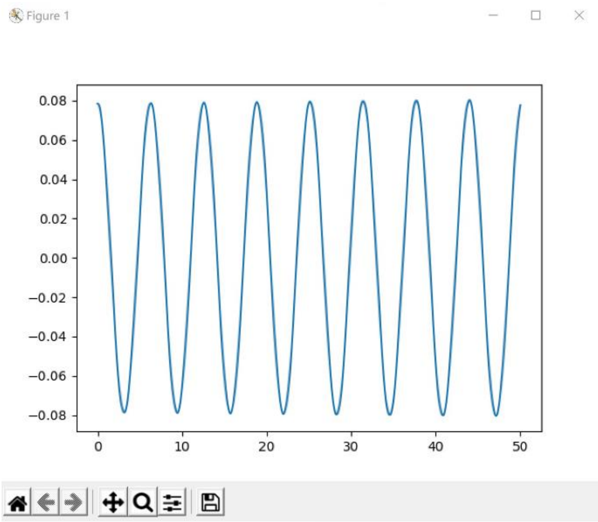


图 1 Euler method $\theta - t$图

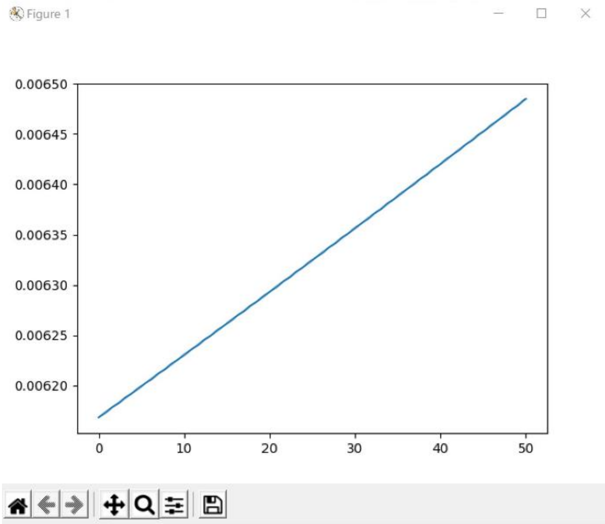


图 2 Euler method $E - t$图

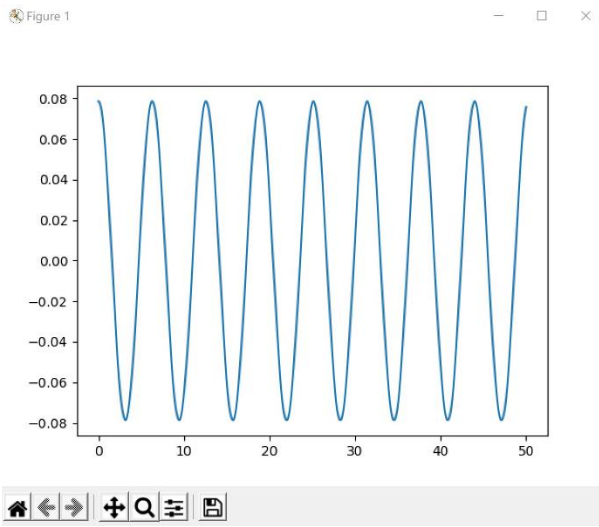


图 3 midpoint method $\theta - t$图

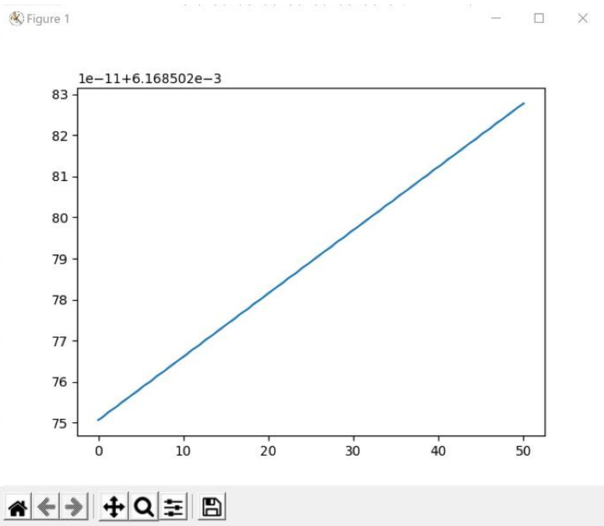


图 4 midpoint method $E - t$图

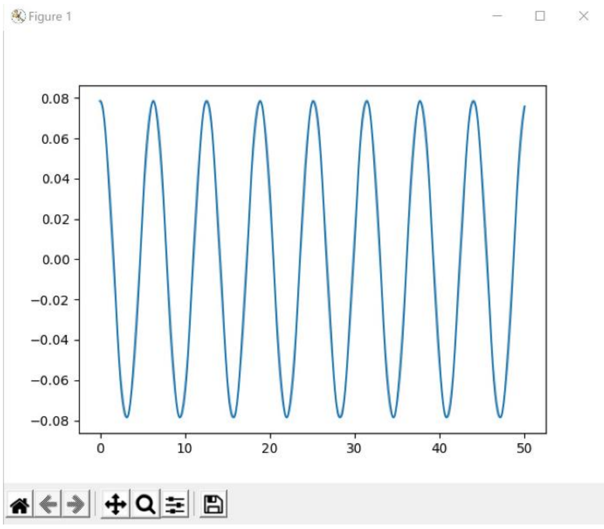


图 5 RK4 method $\theta - t$图

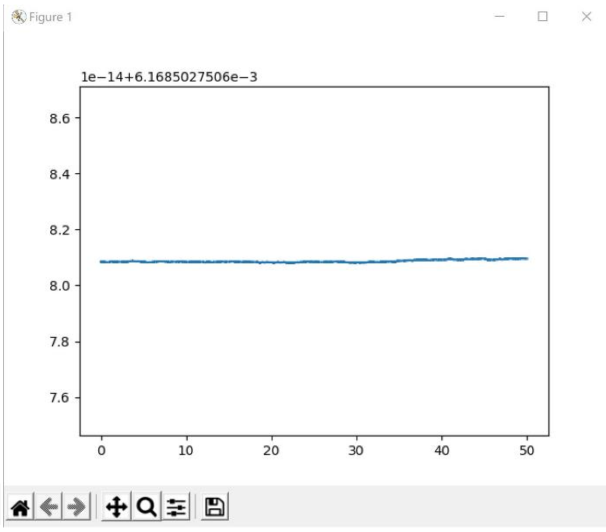


图 6 RK4 method $E - t$图

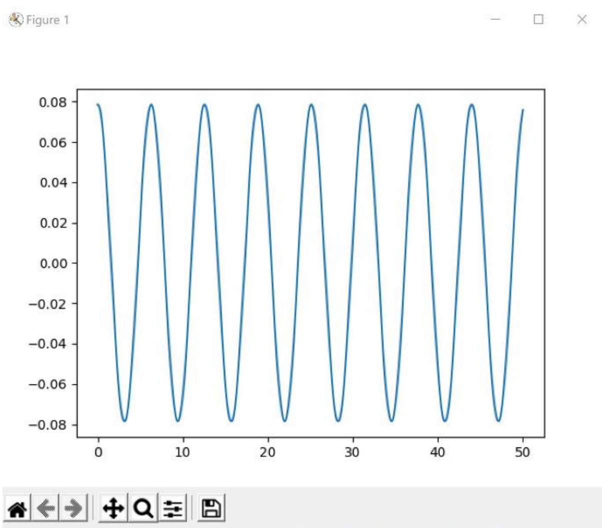


图 7 Euler-trapezoidal method $\theta - t$图

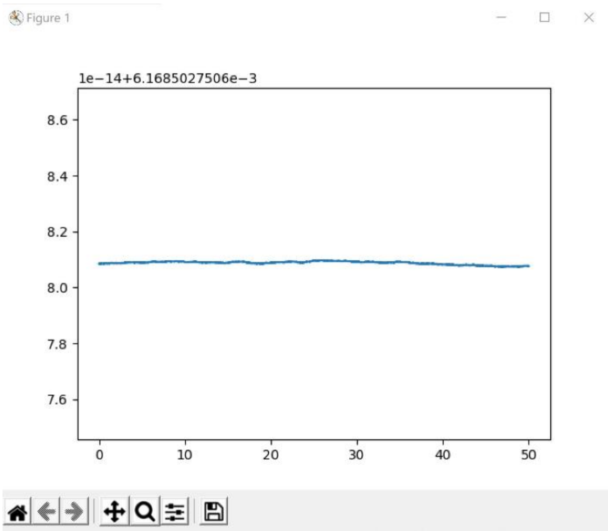


图 8 Euler-trapezoidal method $E - t$图

可以看到，$\theta - t$图总体看来都很相似，但$E - t$图差别很大。

Euler method 能量总体在增加，因为

$$y' = [y_1', y_2'] = \left[y_1 + h \cdot \frac{dy_1}{dt}, y_2 + h \cdot \frac{dy_2}{dt}\right] = [y_1 + hy_2, y_2 - hy_1]$$

$$E'^2 = y_1'^2 + y_2'^2 = (y_1 + hy_2)^2 + (y_2 - hy_1)^2 = y_1^2 + y_2^2 + h^2(y_1^2 + y_2^2) = E^2 + h^2E^2$$

因此总能量一直在以h^2增加。

midpoint method 能量也一直在增加，但数量级已经小了很多，在10^{-11}，原因在于：

$$y = [y_1 + hy_{temp2}, y_2 - hy_{temp1}]$$

$$E'^2 \approx y_1^2 + y_2^2 + h^2\left(y_{temp1}^2 + y_{temp2}^2\right)$$

总能量仍然在小幅增加。

RK4 method 和 Euler-trapezoidal method 能量总体很稳定，在10^{-14}数量级小幅波动。说明 RK4 method 和 Euler-trapezoidal method 是在整体上更优的解法，但也伴随着更大的计算量。

Input: $\theta + \ddot{\theta} = 0, E = \theta^2 + \dot{\theta}^2, \theta_0 = \frac{\pi}{40}, \dot{\theta}_0 = 0$

Output: $\theta(t), \dot{\theta}(t), E(t)$

Function $Euler(T, h)$

$N \leftarrow int(\frac{T}{h})$

$t_{points} \leftarrow [\,], y1_{points} \leftarrow [\,], y2_{points} \leftarrow [\,], energy \leftarrow [\,]$

$y \leftarrow [\frac{\pi}{40}, 0]$

For $i \leftarrow 1\ to\ N$ **Do**

  $t_{points}[i] \leftarrow h \cdot i$

  $y1_{points}[i] \leftarrow y[1]$

  $y2_{points}[i] \leftarrow y[2]$

  $energy[i] \leftarrow y[1]^2 + y[2]^2$

  $y \leftarrow [y[1] + h \cdot y[2],\ y[2] - h \cdot y[1]]$

Return $t_{points}, y1_{points}, y2_{points}, energy$

End Function

Function $midpoint(T, h)$

$N \leftarrow int(\frac{T}{h})$

$t_{points} \leftarrow [\,], y1_{points} \leftarrow [\,], y2_{points} \leftarrow [\,], energy \leftarrow [\,]$

$y \leftarrow [\frac{\pi}{40}, 0]$

For $i \leftarrow 1\ to\ N$ **Do**

  $t_{points}[i] \leftarrow h \cdot i$

  $y1_{points}[i] \leftarrow y[1]$

  $y2_{points}[i] \leftarrow y[2]$

$energy[i] \leftarrow y[1]^2 + y[2]^2$

$y_{temp} \leftarrow \left[y[1] + \frac{h}{2} \cdot y[2], y[2] - \frac{h}{2} \cdot y[1]\right]$

$y \leftarrow [y[1] + h \cdot y_{temp}[2],\ y[2] - h \cdot y_{temp}[1]]$

Return $t_{points}, y1_{points}, y2_{points}, energy$

End Function

其余函数$RK4(T, h)$、$Euler_trapezoidal(T, h)$的伪代码只需要按照上文所述，对相应部分做修改即可，在这里就省略了。

2. 用第 1 题中的方法解 Lorenz 方程。这道题相对简单，将第 1 题中的 Euler-trapezoidal method 复制粘贴过来，稍作修改。令$x = y_1, y = y_2, z = y_3$。

```
# Euler method:
y_derivative = [10 * (y[1] - y[0]), 28 * y[0] - y[1] - y[0] * y[2], y[0] * y[1] - (8/3) * y[2]]
y_temp = [y[0] + y_derivative[0]*h, y[1] + y_derivative[1]*h, y[2] + y_derivative[2]*h]

# Trapezoidal rule:
for j in range(iteration):
    y_derivative[0] = 10 * (y_temp[1] - y_temp[0])
    y_derivative[1] = 28 * y_temp[0] - y_temp[1] - y_temp[0] * y_temp[2]
    y_derivative[2] = y_temp[0] * y_temp[1] - (8/3) * y_temp[2]
    y_temp = [y[0] + y_derivative[0]*h, y[1] + y_derivative[1]*h, y[2] + y_derivative[2]*h]
y = y_temp
```

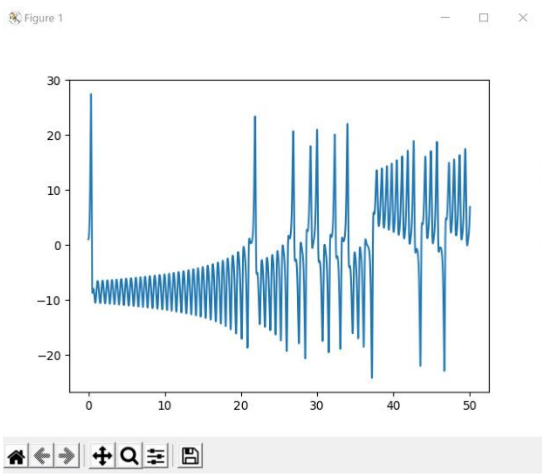
$$y_{derivative} = \frac{dy}{dt} = \left[\frac{dy_1}{dt}, \frac{dy_2}{dt}, \frac{dy_3}{dt}\right] = [10(y_2 - y_1), 28y_1 - y_2 - y_1y_3, y_1y_2 - \frac{8}{3}y_3]$$

$$y = [y_1 + h\frac{dy_1}{dt}, y_2 + h\frac{dy_2}{dt}, y_3 + h\frac{dy_3}{dt}]$$

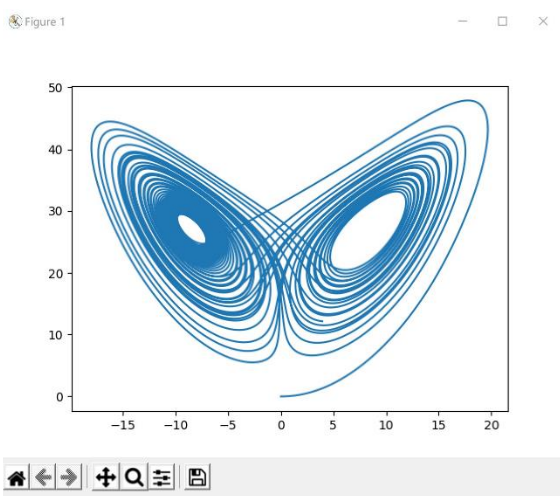
当然在 Trapezoidal rule 部分也做一样的修改，其他没有任何不同。

作图部分绘制的是$y - t$图和$z - x$图，得到的结果如图所示：

Unpredictable nature of the Motion:



Lop-sided butterfly-shaped plot:



Input: $\frac{dy_1}{dt} = 10(y_2 - y_1),\ \frac{dy_2}{dt} = 28y_1 - y_2 - y_1 y_3,\ \frac{dy_3}{dt} = y_1 y_2 - \frac{8}{3} y_3$

Output: $y_2(t),\ y_3(y_1)$

Function $Euler\_trapezoidal(T, h)$

$N \leftarrow int(\frac{T}{h})$

$t_{points} \leftarrow [\,], y1_{points} \leftarrow [\,], y2_{points} \leftarrow [\,], y3_{points} \leftarrow [\,]$

$y \leftarrow [0,1,0]$

$y' \leftarrow [0,1,0]$

For $i \leftarrow 1\ to\ N$ **Do**

  $t_{points}[i] \leftarrow h \cdot i$

  $y1_{points}[i] \leftarrow y[1]$

  $y2_{points}[i] \leftarrow y[2]$

  $y3_{points}[i] \leftarrow y[3]$

  /* Euler method */

  $y_{derivative} \leftarrow [10(y[2] - y[1]),\ 28y[1] - y[2] - y[1]y[3],\ y[1]y[2] - \frac{8}{3}y[3]]$

  $y' \leftarrow [y[1] + h \cdot y_{derivative}[1], y[2] + h \cdot y_{derivative}[2], y[3] + h \cdot y_{derivative}[3]]$

  /* Trapezoidal rule */

  For $j \leftarrow 1\ to\ 3$ **Do**

    $y_{derivative}[1] \leftarrow 10(y'[2] - y'[1])$

    $y_{derivative}[2] \leftarrow 28y'[1] - y'[2] - y'[1]y'[3]$

    $y_{derivative}[3] \leftarrow y'[1]y'[2] - \frac{8}{3}y'[3]$

    $y'[1] \leftarrow y[1] + h \cdot y_{derivative}[1]$

    $y'[2] \leftarrow y[2] + h \cdot y_{derivative}[2]$

    $y'[3] \leftarrow y[3] + h \cdot y_{derivative}[3]$

  $y \leftarrow y'$

Return $t_{points}, y1_{points}, y2_{points}, y3_{points}$

End Function

3. 使用 shooting method 解决一维 quantum harmonic oscillator 本征值问题。一维 quantum harmonic oscillator 的薛定谔方程为：

$$-\frac{\hbar}{2m}\frac{d^2}{dx^2}\psi(x)+\frac{m\omega^2}{2}x^2\psi(x)=E\psi(x)$$

将方程无量纲化，令$\frac{\hbar}{2m}=1,\frac{m\omega^2}{2}=1$，原方程化为：

$$(V(x)-E)\psi(x)=(x^2-E)\psi(x)=\frac{d^2}{dx^2}\psi(x)$$

方程的两个边界条件为：

$$\psi(x\to-\infty)=0,\psi(x\to\infty)=0$$

由于无法在真正的无穷远处设置边界条件，在计算时应选取合适的长度L使：

$$\psi(x=-L)\approx 0,\psi(x=L)\approx 0$$

且计算量不至于太大。

解决问题的基本思路：先将边值问题化为初值问题，我们已经有$\psi(x=-L)=0$，设置$\frac{d}{dx}\psi(x=-L)=1$（这个值的设置是任意的，因为ψ的相对幅值无关紧要，可被归一化）。再选择一个平凡的本征值E，使用 RK4 method 计算出$x=L$处的$\psi(E)$值。通过调整E值，使$x=L$处的$\psi(E)\to 0$。

定义$RK4(f,\psi_0,x,V,E)$函数，通过初值ψ_0来计算$x\in[-L,L]$范围内的ψ值。

```python
def derivative(y, V, E):
    # We set dψ/dx = ϕ, d^2ψ/dx^2 = dϕ/dx = (V-E)ψ = (V-x^2)ψ
    # y = [ψ, ϕ]
    result = np.asarray([y[1], (V - E) * y[0]])
    return result

def RK4(f, psi0, x, V, E):
    # Use RK4 method to calculate ψ in the region
    psi = np.array([psi0] * len(x))

    for i in range(len(x) - 1):
        h = x[i+1] - x[i]
        k1 = f(psi[i], V[i], E)
        k2 = f(psi[i] + (h*k1) / 2, V[i], E)
        k3 = f(psi[i] + (h*k2) / 2, V[i], E)
        k4 = f(psi[i] + (h*k3), V[i], E)
        psi[i+1] = psi[i] + h * (k1 + 2*k2 + 2*k3 + k4) / 6

    return psi
```

传入的参数f是一个函数，用以计算$f(\psi, V, E) = \frac{d}{dx}\psi$。$derivative(y, V, E)$就是用来计算导数的函数。

$$\frac{dy_1}{dx} = \frac{d}{dx}\psi = \phi = y_2, \qquad \frac{dy_2}{dx} = \frac{d}{dx}\phi = \frac{d^2}{dx^2}\psi = (V - E)\psi$$

传入的$y = [\psi, \phi]$，返回的$\frac{dy}{dx} = \left[\frac{d\psi}{dx}, \frac{d\phi}{dx}\right] = [\phi, (V - E)\psi]$。这样就完成了通过 RK4 method 计算$\psi$值的功能。

定义$count_zeros(func)$函数来计算波函数与 x 轴的交点数。当$func[i], func[i + 1]$异号时，我们认为波函数穿过了一次 x 轴，交点数+1。

```python
def count_zeros(func):
    # Count the number of zero-crossings of the wave function
    n = len(func)
    counter = 0
    for i in range(n - 1):
        if func[i] * func[i+1] < 0:
            counter += 1
    return counter
```

我们先通过波函数与 x 轴的交点来定出E的大致范围，因为能量E越高，波函数与 x 轴的交点即节点越多。所以当节点数大于我们需要的节点数时，能量偏高，我们需要降低E的值；当节点数小于我们需要的节点数时，能量偏低，我们需要升高E的值。

定义$shooting(E_{min}, E_{max}, zeros, \psi_0, x, V)$函数来实现 shooting method。$E_{min}, E_{max}$是我们定下的能量上下界，我们可以将范围定的宽一些（我定为$0 \le E \le 100$），以确保E落在范围内。将E的$tolerance$定为10^{-10}，然后通过 while 循环缩小E的范围直至其精确至$tolerance$。$minE, maxE$分别为E的下界和上界，在循环中令$E = (minE + maxE)/2$，当节点数大于我们需要的节点数时，令$maxE = E$，这样就降低了E的值；同理，当节点数小于我们需要的节点数时，令$minE = E$，以此升高E的值。当节点数与我们需要的节点数相同时，再看$\psi(x = L)$的值是否小于10^{-5}，并根据波函数的奇偶性来调整E的值。

最终当节点数$num = zeros, \psi(x = L) < 10^{-5}$时，我们返回$E$的值；或者当$maxE - minE = \delta E < 10^{-12}$时我们同样返回$E$的值。

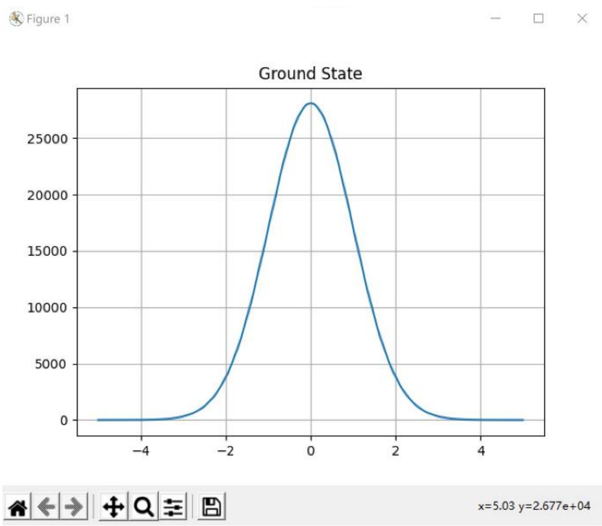
```python
def shooting(E_min, E_max, zeros, psi0, x, V):
    ''' We use Shooting method to calculate the eigenvalue E
    E_min - the lower bound of E
    E_max - the upper bound of E
    zeros - the number of zero-crossings of the wave function
    x - the array of x points
    V - the array of points of potential V(x)
    '''
    tolerance = 1e-10
    minE = E_min
    maxE = E_max

    while (maxE - minE) > tolerance:
        E = (minE + maxE) / 2.0
        psi = RK4(derivative, psi0, x, V, E)[: , 0]
        num = count_zeros(psi)

        if num > zeros:
            maxE = E
        elif num < zeros:
            minE = E
        elif abs(psi[-1]) > 1e-5:

            if (num%2 == 0):
                if psi[-1] > 0:
                    maxE = E
                else:
                    minE = E
            else:
                if psi[-1] > 0:
                    minE = E
                else:
                    maxE = E
        else:
            return E

    return E
```

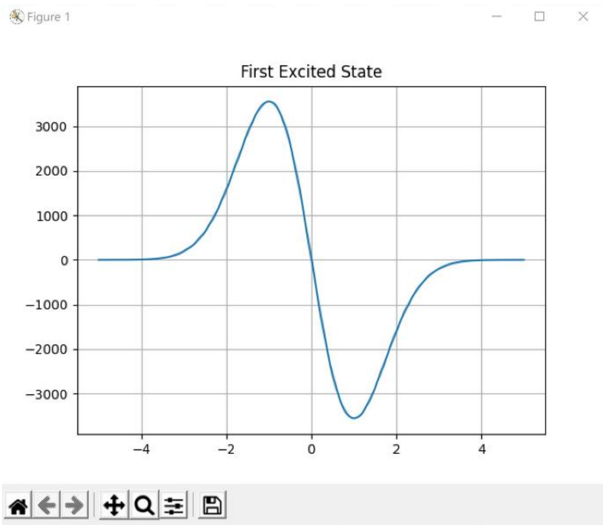
最后在$Harmonic_Oscillator()$和$main()$两个函数中完成参数的设置。

波函数的范围设置为$x \in [-5.0, 5.0]$，实际上 quantum harmonic oscillator 问题是有解析解的，$\psi_0(x) = e^{-\frac{x^2}{2}}, \psi_1(x) = \sqrt{2}x \cdot e^{-\frac{x^2}{2}}$，在$x = \pm 5.0$处的确有$\psi(x) \to 0$。x 点的间隔$\Delta x = 0.01$，包括首尾共 1001 个点。初值设置为$[\psi_0, \phi_0] = [0.0, 1.0]$。能量$E$的上下界设为$E_{min} = 0, E_{max} = 100$，由于我们有这个问题的解析解，$E_n = \left(n + \frac{1}{2}\right)\hbar\omega$，而我们在前文中已经将参数设置为$\frac{\hbar}{2m} = 1, \frac{m\omega^2}{2} = 1$，代入得$E_n = 2n + 1$，所以在$0 \le E \le 100$的范围内至少可以找到前数十个激发态的解。程序省去了归一化的步骤。

```python
def Harmonic_Oscillator(E_min, E_max, zeros):
    # Initial-value ψ0 = 0.0, and we set ϕ0 = 1.0
    psi_0 = np.asarray([0.0, 1.0])
    x = np.linspace(-5.0, 5.0, num=1001)
    V = x ** 2
    E = shooting(E_min, E_max, zeros, psi_0, x, V)
    psi = RK4(derivative, psi_0, x, V, E)[: , 0]

    return E, x, psi

def main():
    E1, x1_points, psi1_points = Harmonic_Oscillator(0, 100, 1)
    print('Found Ground State at E = %f' % E1)
    plt.plot(x1_points, psi1_points)
    plt.title('Ground State')
    plt.grid()
    plt.show()

    E2, x2_points, psi2_points = Harmonic_Oscillator(0, 100, 2)
    print('Found First Excited State at E = %f' % E2)
    plt.plot(x2_points, psi2_points)
    plt.title('First Excited State')
    plt.grid()
    plt.show()
```

程序运行的结果如图所示：





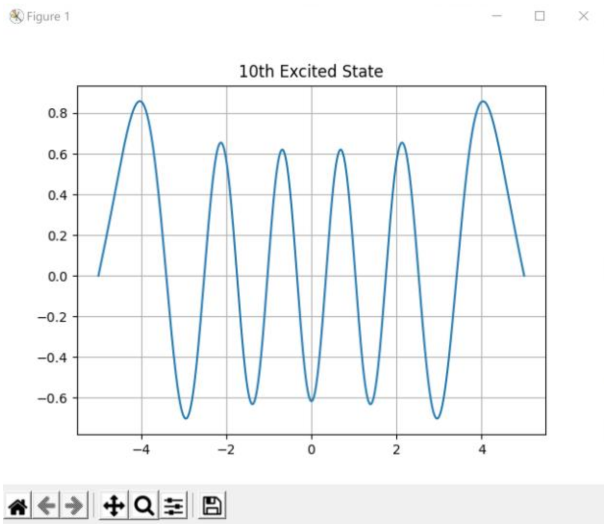
找到的本征值能量E：

```
PS C:\Users\11765\Desktop\学习\物理\计算物理\作业\homework7>
.python-2020.11.371526539\pythonFiles\lib\python\debugpy\laur
算物理\作业\homework7\7_3_17307110134.py'
Found Ground State at E = 1.000008
Found First Excited State at E = 3.000008
```

与解析解非常接近。

同样的方法也可以画出更高的激发态的波函数，以及算出更高激发态的能量，下图所示

是第十激发态的波函数：



Input: $-\frac{\hbar}{2m}\frac{d^2}{dx^2}\psi(x)+\frac{m\omega^2}{2}x^2\psi(x)=E\psi(x)$

Output: The wave function and eigenvalue E of the ground state and first excited state

Function $derivative(y,V,E)$ /* $y=[\psi,\phi]$*/

$result \leftarrow [y[2],(V-E)y[1]]$

Return $result$

End Function

Function $RK4(f,\psi_0,x,V,E)$ /* Use RK4 method to calculate ψ */

$\psi \leftarrow [[\psi_0],[\psi_0],\ldots,[\psi_0]]$

For $i \leftarrow 1\ to\ len(x)$ **Do**

$h \leftarrow x[i+1]-x[i]$

$k1 \leftarrow f(\psi[i],\ V[i],\ E)$

$k2 \leftarrow f(\psi[i]+\frac{h\cdot k1}{2},\ V[i],\ E)$

$k3 \leftarrow f(\psi[i]+\frac{h\cdot k2}{2},\ V[i],\ E)$

$k4 \leftarrow f(\psi[i]+h\cdot k3,\ V[i],\ E)$

$\psi[i+1]=\psi[i]+h\cdot\frac{k1+2k2+2k3+k4}{6}$

Return ψ

End Function

Function $count\_zeros(func)$

/* Count the number of zero-crossings of the wave function */

$n \leftarrow len(func)$

$counter \leftarrow 0$

For $i \leftarrow 1 \; to \; n-1$ **Do**

  If $func[i] \times func[i+1] < 0$ **Then**

    $counter \leftarrow counter + 1$

  End If

Return $counter$

End Function

Function $shooting(E_{min}, E_{max}, zeros, \psi_0, x, V)$

/*$E_{min} -$ the lower bound of E*/

/*$E_{max} -$ the upper bound of E*/

/* $zeros -$ the number of zero-crossings of the wave function */

/* $x -$ the array of x points */

/* $V -$ the array of points of potential $V(x)$*/

$tolerance \leftarrow 10^{-10}$

$minE \leftarrow E_{min}$

$maxE \leftarrow E_{max}$

While $maxE - minE > tolerance$ **Do**

  $E \leftarrow \frac{minE+maxE}{2}$

  $\psi \leftarrow RK4(derivative, \psi_0, x, V, E)$

  $num \leftarrow count\_zeros(\psi)$

  If $num > 0$ **Then**

    $maxE \leftarrow E$

  Elif $num < 0$ **Then**

    $minE \leftarrow E$

End If

End While

Return E

End Function