1. 使用 relaxation method 解泊松方程：

$$\nabla^2\varphi(x,y) = -\frac{\rho(x,y)}{\varepsilon_0}$$

定义$Gauss_Seidel(L_x, L_y, h, err, iter)$函数来实现 Gauss-Seidel method。

```python
def Gauss_Seidel(L_x, L_y, h, err, iter):
# x belongs to [0, L_x]
# y belongs to [0, L_y]
# h = Δx = Δy
# err - maximum error
# iter - maximum iteration times
    n_x = int((L_x / h) + 1)
    n_y = int((L_y / h) + 1)
    x = np.linspace(0, L_x, n_x)
    y = np.linspace(0, L_y, n_y)
    X, Y = np.meshgrid(x, y)
```

$x \in [0, L_x], y \in [0, L_y]$，$h$是$x$方向和$y$方向上的步长，$err$是所能允许的最大误差，$iter$是最大迭代次数。$n_x = \frac{L_x}{h} + 1, n_y = \frac{L_y}{h} + 1$分别是$x$方向和$y$方向上的格点数，并生成$X, Y$两个矩阵以便后续画三维图时使用。

```python
    f = np.zeros([n_y, n_x]) # Source term f(x, y) = 0
#   f = -np.ones([n_y, n_x])
    u = np.ones([n_y, n_x])  # Initial guess
    u[0,:] = 0.0             # Boundary conditions
    u[-1,:] = 0.0
    u[:,0] = 0.0
    u[:,-1] = 1.0
#   u[:,-1] = 0.0
    error = 1.0              # Set an arbitrary initial error
    iteration = 0            # Iteration times
```

$$\nabla^2\varphi(x,y) = -\frac{\rho(x,y)}{\varepsilon_0} = f(x,y)$$

$f(x,y)$ 就是方程的$source\ term$，$u(x,y)$就是我们要求的$\varphi(x,y)$。易知f, u都是$y \times x$的矩阵（y轴对应矩阵的行数，x轴对应矩阵的列数，所以矩阵是$y \times x$而不是$x \times y$）。

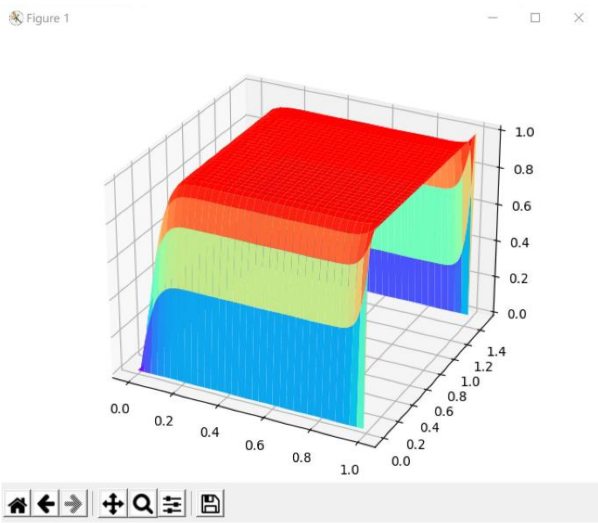
初始化f, u矩阵，对于(a)题f值全部为 0，对于(b)题f值全部为-1；对于两个小题，u的初值均全部设为 1。然后对(a) (b)题设置不同的边界条件，两题不同之处在于:(a)题$u[x,-1] = \varphi(x, L_y) = 1.0$，(b)题$u[x,-1] = \varphi(x, L_y) = 0.0$。并将迭代次数$iteration$设为 0。

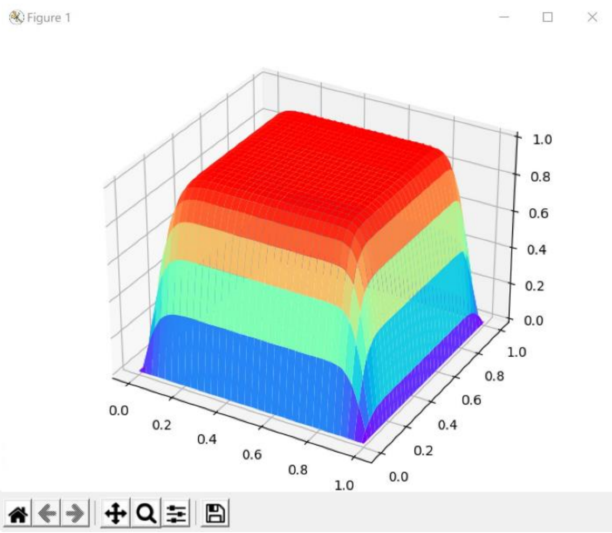
```python
    while error > err and iteration < iter:
        u_temp = u.copy()

        for i in range(1, n_y - 1):
            for j in range(1, n_x - 1):
                u[i, j] =  (u[i+1, j] + u[i-1, j] + u[i, j+1] + u[i, j-1] - f[i, j]*h*h) / 4

        error = (abs(u - u_temp)).max()
        iteration += 1
```

迭代继续的条件为误差$error$仍大于设定的最大误差err，或者迭代次数$iteration$仍小于设定的最大迭代次数$iter$。边界条件$u[0,y]=u[-1,y]=u[x,0]=0.0, u[x,-1]=1.0$在迭代过程中不能改变，所以改变的就是$u[x,y](x\in[1,n_y-1), y\in[1,n_x-1))$。每次迭代中$u[i,j]=\frac{u[i+1,j]+u[i-1,j]+u[i,j+1]+u[i,j-1]-h^2f[i,j]}{4}$，并重新计算误差$error$和迭代次数$iteration$。

```python
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_surface(X, Y, u, cmap='rainbow')
    plt.show()

Gauss_Seidel(1.0, 1.5, 0.01, 0.01, 1000)
# Gauss_Seidel(1.0, 1.0, 0.01, 0.01, 1000)
```

最后将得到的$u=\varphi(x,y)$绘制成三维图，得到的结果如图所示：





左图对应(a)题，右图对应(b)题。

Input: $\nabla^2\varphi(x,y) = -\frac{\rho(x,y)}{\varepsilon_0}$, boundary conditions

Output: $\varphi(x,y)$

Function $Gauss\_Seidel(L_x, L_y, h, err, iter)$

$n_x \leftarrow int\left(\frac{L_x}{h}\right) + 1,\ n_y \leftarrow int\left(\frac{L_y}{h}\right) + 1$

$$f \leftarrow \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}, u \leftarrow \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ 0 & 1 & \dots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix}$$

$error \leftarrow 1.0, iteration \leftarrow 0$

While $error > err$ **And** $iteration < iter$ **Do**

    $u_{temp} \leftarrow u$

    For $i \leftarrow 1\ to\ n_y - 2$ **Do**

        For $j \leftarrow 1\ to\ n_x - 2$ **Do**

            $u[i,j] \leftarrow \frac{u[i+1,j]+u[i-1,j]+u[i,j+1]+u[i,j-1]-h^2 f[i,j]}{4}$

    $error \leftarrow max\,\{|u - u_{temp}|\}$

    $iteration \leftarrow iteration + 1$

Return u

End Function

2. 用 Crank–Nicolson method 求解含时薛定谔方程。由于是方势阱，可以设$V(-L \le x \le L) = 0$，薛定谔方程化为：

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} = i\hbar\,\frac{\partial\psi}{\partial t}$$

令$\Delta x = a, \Delta t = h$，上式可化为：

$$\psi(x,t+h) - h\frac{i\hbar}{4ma^2}[\psi(x+a,t+h) - 2\psi(x,t+h) + \psi(x-a,t+h)]$$
$$= \psi(x,t) + h\frac{i\hbar}{4ma^2}[\psi(x+a,t) - 2\psi(x,t) + \psi(x-a,t)]$$

将方程写为矩阵形式：

$$\boldsymbol{A}\psi(t+h) = \boldsymbol{B}\psi(t)$$

其中：

$$\psi(t) = (\psi(a,t), \psi(2a,t), \psi(3a,t)\ \ldots)^T$$

$\boldsymbol{A}, \boldsymbol{B}$均为三对角矩阵：

$$\boldsymbol{A} = \begin{pmatrix} a_1 & a_2 & 0 & 0 & \ldots \\ a_2 & a_1 & a_2 & 0 & \ldots \\ 0 & a_2 & a_1 & a_2 & \ldots \\ 0 & 0 & a_2 & a_1 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \qquad \boldsymbol{B} = \begin{pmatrix} b_1 & b_2 & 0 & 0 & \ldots \\ b_2 & b_1 & b_2 & 0 & \ldots \\ 0 & b_2 & b_1 & b_2 & \ldots \\ 0 & 0 & b_2 & b_1 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

其中：

$$a_1 = 1 + h\frac{i\hbar}{2ma^2}, a_2 = -h\frac{i\hbar}{4ma^2}, b_1 = 1 - h\frac{i\hbar}{2ma^2}, b_2 = h\frac{i\hbar}{4ma^2}$$

进行无量纲化处理，令$\alpha = \frac{\Delta t}{\Delta x^2} = \frac{h}{a^2}, \frac{\hbar}{4m} = 1$：

$$a_1 = 1 + 2i\alpha, a_2 = -i\alpha, b_1 = 1 - 2i\alpha, b_2 = i\alpha$$

$$\psi(t+h) = \boldsymbol{A}^{-1}\boldsymbol{B}\psi(t)$$

定义$Crank_Nicolson(alpha, psi)$函数来实现 Crank-Nicolson method，通过α和ψ_n来求出ψ_{n+1}。

$n = len(psi)$就是我们要设置的$\boldsymbol{A}, \boldsymbol{B}$两个矩阵的维数。通过 python 与稀疏矩阵有关的库 scipy.sparse，我们可以在对角线上设置元素，以构造$\boldsymbol{A}, \boldsymbol{B}$两个三对角矩阵。

$$vec1 = [i\alpha, i\alpha, \ldots], \qquad vec2 = [1-2i\alpha, 1-2i\alpha, \ldots],$$

$$vec3 = [-i\alpha, -i\alpha, \ldots], \qquad vec4 = [1+2i\alpha, 1+2i\alpha, \ldots]$$

```python
def Crank_Nicolson(alpha, psi):
    # psi - The wave function ψ_n
    # alpha - Δt/Δx^2
    # Return the wave function ψ_n+1
    n = len(psi)
    diags = np.array([0, -1, 1])
    vec1 = np.array([alpha*1j for i in range(n)])
    vec2 = np.array([(1 - 2*alpha*1j) for i in range(n)])
    vec3 = np.array([-alpha*1j for i in range(n)])
    vec4 = np.array([(1 + 2*alpha*1j) for i in range(n)])

    data1 = np.array([vec2, vec1, vec1])
    U1 = spdiags(data1, diags, n, n).toarray()

    data2 = np.array([vec4, vec3, vec3])
    U2 = spdiags(data2, diags, n, n).toarray()

    inv_U2 = np.linalg.inv(U2)
    U = np.dot(inv_U2, U1)
    next_psi = np.dot(U, psi)
    next_psi[0] = next_psi[-1] = 0.0+0.0j

    return next_psi
```

设置完后，我们有：

$$\boldsymbol{U_2} = \boldsymbol{A} = \begin{pmatrix} 1+2i\alpha & -i\alpha & 0 & 0 & \dots \\ -i\alpha & 1+2i\alpha & -i\alpha & 0 & \dots \\ 0 & -i\alpha & 1+2i\alpha & -i\alpha & \dots \\ 0 & 0 & -i\alpha & 1+2i\alpha & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

$$\boldsymbol{U_1} = \boldsymbol{B} = \begin{pmatrix} 1-2i\alpha & i\alpha & 0 & 0 & \dots \\ i\alpha & 1-2i\alpha & i\alpha & 0 & \dots \\ 0 & i\alpha & 1-2i\alpha & i\alpha & \dots \\ 0 & 0 & i\alpha & 1-2i\alpha & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\psi_{n+1} = \boldsymbol{U_2}^{-1}\boldsymbol{U_1}\psi_n$$

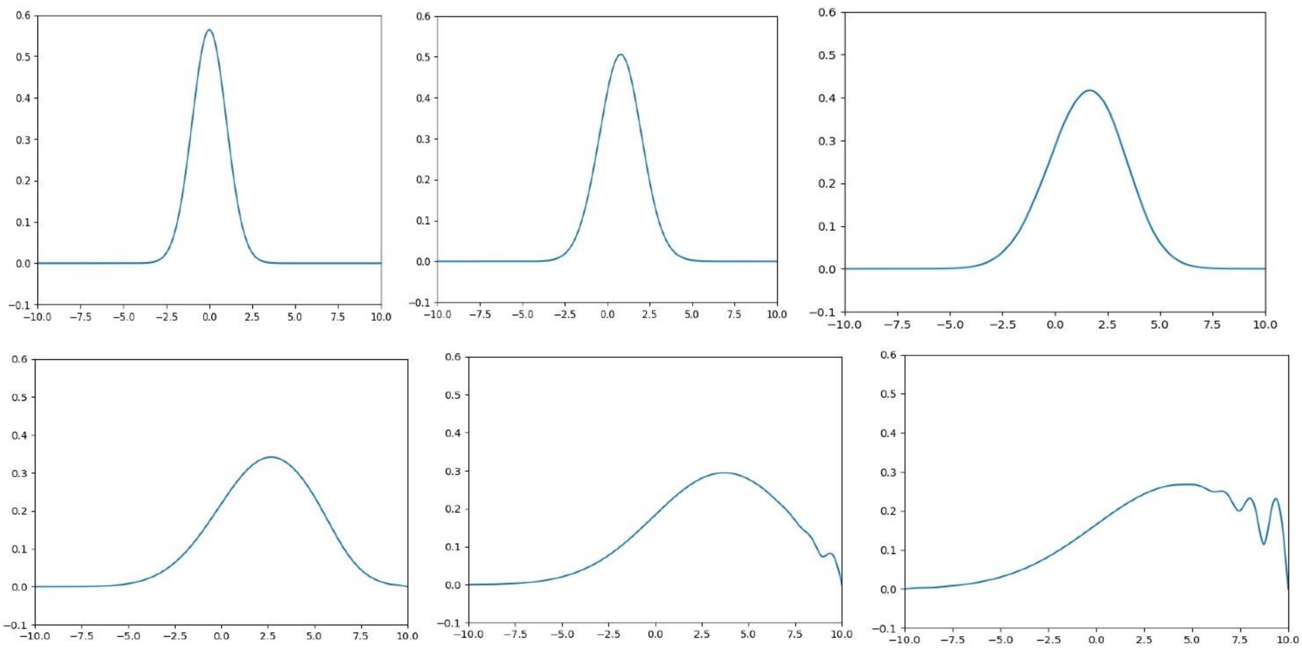
由于我们求解的是方势阱中的波函数，为简化问题，将方势阱当作无限深方势阱，波函数在势阱的两个端点处应该为 0，所以设置$\psi[0] = \psi[-1] = 0$。

$Initial_State(x)$函数定义了波函数的初始形式，即：

$$\psi(x,0) = \sqrt{\frac{1}{\pi}}\exp\left(ix - \frac{x^2}{2}\right)$$

```python
def Initial_State(x):
    # Calculate the Gaussian initial state ψ(x,0)
    psi = np.sqrt(1 / np.pi) * np.exp(1j * x - x**2 / 2)
    return psi

def wave_function():
    plt.ion()
    x = np.linspace(-10, 10, 2001)
    y = Initial_State(x)
    y[0] = y[-1] = 0.0+0.0j

    for i in range(50):
        plt.clf()
        plt.axis([-10, 10, -0.1, 0.6])
        plt.plot(x,abs(y))
        y = Crank_Nicolson(500, y)
        plt.pause(0.05)

    return x, y
```

初始波函数处在势阱的中心，势阱范围$x \in [-10,10]$，我们取间隔$\Delta x = 0.01$，共 2001 个点。调用$Crank_Nicolson(alpha, psi)$函数时取$alpha = \alpha = \frac{\Delta t}{\Delta x^2} = 500$，故$\Delta t = 0.05s$。我们把一系列波函数$\psi_0, \psi_1, \psi_2$ ...绘制出来，就形成了波函数随时间的演化图。结果如图所示（由于要求解2001×2001的矩阵，因此需要一点计算时间）：



可以看出，波函数向右传播，峰逐渐变矮变宽。由于我只是简单地把边界条件设为 0，当波函数的峰逐渐接近边界时，波函数出现了剧烈震荡。

Input: $\psi(x,0) = \sqrt{\frac{1}{\pi}}\exp\left(ix - \frac{x^2}{2}\right)$, boundary conditions

Output: $\psi(x,t)$

Function $Crank\_Nicolson(alpha, psi)$

$n \leftarrow len(psi)$

$$\boldsymbol{U_1} \leftarrow \begin{pmatrix} 1-2i\alpha & i\alpha & 0 & 0 & \dots \\ i\alpha & 1-2i\alpha & i\alpha & 0 & \dots \\ 0 & i\alpha & 1-2i\alpha & i\alpha & \dots \\ 0 & 0 & i\alpha & 1-2i\alpha & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$$\boldsymbol{U_2} \leftarrow \begin{pmatrix} 1+2i\alpha & -i\alpha & 0 & 0 & \dots \\ -i\alpha & 1+2i\alpha & -i\alpha & 0 & \dots \\ 0 & -i\alpha & 1+2i\alpha & -i\alpha & \dots \\ 0 & 0 & -i\alpha & 1+2i\alpha & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$\boldsymbol{U} \leftarrow \boldsymbol{U_2}^{-1}\boldsymbol{U_1}$

$next\_psi \leftarrow \boldsymbol{U} \cdot psi$

Return $next\_psi$

End Function

Function $Initial\_State(x)$

$psi \leftarrow \sqrt{\frac{1}{\pi}}exp\left(ix - \frac{x^2}{2}\right)$

Return psi

End Function

Function $wave\_function()$

$x \leftarrow [-10, -9.99, -9.98, \dots, 9.98, 9.99, 10]$

$y \leftarrow Initial\_State(x)$

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

For $i \leftarrow 1\ to\ 30$ **Do**

    $plot(x, |y|)$

    $y \leftarrow Crank\_Nicolson(500, y)$

Return $\boldsymbol{x}, \boldsymbol{y}$

End Function