

# 计算物理作业 8

杨远青 22300190015  CompPhys 24

2024 年 11 月 26 日

正面迎击 *ddl* 军团！

## 1 题目 1：松弛法求解泊松方程

### 1.1 题目描述

Consider the Poisson equation:

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

from electrostatics on a rectangular geometry with  $x \in [0, L_x]$  and  $y \in [0, L_y]$ . Write a program that solves this equation using the relaxation method and test your program with the following cases:

- (a)  $\rho(x, y) = 0$ ,  $\varphi(0, y) = \varphi(L_x, y) = \varphi(x, 0) = 0$ ,  $\varphi(x, L_y) = 1 \text{ V}$ ,  $L_x = 1 \text{ m}$ , and  $L_y = 1.5 \text{ m}$ ;  
(b)  $\frac{\rho(x, y)}{\varepsilon_0} = 1 \text{ V/m}^2$ ,  $\varphi(0, y) = \varphi(L_x, y) = \varphi(x, 0) = \varphi(x, L_y) = 0$ , and  $L_x = L_y = 1 \text{ m}$ .

### 1.2 程序描述

本程序支持在题示两种案例与自定义案例（指定矩形区域大小、求解参数、均匀源项与第一类边界条件，即四周电势）下的求解，并内置了解析解的计算，以便于对比。其中对于案例 (a)，即无源电荷，三边接地，齐次解为：

$$\phi_h(x, y) = \frac{4V_0}{\pi} \sum_{n=1,3,5,\dots}^N \frac{1}{n} \sin\left(\frac{n\pi x}{L_x}\right) \frac{\sinh\left(\frac{n\pi y}{L_x}\right)}{\sinh\left(\frac{n\pi L_y}{L_x}\right)}$$

考虑到  $\sinh$  函数在  $\approx \exp(700)$  会数值溢出，在  $N$  达到  $N_{approx}$  时进行指数近似

$$\frac{\sinh\left(\frac{n\pi y}{L_x}\right)}{\sinh\left(\frac{n\pi L_y}{L_x}\right)} \approx \exp\left(\frac{n\pi(y - L_y)}{L_x}\right)$$

对于案例 (b)，即均匀源电荷，四边接地，满足接地条件与均匀源项的特解为：

$$\phi_p(x, y) = \sum_{n=1,3,5,\dots}^N \sum_{m=1,3,5,\dots}^M \frac{16\rho}{\varepsilon_0 \pi^2 nm \left( \left(\frac{n\pi}{L_x}\right)^2 + \left(\frac{m\pi}{L_y}\right)^2 \right)} \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right)$$

理论上这个级数不是绝对收敛的，且只能计算到一定的阶数，应当采用黎曼求和的对角线求和次序，但在最初的代码中偷懒直接使用了内外 for 循环，结果是解析解与误差图中 x,y 方向不对称。但在测试中发现，这种不对称的周期正好与截断项数  $N, M$  可以对应，因此保留了这个小 bug，恰可以增加一重误差图的物理含义检验。对于自定义案例，

根据用户指定的边界条件，相应的解析解只需计算四个边界条件（需要改变自变量进行旋转）对应的齐次解  $\phi_h^i(x, y)$  进行叠加，再加上均匀源项的接地特解  $\phi_p(x, y)$  即可

$$\phi(x, y) = \sum_{i=1}^4 \phi_h^i(x, y) + \phi_p(x, y)$$

在求解器 `PoissonSolver` 类中，内置了 Jacobi, Gauss-Seidel, SOR 三种迭代方法，用户可以通过 `method` 参数选择。它们的算法细节将在伪代码中详细介绍。其中 SOR 的松弛因子  $\omega$  根据输入参数由 `get_optimal_omega` 方法自动计算，采用的公式来自 *Numeric Recipes* (2nd ed.) 的第 19.5 节，即

$$\omega = \frac{2}{1 + \sqrt{1 - \rho_{\text{Jacobi}}^2}}$$

其中  $\rho_{\text{Jacobi}}$  为 Jacobi 方法的谱半径

$$\rho_{\text{Jacobi}} = \frac{\cos \frac{\pi}{N_x} + \left(\frac{\Delta x}{\Delta y}\right)^2 \cos \frac{\pi}{N_y}}{1 + \left(\frac{\Delta x}{\Delta y}\right)^2}$$

在等间距正方形大网格中化为课件中的近似表达式  $\omega \simeq \frac{2}{1+\pi/L}$  实际测试表明该因子选取大幅加快收敛速度，赞！

请在 `Problem_1/src` 目录下运行 `python -u poisson.py`, 需安装辅助计算的 `numpy` 库与绘图用的 `matplotlib` 库。程序从用户输入获取求解案例及参数之后，将首先输出三种方法的迭代动画（步数过多时抽取最多 100 帧进行展示），随后绘制三者的收敛曲线，即每步最大变化量随步数的变化。最后输出计算结果及与解析解的对比图，以及计算误差图，详见下文的结果示例。

### 1.3 伪代码

Powered by `LATeX pseudocode generator`

---

**Algorithm 1:** Jacobi Method for Solving Poisson's Equation

---

```

Input:  $\phi$  (potential matrix), tolerance, max_iter
Output:  $\phi$  (updated potential matrix)

1  $dx^2 \leftarrow dx^2$ ,  $dy^2 \leftarrow dy^2$ ,  $\text{denom} \leftarrow 2 \times (dx^2 + dy^2)$ ; // Precompute constants
2 for  $it \leftarrow 1$  to max_iter do
3    $\phi_{\text{new}} \leftarrow \phi$ ; // Create a copy for new updates
4   for  $i \leftarrow 1$  to  $N_x - 2$  do
5     for  $j \leftarrow 1$  to  $N_y - 2$  do
6        $\phi_{\text{new}}[i, j] \leftarrow \frac{(\phi[i+1, j] + \phi[i-1, j]) \times dy^2 + (\phi[i, j+1] + \phi[i, j-1]) \times dx^2 + dx^2 \times dy^2 \times \rho_{\epsilon_0}[i, j]}{\text{denom}}$ ; // Jacobi update
7     end
8   end
9    $\text{max\_change} \leftarrow \max(|\phi_{\text{new}} - \phi|)$ ;
10  if  $\text{max\_change} < \text{tolerance}$  then
11    return  $\phi_{\text{new}}$ ; // Convergence reached
12  end
13   $\phi \leftarrow \phi_{\text{new}}$ ; // Update for next iteration
14 end
15 return  $\phi$ ; // Return after max iterations

```

---

---

**Algorithm 2:** Gauss-Seidel Method for Solving Poisson's Equation

---

**Input:**  $\phi$  (potential matrix), **tolerance**, **max\_iter**

**Output:**  $\phi$  (updated potential matrix)

```
1  $dx^2 \leftarrow dx^2, dy^2 \leftarrow dy^2, \text{denom} \leftarrow 2 \times (dx^2 + dy^2) ;$  // Precompute constants
2 for  $it \leftarrow 1$  to  $\text{max\_iter}$  do
3    $\text{max\_change} \leftarrow 0 ;$  // Reset max change for this iteration
4   for  $i \leftarrow 1$  to  $N_x - 2$  do
5     for  $j \leftarrow 1$  to  $N_y - 2$  do
6        $\phi_{\text{old}} \leftarrow \phi[i, j] ;$ 
7        $\phi[i, j] \leftarrow \frac{(\phi[i+1, j] + \phi[i-1, j]) \times dy^2 + (\phi[i, j+1] + \phi[i, j-1]) \times dx^2 + dx^2 \times dy^2 \times \rho_{\epsilon_0}[i, j]}{\text{denom}} ;$  // Gauss-Seidel update
8        $\text{max\_change} \leftarrow \max(\text{max\_change}, |\phi[i, j] - \phi_{\text{old}}|) ;$  // Track max change
9     end
10   end
11   if  $\text{max\_change} < \text{tolerance}$  then
12     return  $\phi ;$  // Convergence reached
13   end
14 end
15 return  $\phi ;$  // Return after max iterations
```

---

---

**Algorithm 3:** SOR Method for Solving Poisson's Equation

---

**Input:**  $\phi$  (potential matrix),  $\omega$  (relaxation factor), **tolerance**, **max\_iter**

**Output:**  $\phi$  (updated potential matrix)

```
1  $dx^2 \leftarrow dx^2, dy^2 \leftarrow dy^2, \text{denom} \leftarrow 2 \times (dx^2 + dy^2) ;$  // Precompute constants
2 for  $it \leftarrow 1$  to  $\text{max\_iter}$  do
3    $\text{max\_change} \leftarrow 0 ;$  // Reset max change for this iteration
4   for  $i \leftarrow 1$  to  $N_x - 2$  do
5     for  $j \leftarrow 1$  to  $N_y - 2$  do
6        $\phi_{\text{old}} \leftarrow \phi[i, j] ;$ 
7        $\phi_{\text{new}} \leftarrow \frac{(\phi[i+1, j] + \phi[i-1, j]) \times dy^2 + (\phi[i, j+1] + \phi[i, j-1]) \times dx^2 + dx^2 \times dy^2 \times \rho_{\epsilon_0}[i, j]}{\text{denom}} ;$  // Standard update
8        $\phi[i, j] \leftarrow (1 - \omega) \cdot \phi_{\text{old}} + \omega \cdot \phi_{\text{new}} ;$  // SOR update
9        $\text{max\_change} \leftarrow \max(\text{max\_change}, |\phi[i, j] - \phi_{\text{old}}|) ;$  // Track max change
10    end
11   end
12   if  $\text{max\_change} < \text{tolerance}$  then
13     return  $\phi ;$  // Convergence reached
14   end
15 end
16 return  $\phi ;$  // Return after max iterations
```

---

## 1.4 结果示例

### 1.4.1 Case (a): 无源电荷，三边接地

```
● (base) gilbert@Gilbert-YoungMacBook src % python -u poisson.py

==== 泊松方程求解器 ====
请选择要求解的案例：
a - 无源项，顶部电势为1V其余为0V
b - 均匀源项( $\rho/\epsilon_0 = 1 \text{ V/m}^2$ )，边界全为0V
c - 自定义均匀源项和边界条件

请输入选项 (a/b/c): a

请输入网格点数 (Nx, Ny) 和最大迭代次数 (max_iter)，按回车使用默认值：
请输入 x 方向的网格点数 Nx (默认 50):
请输入 y 方向的网格点数 Ny (默认 75):
请输入最大迭代次数 max_iter (默认 10000): 1000
在 n = 149 时开始使用指数近似
解析解未在最大 n 值 1000 内收敛，最后一项贡献为 1.27e-03
解析解使用的傅里叶级数中最大 n: 999
从 n = 149 开始使用指数近似

使用 jacobi方法求解...
Jacobi方法达到最大迭代次数 1000 仍未收敛
最大偏差: 6.29e-02V
最大偏差位置: (x=0.490m, y=0.912m)
迭代次数: 1000
求解时间: 0.0200秒
2024-11-25 18:05:20.419 python[15777:372426] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-25 18:05:20.419 python[15777:372426] +[IMKInputSession subclass]: chose IMKInputSession_Modern

使用 gauss_seidel方法求解...
Gauss-Seidel方法达到最大迭代次数 1000 仍未收敛
最大偏差: 1.38e-02V
最大偏差位置: (x=0.490m, y=0.770m)
迭代次数: 1000
求解时间: 2.7074秒

使用 sor方法求解...
使用最优松弛因子:  $\omega = 1.899$ 
SOR方法在第 202 次迭代收敛
最大偏差: 7.25e-03V
最大偏差位置: (x=0.959m, y=1.439m)
迭代次数: 202
求解时间: 0.6313秒
```

图 1: (a): 终端输出

本次测试特地将最大迭代次数截断在 1000，以测试三种方式的精度与收敛速度。可以看到最终与解析解的最大误差比较中，Jacobi 方法的误差最大，而 Gauss-Seidel 方法的误差较小，SOR 方法的误差最小，且在第 202 步便已收敛至指定精度。在耗时方面，Jacobi 因为不需要使用实时更新的  $\phi$ ，可以进行多线程并行，所以耗时反而最短，而 Gauss-Seidel 方法（隐式）因为需要使用新的  $\phi$ ，耗时最长，SOR 方法介于两者之间，虽不能并行，但收敛速度最快。

在输出的误差比较图中，Jacobi 与 Gauss-Seidel 方法的误差图形状相似，最大偏差点均在中心处，但误差更小，而 SOR 方法几乎没有偏差，最大偏差点在硬性边界条件的不连续处，这非程序本身的问题。在运行动画时将能更直观展现迭代过程，SOR 的更新更为迅速且方向准确。

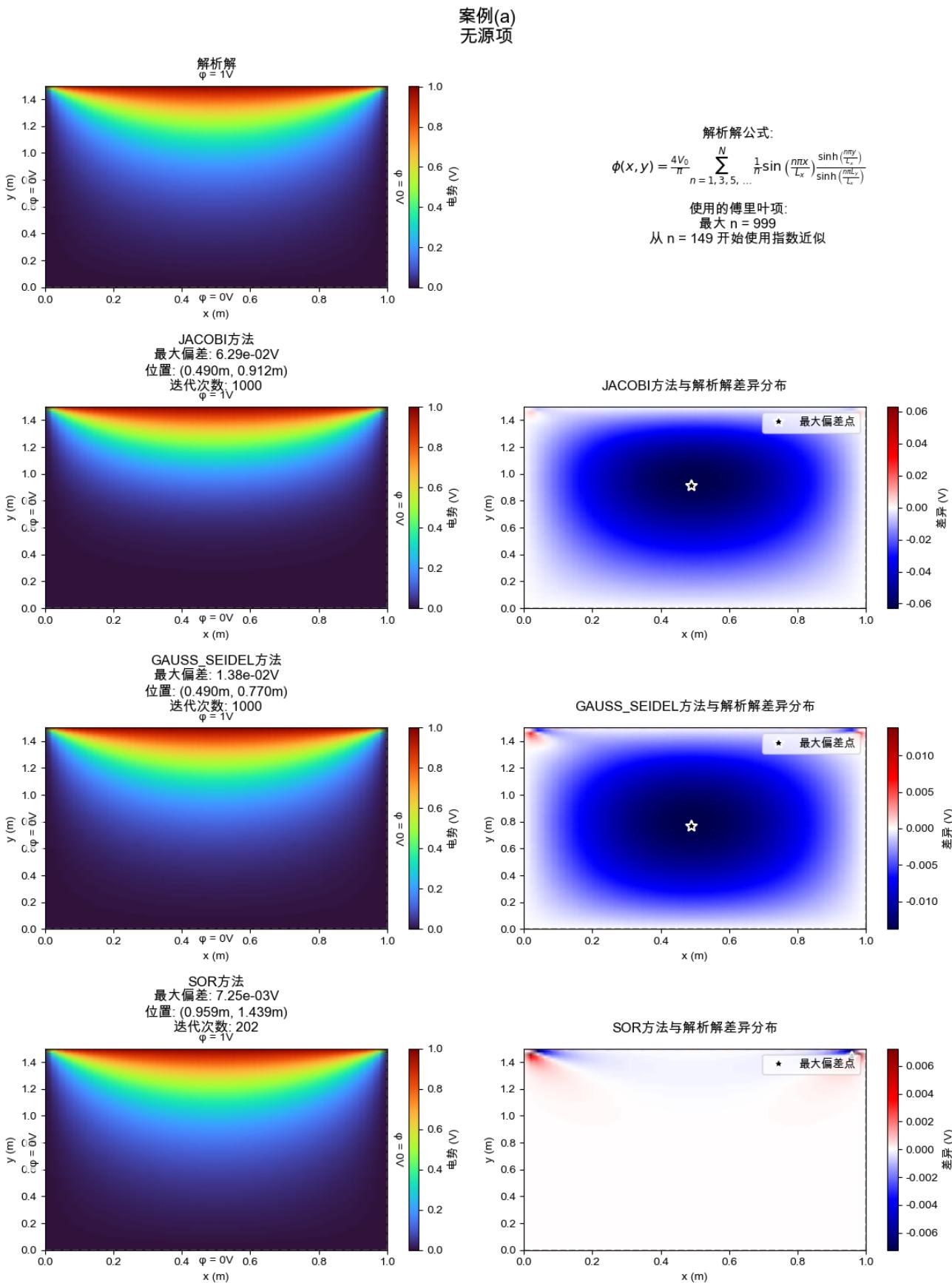


图 2: (a): 计算结果及对比

### 各方法的收敛曲线

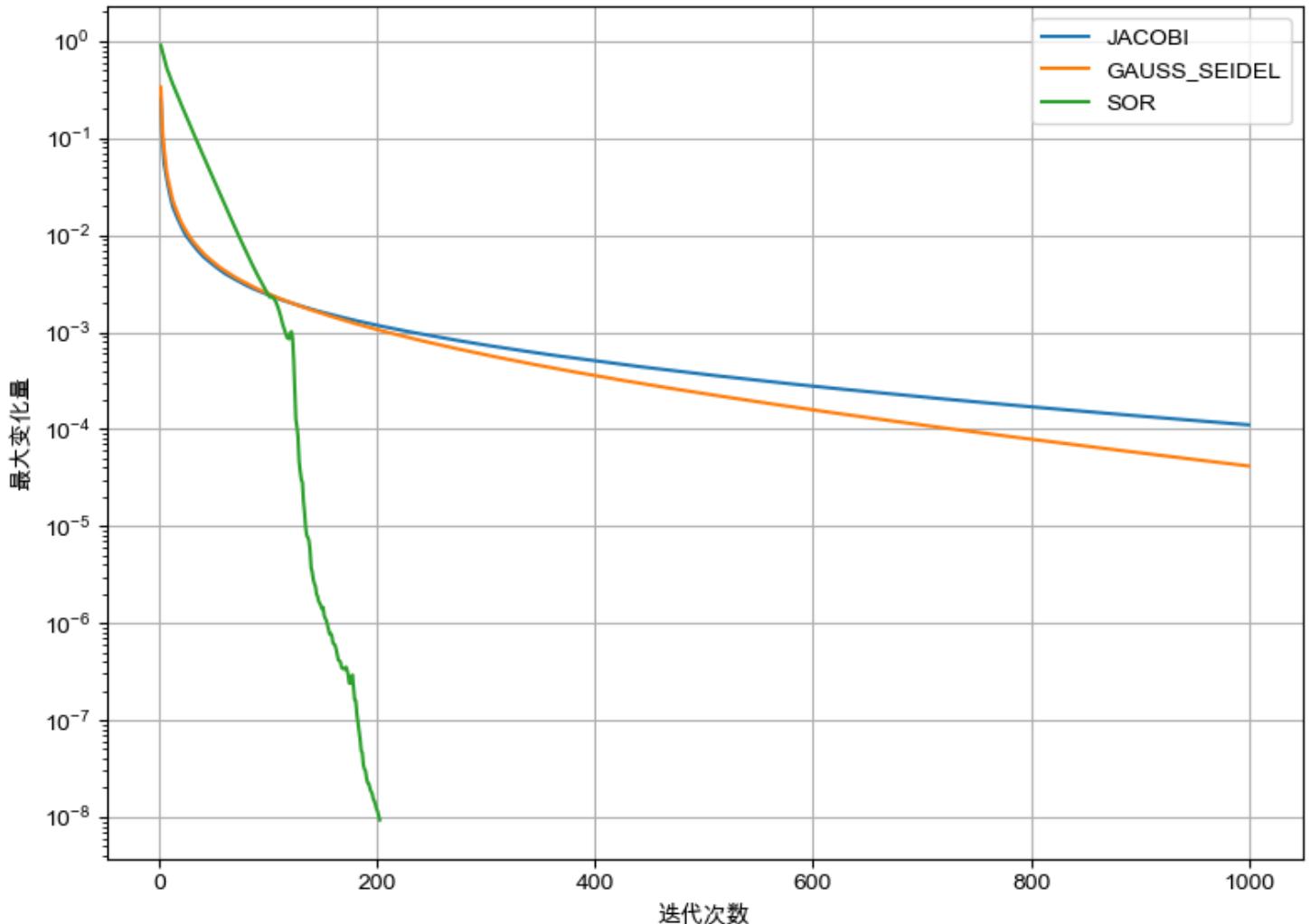


图 3: (a): 收敛曲线对比

收敛曲线的趋势并不完全如课件上的

$$r \simeq \begin{cases} \frac{1}{2}pL^2 & \text{for Jacobi's method,} \\ \frac{1}{4}pL^2 & \text{for the Gauss-Seidel method,} \\ \frac{1}{3}pL & \text{for SOR with } \omega \simeq 2/(1 + \pi/L). \end{cases}$$

不过线性趋势与二次趋势还是大概可以看出来。最初版本的程序使用上式预测收敛次数的功能也删去了，看上去 SOR 的收敛比预期更为迅猛。

### 1.4.2 Case (b): 均匀源电荷, 四边接地

```
● (base) gilbert@Gilbert-YoungMacBook src % python -u poisson.py
==== 泊松方程求解器 ====
请选择要求解的案例:
a - 无源项, 顶部电势为1V其余为0V
b - 均匀源项( $\rho/\epsilon_0 = 1 \text{ V/m}^2$ ), 边界全为0V
c - 自定义均匀源项和边界条件

请输入选项 (a/b/c): b

请输入网格点数 (Nx, Ny) 和最大迭代次数 (max_iter), 按回车使用默认值:
请输入 x 方向的网格点数 Nx (默认 50):
请输入 y 方向的网格点数 Ny (默认 50):
请输入最大迭代次数 max_iter (默认 10000): 3000
INFO:root:解析解在 n = 11, m = 49 项时达到收敛, 最大项贡献为 2.71e-21
解析解使用的傅里叶级数项数: N = 11, M = 49

使用 jacobi方法求解...
Jacobi方法达到最大迭代次数3000仍未收敛
最大偏差: 2.25e-04V
最大偏差位置: (x=0.571m, y=0.449m)
迭代次数: 3000
求解时间: 0.0443秒
2024-11-25 18:07:32.196 python[15971:375251] +[IMKClient subclass]: chose IMKClient_Modern
2024-11-25 18:07:32.196 python[15971:375251] +[IMKInputSession subclass]: chose IMKInputSession_Modern

使用 gauss_seidel方法求解...
Gauss-Seidel方法在第2537次迭代收敛
最大偏差: 1.67e-04V
最大偏差位置: (x=0.980m, y=0.469m)
迭代次数: 2537
求解时间: 4.3666秒

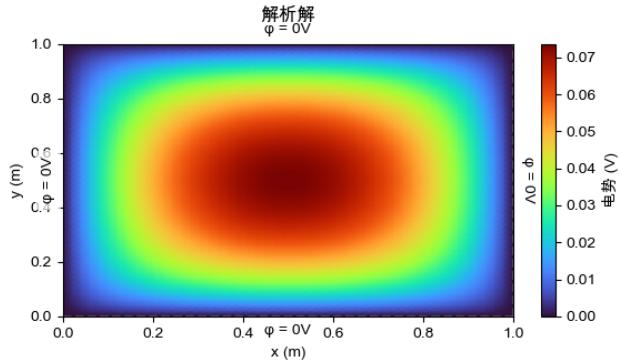
使用 sor方法求解...
使用最优松弛因子:  $\omega = 1.882$ 
SOR方法在第133次迭代收敛
最大偏差: 1.67e-04V
最大偏差位置: (x=0.980m, y=0.449m)
迭代次数: 133
求解时间: 0.2592秒
```

图 4: (b): 终端输出

本案例 Jacobi 在 3000 次迭代中都未收敛, Gauss-Seidel 方法在第 2537 次迭代侥幸收敛, 而 SOR 方法在第 133 次迭代便收敛至指定精度。

比较有意思的是下面的误差比较图, 在 x 方向出现了明显的周期性, 且可以在下面两幅子图中验证, 该条纹周期恰与上方显示的解析解 x 方向级数截断  $N$  对应, 表明该误差是解析解截断导致, 验证前文所述的, 未按照对角线法则计算的级数收敛问题。

案例(b)  
均匀源项 ( $\rho/\epsilon_0 = 1.0 \text{ V/m}^2$ )



解析解公式:

$$\phi(x, y) = \sum_{n=1, 3, 5, \dots}^N \sum_{m=1, 3, 5, \dots}^M \frac{16\rho}{\epsilon_0 \pi^2 n m \left( \left(\frac{n\pi}{L_x}\right)^2 + \left(\frac{m\pi}{L_y}\right)^2 \right)} \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right)$$

使用的傅里叶项:  
最大奇数项: N = 11, M = 49

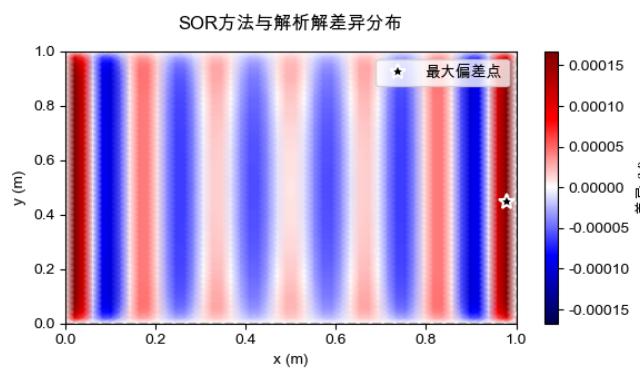
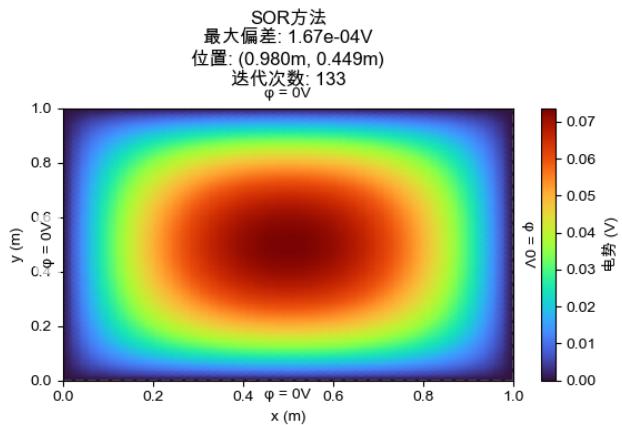
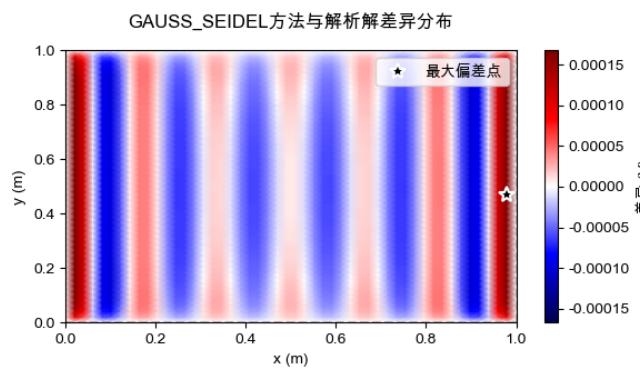
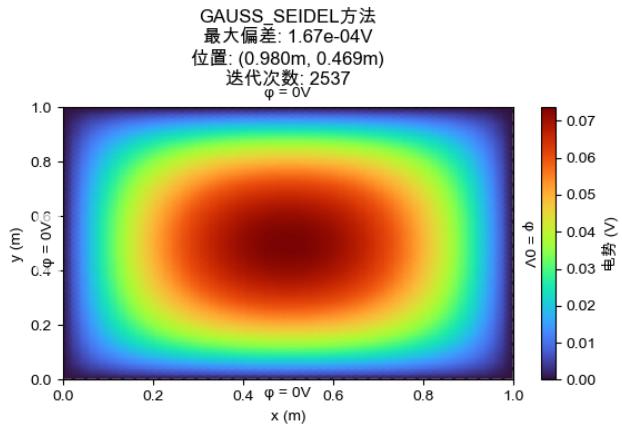
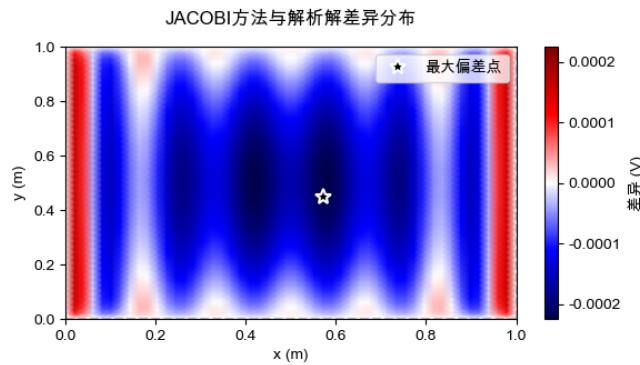
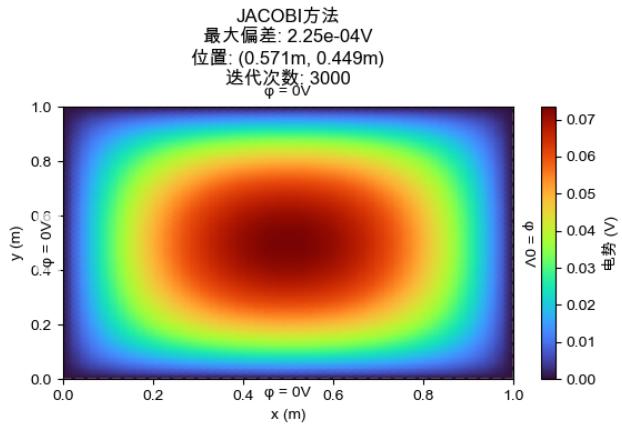


图 5: (b): 计算结果及对比

各方法的收敛曲线

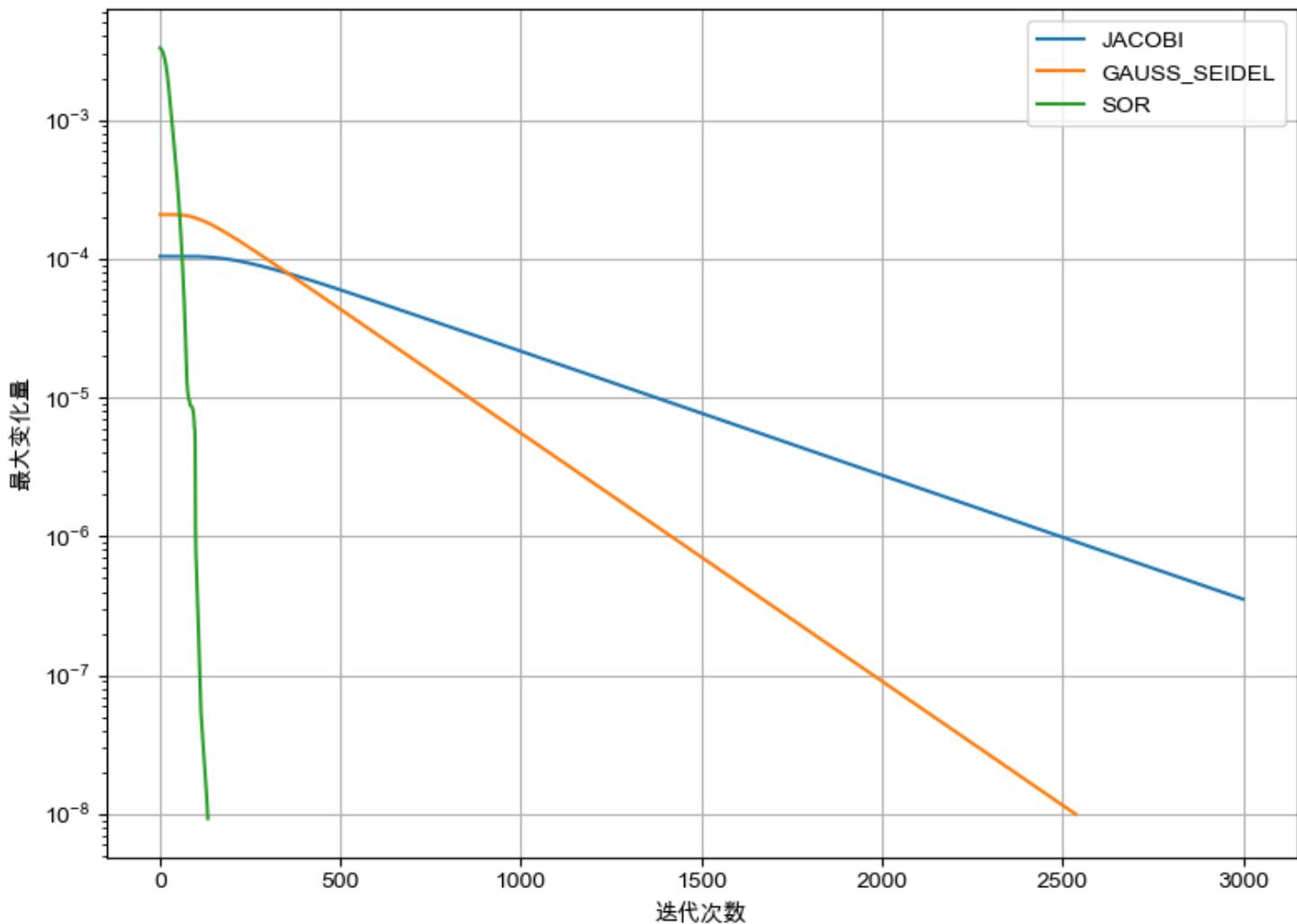


图 6: (b): 收敛曲线对比

此次收敛速度方面，仍是 SOR 大获全胜，不过尚不清楚其余两种方法的起始点收敛曲线凹凸性起源。

### 1.4.3 Case (c): 均匀源电荷，自定义边界

```
● (base) gilbert@Gilbert-YoungMacBook src % python -u poisson.py
==== 泊松方程求解器 ====
请选择要求解的案例：
a - 无源项，顶部电势为1V其余为0V
b - 均匀源项( $\rho/\epsilon_0 = 1 \text{ V/m}^2$ ), 边界全为0V
c - 自定义均匀源项和边界条件

请输入选项 (a/b/c): c

==== 请输入自定义参数 ====
(注: 所有长度单位为米(m), 电势单位为伏特(V))
请输入 x 方向长度 Lx: 1
请输入 y 方向长度 Ly: 2
请输入均匀源项大小 ( $\rho/\epsilon_0$ ): -1

请输入边界电势值:
left 边界电势: 1
right 边界电势: 2
bottom 边界电势: -1
top 边界电势: 0

请输入网格点数 (Nx, Ny) 和最大迭代次数 (max_iter), 按回车使用默认值:
请输入 x 方向的网格点数 Nx (默认 50): 100
请输入 y 方向的网格点数 Ny (默认 100): 200
请输入最大迭代次数 max_iter (默认 10000): 5000
INFO:root:解析解在 n = 11, m = 199 项时达到收敛, 最大项贡献为 3.32e-22
在 n = 447 时开始使用指数近似
解析解未在最大 n 值 3000 内收敛, 最后一项贡献为 4.25e-04
在 n = 447 时开始使用指数近似
解析解未在最大 n 值 3000 内收敛, 最后一项贡献为 8.49e-04
在 n = 113 时开始使用指数近似
解析解未在最大 n 值 3000 内收敛, 最后一项贡献为 4.25e-04
解析解计算完成
特解使用的最大奇数项: N = 11, M = 199
齐次解使用的最大傅里叶项: n = 2999
从 n = 113 开始使用指数近似

使用 jacobi 方法求解...
Jacobi方法达到最大迭代次数5000仍未收敛
最大偏差: 3.46e-01V
最大偏差位置: (x=0.505m, y=1.035m)
迭代次数: 5000
求解时间: 0.7881秒
2024-11-25 18:09:15.298 python[16107:377295]
2024-11-25 18:09:15.298 python[16107:377295]

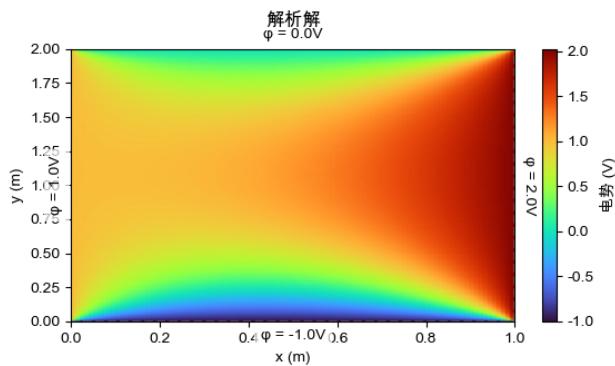
使用 gauss_seidel 方法求解...
Gauss-Seidel方法达到最大迭代次数5000仍未收敛
最大偏差: 7.29e-02V
最大偏差位置: (x=0.495m, y=0.995m)
迭代次数: 5000
求解时间: 70.1102秒

使用 sor 方法求解...
使用最优松弛因子:  $\omega = 1.952$ 
SOR方法在第439次迭代收敛
最大偏差: 2.62e-02V
最大偏差位置: (x=1.000m, y=1.960m)
迭代次数: 439
求解时间: 7.0361秒
```

图 7: (c): 终端输出

本次在矩形区域添加均匀的负电荷，四周边界条件电势均不相等，程序仍完美执行任务。

自定义案例  
均匀源项 ( $\rho/\epsilon_0 = -1.0 \text{ V/m}^2$ )



解析解公式:  
特解 (源项):

$$\phi_p(x, y) = \sum_{n=1, 3, 5, \dots}^N \sum_{m=1, 3, 5, \dots}^M \frac{16\rho}{\epsilon_0 \pi^2 n m \left(\frac{n\pi}{L_x}\right)^2 + \left(\frac{m\pi}{L_y}\right)^2} \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right)$$

特解使用的最大奇数项:  $N = 11, M = 199$

齐次解 (边界条件):

$$\phi_h(x, y) = \sum_{i=1}^4 \frac{4V_i}{\pi} \sum_{n=1, 3, 5, \dots}^N \frac{1}{n} \sin\left(\frac{n\pi x}{L_x}\right) \frac{\sinh\left(\frac{n\pi y}{L_y}\right)}{\sinh\left(\frac{n\pi L_y}{L_x}\right)}$$

齐次解使用的最大傅里叶项:  $n = 2999$   
从  $n = 113$  开始使用指数近似

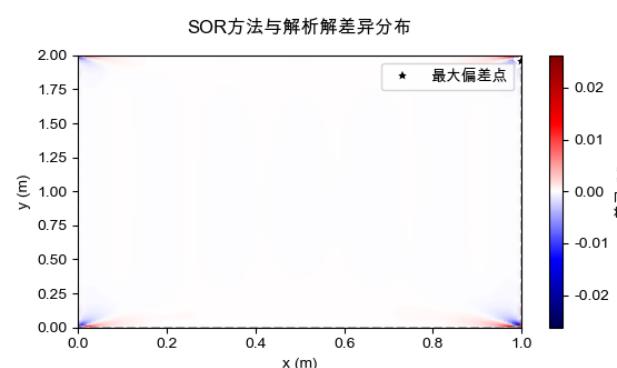
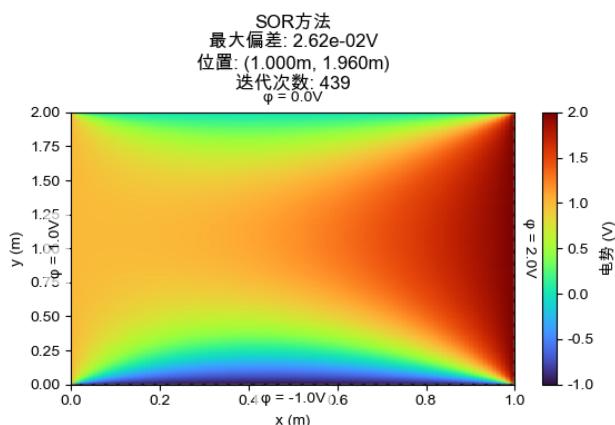
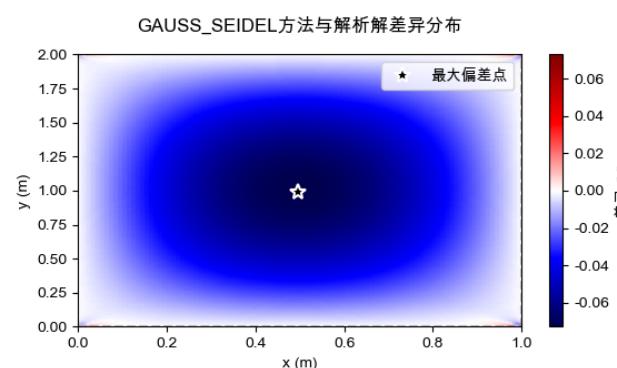
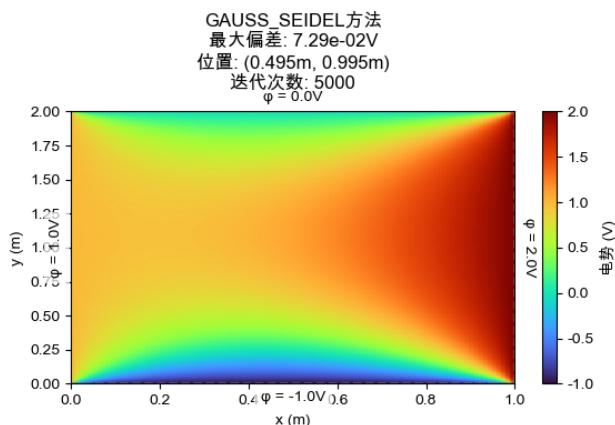
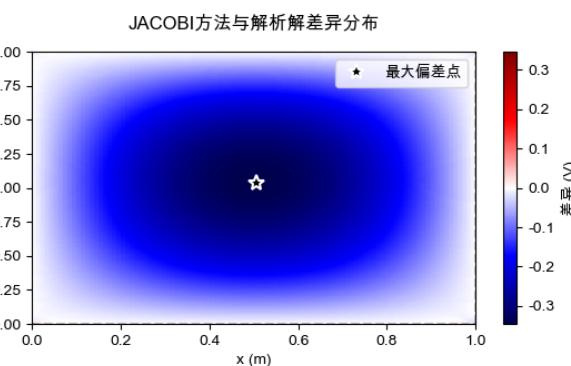
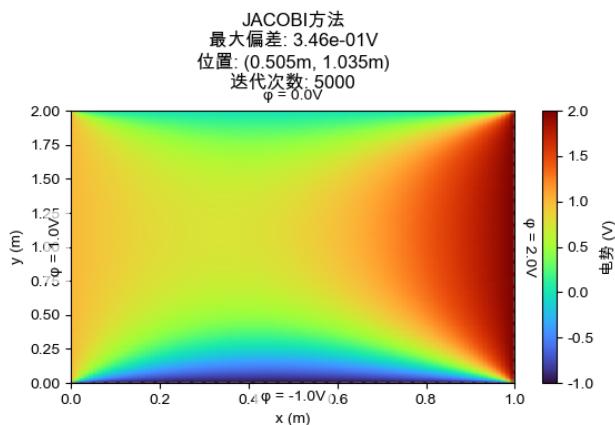


图 8: (c): 计算结果及对比

各方法的收敛曲线

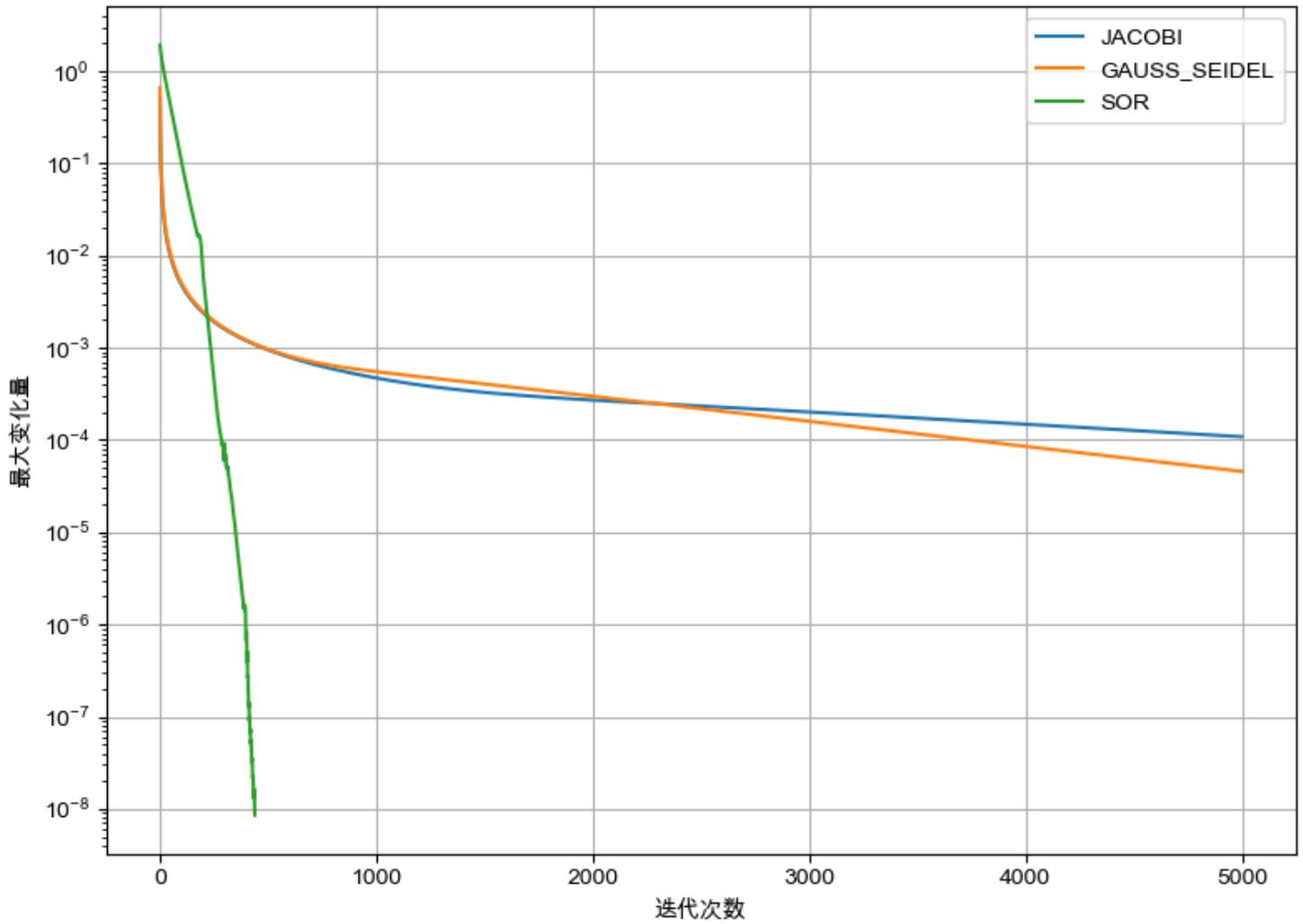


图 9: (c): 收敛曲线对比

## 2 题目 2：含时薛定谔方程求解

### 2.1 题目描述

Solve the time-dependent Schrödinger equation using both the Crank–Nicolson scheme and a stable explicit scheme. Consider the one-dimensional case and test it by applying it to the problem of a square well with a Gaussian initial state coming in from the left.

Hint: The Gaussian initial state could be expressed as:

$$\Psi(x, 0) = \sqrt{\frac{1}{\pi}} \exp \left[ ik_0 x - \frac{(x - \xi_0)^2}{2} \right].$$

### 2.2 程序描述

本程序通过 `Parameters` 类管理参数，包括网格参数（时空坐标剖分）、势阱参数（宽度、深度与中心位置）以及初始波包参数（宽度、位置与动量）。定义了一个 `SchrodingerSolver` 求解器基类，包含了 Crank–Nicolson

解法与显式解法的接口，以及一些共用的方法，如检查输入参数是否满足 Von Neumann 稳定性条件。两个求解器 `CrankNicolsonSolver` 与 `ExplicitSolver` 继承自基类，分别实现了 Crank–Nicolson 解法与显式解法。本题求解的一维含时薛定谔方程

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

在原子单位制  $\hbar = m = 1$  下化简为

$$i \frac{\partial \psi}{\partial t} = -\frac{1}{2} \frac{\partial^2 \psi}{\partial x^2} + V(x)\psi$$

离散化时记  $\psi_j^n$  为在位置  $x_j = j\Delta x$  和时间  $t^n = n\Delta t$  处的波函数值。

### 2.2.1 Crank-Nicolson 算法

类比二阶偏微分方程中的 Gauss-Seidel 迭代，隐式的 Crank-Nicolson 算法对时间导数采用前向差分，对中心差分的空间导数和势能项取时间切片  $n$  和  $n+1$  的平均，即：

$$i \frac{\psi_j^{n+1} - \psi_j^n}{\Delta t} = -\frac{1}{4} \left( \frac{\psi_{j+1}^{n+1} - 2\psi_j^{n+1} + \psi_{j-1}^{n+1}}{(\Delta x)^2} + \frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{(\Delta x)^2} \right) + \frac{1}{2} V_j (\psi_j^{n+1} + \psi_j^n)$$

可以整理为半步演化形式

$$\left( 1 + i \frac{\Delta t}{2} \hat{H} \right) \psi^{n+1} = \left( 1 - i \frac{\Delta t}{2} \hat{H} \right) \psi^n$$

其中哈密顿算符在每个时间切片上离散化为

$$\hat{H}\psi_j = -\frac{1}{2(\Delta x)^2} (\psi_{j+1} - 2\psi_j + \psi_{j-1}) + V_j \psi_j$$

实际代码实现中，构造了两个三对角矩阵，化方程为  $\mathbf{A}\psi^{n+1} = \mathbf{B}\psi^n$ ，满足  $\mathbf{A} = \mathbf{I} + i\frac{\Delta t}{2}\mathbf{H}$  与  $\mathbf{B} = \mathbf{I} - i\frac{\Delta t}{2}\mathbf{H}$ ，即

$$\mathbf{A}, \mathbf{B} = \begin{pmatrix} 1 + 2\alpha \pm \frac{i\Delta t}{2}V_1 & \mp\alpha & 0 & \cdots & 0 \\ \mp\alpha & 1 + 2\alpha \pm \frac{i\Delta t}{2}V_2 & \mp\alpha & \ddots & \vdots \\ 0 & \mp\alpha & 1 + 2\alpha \pm \frac{i\Delta t}{2}V_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \mp\alpha \\ 0 & \cdots & 0 & \mp\alpha & 1 + 2\alpha \pm \frac{i\Delta t}{2}V_{N_x} \end{pmatrix}$$

其中  $\alpha = \frac{i\Delta t}{4(\Delta x)^2}$ ，模长需满足 Von Neumann 稳定性条件。

在演化步骤中，三对角矩阵均使用 CSC 稀疏格式存储，并使用 `scipy.sparse.linalg.spsolve` 进行求解，以提高计算效率和节省内存。

### 2.2.2 显式算法

显式算法对时间导数与空间导数采用中心差分，但不对相邻切片的势能或者空间导数进行平均

$$i \frac{\psi_j^{n+1} - \psi_j^{n-1}}{2\Delta t} = -\frac{1}{2} \frac{\psi_{j+1}^n - 2\psi_j^n + \psi_{j-1}^n}{(\Delta x)^2} + V_j \psi_j^n$$

故可以直接进行显式更新

$$\psi_j^{n+1} = \psi_j^{n-1} + \frac{i\Delta t}{\Delta x^2} (\psi_{j+1}^n + \psi_{j-1}^n - 2\psi_j^n) - 2i\Delta t V_j \psi_j^n$$

化为矩阵形式即为

$$\psi^{n+1} = \psi^{n-1} + \frac{i\Delta t}{(\Delta x)^2} \mathbf{L}\psi^n - 2i\Delta t \mathbf{V}\psi^n$$

其中  $\mathbf{V}$  即离散的势能项，动能项为拉普拉斯算符

$$\mathbf{L} = \begin{pmatrix} -2 & 1 & 0 & \cdots & 1 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & 1 & -2 \end{pmatrix}$$

实际代码实现中没有显式定义  $\mathbf{L}$ ，而是在每一步中借助 `np.roll` 函数（将数组在周期边界条件下顺次移动），直接计算了  $\mathbf{L}\psi^n$ 。在第一步中因为没有  $\psi^{n-1}$ ，故使用 Crank-Nicolson 算法进行第一步演化，之后都使用显式算法进行演化。

## 2.3 伪代码

Powered by `LATEX pseudocode generator`

---

### Algorithm 4: Crank-Nicolson Method for Time Evolution (Optimized)

---

**Input:**  $\psi$  (initial wave function),  $\Delta t$ ,  $\Delta x$ ,  $V$  (potential),  $N_x$  (spatial resolution),  $N_t$  (time steps)

**Output:**  $\psi_{\text{history}}$  (wave function at all time steps)

```

1  $\alpha \leftarrow i\Delta t/(4\Delta x^2)$  ; // Precompute coefficient
2  $\mathbf{A} \leftarrow \text{ConstructMatrix}([-\alpha, 1 + 2\alpha + i\Delta t V/2, -\alpha], [-1, 0, 1], N_x)$  ; // Left-hand matrix
3  $\mathbf{B} \leftarrow \text{ConstructMatrix}([\alpha, 1 - 2\alpha - i\Delta t V/2, \alpha], [-1, 0, 1], N_x)$  ; // Right-hand matrix
4 for  $t \leftarrow 1$  to  $N_t - 1$  do
5    $\psi \leftarrow \text{spsolve}(\mathbf{A}, \mathbf{B} \cdot \psi)$  ; // Solve  $\mathbf{A}\psi^{(n+1)} = \mathbf{B}\psi^{(n)}$ 
6   Append  $\psi$  to  $\psi_{\text{history}}$  ; // Record updated wave function
7 end
8 return  $\psi_{\text{history}}$ 

```

---



---

### Algorithm 5: Explicit Time Evolution with Crank-Nicolson First Step (Optimized)

---

**Input:**  $\psi$  (initial wave function),  $\Delta t$ ,  $\Delta x$ ,  $V$  (potential),  $N_x$  (spatial resolution),  $N_t$  (time steps)

**Output:**  $\psi_{\text{history}}$  (wave function at all time steps)

```

1  $\alpha \leftarrow i\Delta t/\Delta x^2, \psi_{\text{prev}} \leftarrow \psi, \psi_{\text{history}} \leftarrow [\psi]$  ; // Initialize constants and history
2  $\psi \leftarrow \text{CrankNicolsonStep}(\psi, \Delta t, \Delta x, V)$  ; // Perform first step using CN method
3 Append  $\psi$  to  $\psi_{\text{history}}$  ; // Subsequent steps using explicit method
4 for  $t \leftarrow 2$  to  $N_t - 1$  do
5    $\psi_{\text{current}} \leftarrow \psi, \psi_{\text{jp1}} \leftarrow \text{np.roll}(\psi_{\text{current}}, 1), \psi_{\text{jm1}} \leftarrow \text{np.roll}(\psi_{\text{current}}, -1)$  ; // Compute shifts
6    $\mathbf{L}\psi^n \leftarrow \psi_{\text{jp1}} + \psi_{\text{jm1}} - 2\psi_{\text{current}}$  ; // Laplacian action
7    $\psi \leftarrow \psi_{\text{prev}} + \alpha \cdot (\mathbf{L}\psi^n) - 2i\Delta t V \cdot \psi_{\text{current}}$  ; // Update  $\psi^{(n+1)}$ 
8    $\psi_{\text{prev}} \leftarrow \psi_{\text{current}}$  ; // Update  $\psi^{(n-1)}$  for next step
9   Append  $\psi$  to  $\psi_{\text{history}}$  ; // Record updated wave function
10 end
11 return  $\psi_{\text{history}}$ 

```

---

## 2.4 结果示例

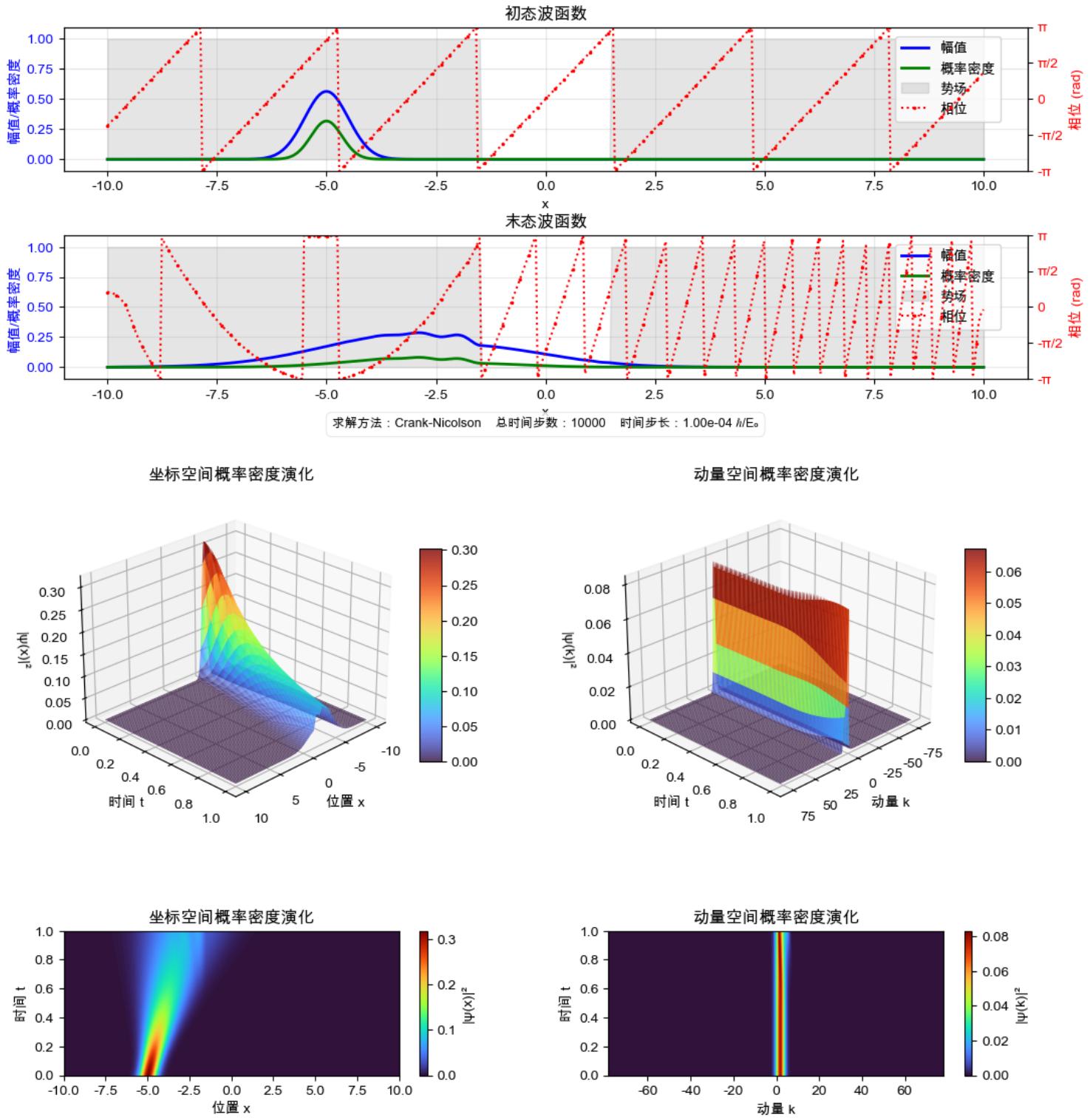


图 10: Crank–Nicolson 解法结果

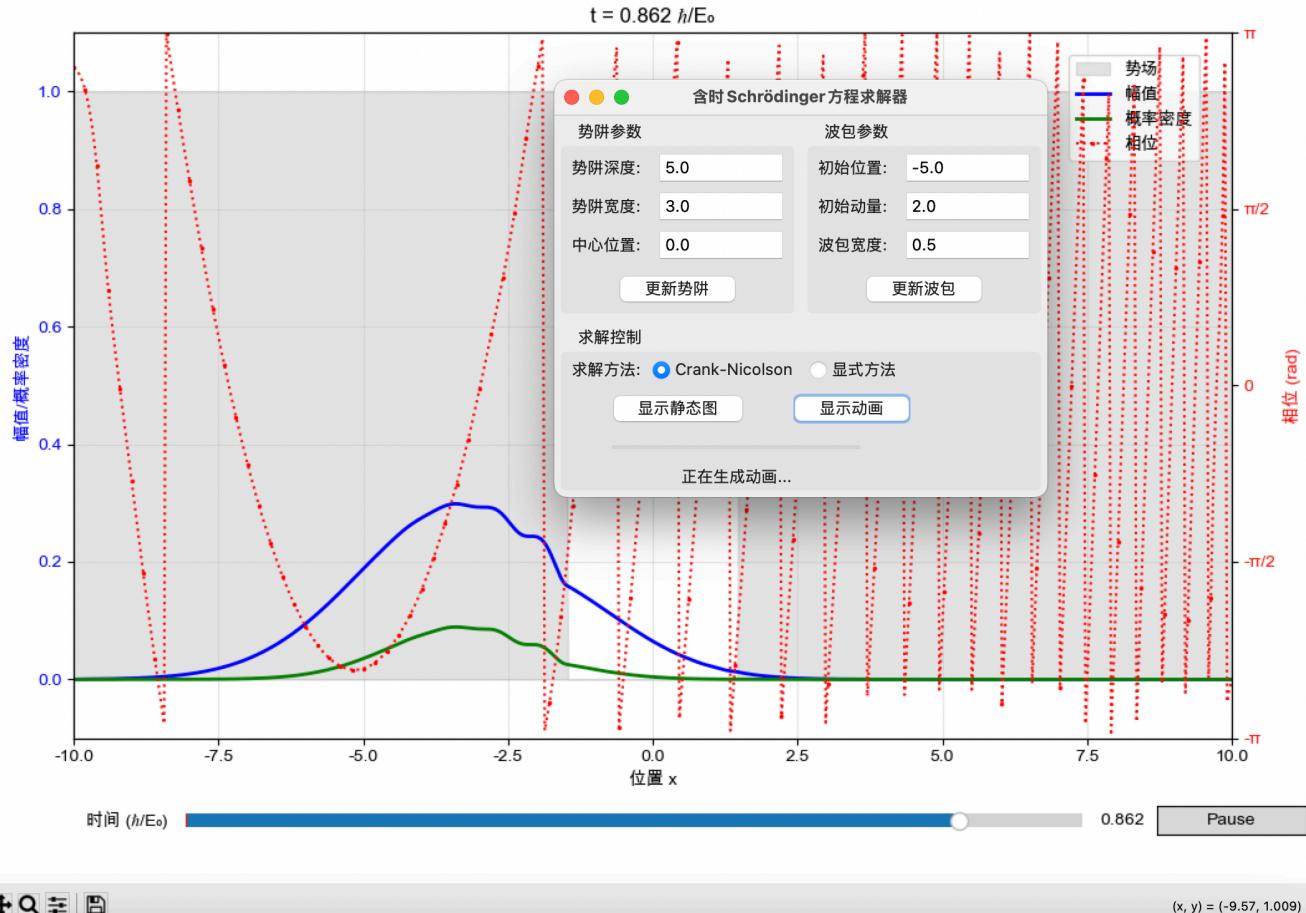
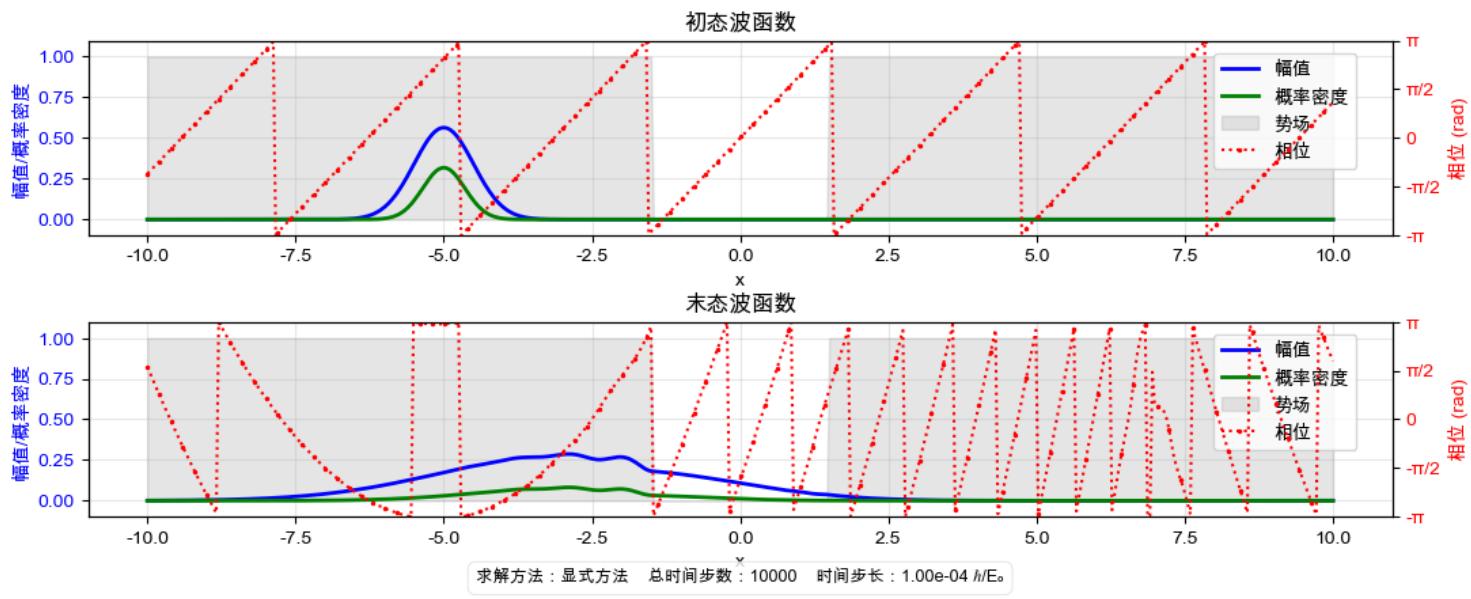
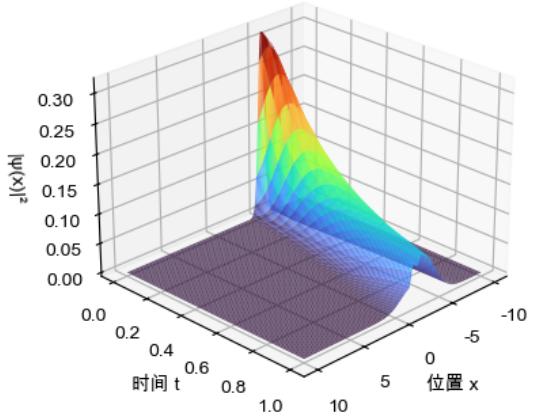


图 11: Crank-Nicolson 解法中间态（动画截图）

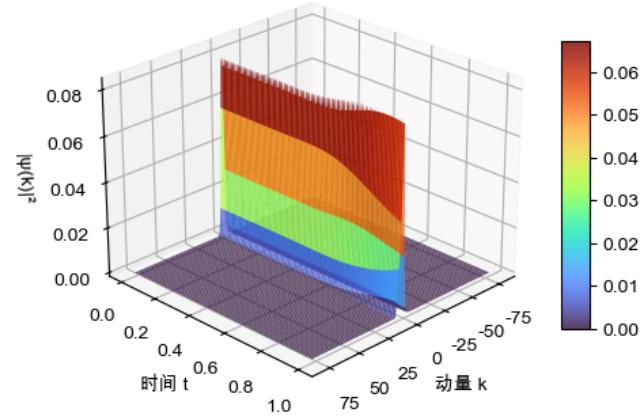
本题使用的默认参数也可从图中读出，下图表明两种解法结果一致。动画运行需要先点击“显示动画”按钮，待动画生成就绪后会显示并自动播放，有进度条可供拖动回放。动画播放器存在一些已知 bug，没力气修复了，不影响求解结果与静态图展示。



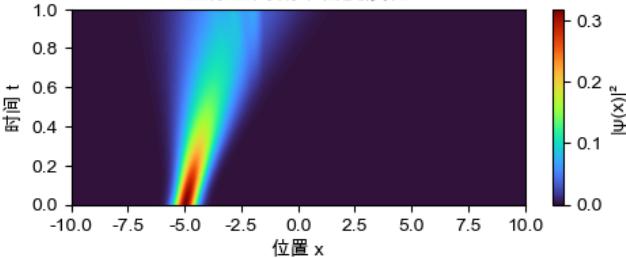
坐标空间概率密度演化



动量空间概率密度演化



坐标空间概率密度演化



动量空间概率密度演化

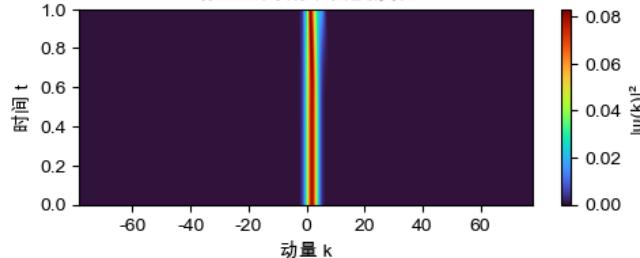


图 12: 显式解法结果

### 3 题目 3：波动方程显式求解稳定条件

#### 3.1 题目描述

Prove the stability condition of the explicit scheme of the 1D wave equation by performing Von Neumann stability analysis:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}.$$

If  $c\Delta t/\Delta x \leq 1$ , the explicit scheme is stable.

#### 3.2 证明