

多重网格-五点差分-共轭梯度法 ——求解二维泊松方程的一类边值问题： 从电荷分布求解电场的数值方法

欧纪阳 2019141220016

College of Physics, Sichuan University, Chengdu 610064, China

2021 年 5 月 14 日

摘要

本文报告了一种在二维空间中求解泊松方程的一类边值问题的方法——多重网格-五点差分-共轭梯度法，该算法为有限差分法的改进方法，利用五点差分法作为数值微分近似，构建对角线稀疏矩阵，以共轭梯度方法为线性系统求解器，采用 V 循环多重网格方法加速迭代。本文还展示了用 Python 求解在有限矩形区域内给定电荷分布，并已知电势的边值的条件下的，该区域内的电势和电场分布。相较于高斯-赛德尔、过松弛迭代等求解算法，其收敛速度更快，精度更高。在复杂的边值条件下，与普通网格下的共轭梯度迭代相比，可快速降低误差。

关键词：多重网格法 共轭梯度法 稀疏矩阵 泊松方程 边值问题 Python

目录

1 介绍	2
2 方法	2
2.1 五点差分方法	2
2.2 系数矩阵的组装	3
2.3 添加一类边值条件	4
2.4 共轭梯度方法	4
2.5 多重网格方法	5
2.6 可视化与误差分析	5
3 结果	7
3.1 二维正弦电荷分布	7
3.2 点电荷分布	7
3.3 连续体电荷分布	7
4 讨论	8
5 附录	9

1 介绍

求解偏微分方程的数值方法有许多，其步骤大致分为以下几个部分：用差分代替微分，带入边界条件组装系数矩阵，使用合适的线性方程求解器进行求解。以二维泊松方程为例，其只含有两项对相同坐标的二阶偏微分，最常用的差分方法就是对每一个坐标取三点中心差分，对于线性求解迭代方法，主要有雅克比方法、高斯-赛德尔方法、连续过松弛迭代方法、共轭梯度方法等。雅克比方法是线性方程组系统中的一种不动点迭代方法，把系数矩阵 A 分解为主对角线矩阵 D ，上三角矩阵 U 和下三角矩阵 L ，其不动点迭代形式为，

$$x_{i+1} = D^{-1}[b - (L + U)x_i] \quad (1)$$

其中 b 表示常数项。与雅克比方法紧密相关的迭代方法是高斯-赛德尔方法，两者之间的差别是后者最近更新的未知变量的值在每一部中都使用，即使更新发生在当前步骤，其迭代形式为，

$$x_{i+1} = D^{-1}(b - Ux_i - Lx_{i+1}) \quad (2)$$

因此其收敛速度常常比雅克比方法更快。连续过松弛方法是使用高斯-赛德尔的求解方向，并使用过松弛加以加快收敛速度，其迭代形式为

$$x_{i+1} = (\omega L + D)^{-1}[(1 - \omega)Dx_i - \omega Ux_i] + \omega(D + \omega L)^{-1}b \quad (3)$$

其中 ω 成为松弛因子，当 $\omega > 1$ 时成为过松弛，对于不同的新系统有不同的 ω 最优选择。过松弛迭代对于一般的线性系统已经达到很高的迭代速度，但是对于具有大量零的稀疏矩阵来说，其效率往往较低，例如二维泊松方程，其系数矩阵只在五条对角线上具有非零元素，随着网格的精细，除了迭代速度减慢以外，存储矩阵所需的内存也不断增大，这对计算机的要求大大提高。以上的这些方法都是在确定的网格下进行迭代，也就是说在整个迭代过程中网格的精度是不会变化的，但是在一些系统中，高频信息与低频信息的收敛速度是不一样的，一种解决的思路就是运用层次离散化的方法进行求解。

本文报告了一种求解泊松方程的高效方法：多重网格-五点差分-共轭梯度法。该方法在每一级网格中，首先利用五点差分法对偏微分做近似，将二维偏微分方程改写为一维的线性方程组，根据边界条件组装好系数矩阵后，利用对角稀疏矩阵进行储存，该矩阵为一个对角正定矩阵（或对角负定矩阵），因此可以利用共轭梯度法作为线性求解器进行求解。在迭代求解的过程中运用 V 循环多重网格方法作为预调节器，现在低精度的网格中进行迭代，再逐步将计算结果插值到更加精细的网格中进行迭代，以达到提高收敛速度，并同时降低高频误差与低频误差的目的。

2 方法

报告的方法的最终目的是通过在二维空间中求解泊松方程的边值问题，

$$\nabla^2 \varphi = -\frac{\rho}{\varepsilon} \Rightarrow \frac{\partial^2 \varphi(x, y)}{\partial x^2} + \frac{\partial^2 \varphi(x, y)}{\partial y^2} = f(x, y) \quad (4)$$

从而通过 $\vec{E} = -\nabla \varphi$ 求解电场强度矢量。(4) 式为一个二阶偏微分方程，计算机无法直接求解，报告的方法实质上是有限差分法的扩展，先通过数值微分将偏微分方程化为线性方程，再在合适的网格上施以合适的线性求解器进行求解。

2.1 五点差分方法

在说明二维问题前，先讨论一维数值微分，对于一个三阶可微函数 $f(x)$ ，根据泰勒定理有，

$$f(x + h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f^{(3)}(c_1) \quad (5)$$

$$f(x - h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f^{(3)}(c_2) \quad (6)$$

其中 $x - h < c_2 < x < c_1 < x + h$, 最后一项为误差项, 两式相减可以得到一阶导数的三点中心差分公式,

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f^{(3)}(c) = \frac{f(x+h) - f(x-h)}{2h} + O(x^3) \quad (7)$$

用该公式在离散情况下对 $\vec{E} = -\nabla\varphi$ 做近似可得,

$$\vec{E}_{i,j} = \left(-\frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2h_x}, -\frac{\varphi_{i,j+1} - \varphi_{i,j-1}}{2h_y} \right) \quad (8)$$

其中 $\{\varphi_{i,j} | 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$, h_x, h_y 分别为 x 方向和 y 方向上的网格精度。

类似地, 该方法可以得到更高阶的差分公式, 对四阶可微函数 $f(x)$, 根据泰勒定理有,

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(c_1) \quad (9)$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(c_2) \quad (10)$$

其中 $x - h < c_2 < x < c_1 < x + h$, 最后一项为误差项, 两式相加可以得到二阶导数的三点中心差分公式,

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12}f^{(4)}(c) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + O(x^4) \quad (11)$$

对泊松方程中的两个微分算子在离散情况下, 分别应用二阶导数的三点中心差分公式,

$$\frac{\partial^2 \varphi_{i,j}}{\partial x^2} = \frac{\varphi_{i-1,j} - 2\varphi_{i,j} + \varphi_{i+1,j}}{h_x^2} \quad \frac{\partial^2 \varphi_{i,j}}{\partial y^2} = \frac{\varphi_{i,j-1} - 2\varphi_{i,j} + \varphi_{i,j+1}}{h_y^2} \quad (12)$$

由于每一个网格的偏微分需要五个网格进行计算, 故称为五点差分法。由此便得到了应用五点差分法近似后得到的泊松方程,

$$\frac{\varphi_{i-1,j} + \varphi_{i+1,j}}{h_x^2} - 2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right)\varphi_{i,j} + \frac{\varphi_{i,j-1} + \varphi_{i,j+1}}{h_y^2} = f_{i,j} \quad (13)$$

2.2 系数矩阵的组装

由于求解的方程是二维的, 需要将其重塑回到一维才能求解,

$$u_{i+jm} = a_{i,j}, \{a_{i,j} | 0 \leq i \leq m-1, 0 \leq j \leq n-1\} \quad (14)$$

例如对于一个 $A_{4 \times 4}$ 的矩阵将会变换为,

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \Rightarrow [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].T \quad (15)$$

对于 (13) 式, φ 与 f 的大小为 (m, n) , 利用以上方法把 (13) 式改写为 $Ax = b$ 的形式, 即 $\varphi \rightarrow x, f \rightarrow b, \nabla^2 \rightarrow A$, 其中系数矩阵 A 为,

$$\left[\begin{array}{ccccccccc} -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \frac{1}{h_y^2} & & & & & \\ \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \frac{1}{h_x^2} & & & & \\ \vdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \frac{1}{h_x^2} & & & \\ \frac{1}{h_y^2} & \vdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \frac{1}{h_y^2} & & \\ & \ddots & \\ & \frac{1}{h_y^2} & \cdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \vdots & \frac{1}{h_y^2} \\ & & \frac{1}{h_y^2} & \cdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots & \vdots \\ & & & \frac{1}{h_y^2} & \cdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} & \cdots \\ & & & & \frac{1}{h_y^2} & \cdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) & \frac{1}{h_x^2} \\ & & & & & \frac{1}{h_y^2} & \cdots & \frac{1}{h_x^2} & -2\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right) \end{array} \right] \quad (16)$$

那么做变换后得到的 x, b 的大小为为 $(1, mn)$, 系数 A 矩阵的大小为 (mn, mn) , 矩阵中同一行同一列的 $\frac{1}{h_x^2}$ 与 $\frac{1}{h_y^2}$ 之间的间距为 $m - 1$, 即矩阵中的 $\cdots :$ 表示占据 $m - 2$ 个元素, 并且 1 级、 -1 级副对角线上的元素, 从第 m 个元素开始, 每隔 $m - 1$ 个元素应该为零 ((16) 式中没有体现出来)。例如对于一个 4×4 的网格, 假设 $h_x = h_y = 1$, 其系数矩阵 A 写作,

$$\left[\begin{array}{cccc} -4 & 1 & & 1 \\ 1 & -4 & 1 & & 1 \\ & 1 & -4 & 1 & & 1 \\ & & 1 & -4 & 0 & & 1 \\ 1 & & & 0 & -4 & 1 & & 1 \\ & 1 & & & 1 & -4 & 1 & & 1 \\ & & 1 & & & 1 & -4 & 1 & & 1 \\ & & & 1 & & & 0 & -4 & 1 & & 1 \\ & & & & 1 & & & 1 & -4 & 1 & & 1 \\ & & & & & 1 & & & 1 & -4 & 0 & & 1 \\ & & & & & & 1 & & & 0 & -4 & 1 & & 1 \\ & & & & & & & 1 & & & 1 & -4 & 1 & & 1 \\ & & & & & & & & 1 & & & 1 & -4 & 1 & & 1 \\ & & & & & & & & & 1 & & & 1 & -4 & 1 & & 1 \end{array} \right] \quad (17)$$

显然该矩阵为一个稀疏矩阵, 只在主对角线、 $1, -1, m, -m$ 级副对角线五条对角线上有元素, 因此在程序中构建此矩阵, 并将其储存到内存中时, 将应用**对角稀疏矩阵方法**, 即内存器只储存非零对角线上的元素, 在编写中使用 `scipy.sparse.dia_matrix` 库进行矩阵的存储与运算。

2.3 添加一类边值条件

对于 (16) 式中的矩阵, 如果没有加边界条件的话, 求解过程会非常缓慢, 而且最终的求解结果显示, 其等价的边界条件为四个边界均满足 $\varphi = 0$, 因此在求解之前应施加一定的边界条件。在经改写后得到的线性方程组中添加一类边界, ($\varphi = \phi(q)$, 其中 q 为边界上的坐标), 的方法十分简单, 只需在系数矩阵 A 中找到边界点对应的行, 将对角线上的元素该写为 1, 再将 b 中对应的元素改写为对应的 $\phi(q)$ 即可。

2.4 共轭梯度方法

在众多的线性方程求解器中, 目前共轭梯度方法是求解稀疏矩阵最优化的方法, 特别是在引入有效的预条件后, 许多其他方法难以处理的问题使用该方法可以高效的解决, 其收敛速度远高于高斯-赛德尔、过松弛迭代等方法, 因此在求解线性近似的偏微分方程时, 首选使用共轭梯度方法, 可以在有限步骤获得线性系统 $Ax = b$ 的解, 其伪代码如算法1。

算法1通过使用预条件方法, 可以使迭代收敛速度加快, 迭代方法的收敛率通常直接或间接依赖于系数矩阵 A 的条件数, 预条件方法的思想便是降低问题中的条件数。在对称连续过松弛中, 预条件子定义为,

$$M = (D + \omega L)D^{-1}(D + \omega U) \quad (18)$$

其中 $A = L + D + U$ 被分解为下三角部分、对角线、以及上三角部分, $0 < \omega < 2$, 当 $\omega = 1$ 时被称为高斯-赛德尔与条件子。虽然预条件可以大大的加快迭代的速度, 但是如果预条件子矩阵难以求逆, 一般情况下反而会适得其反, 故在本文中运用的是无预条件的共轭梯度方法。

```

Input:  $A$ : 系数矩阵
       $b$ : 常数项
       $x_0$ : 初始预测值
1  $d_0 = r_0 = b - Ax_0$ 
2 for  $k = 0, 1, 2, \dots, n-1$  do
3   if  $r_k = 0$  then
4     | break
5   end
6    $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}$ 
7    $x_{k+1} = x_k + \alpha_k d_k$ 
8    $r_{k+1} = r_k - \alpha_k A d_k$ , 计算余项
9    $\beta = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$ 
10 end

```

Output: x_k : 线性方程组的解

Algorithm 1: 共轭梯度方法

2.5 多重网格方法

在数值分析中，多重网格法是一种使用层次离散化来求解微分方程的方法，是多分辨率方法的一个例子。由于直接在高分辨率（用于求解的间隔小）上进行求解时对于低频部分收敛较慢，与间隔的平方成反比，便想到先在低分辨率（间隔较大）上进行求解，然后再进行插值，提高其分辨率，再在更高分辨率进行计算，这样的方法就叫做多重网格方法，且迭代过程主要包含以下几个重要的组成部分：

- 平滑 (smoothing): 用线性求解器进行迭代，降低高频误差。
- 计算残差 (Residual Computation): 在平滑之后计算剩余误差 (残差) $r_i = b - Ax_i$ 。
- 限制 (Restriction): 降低采样率，把残差放到更加粗糙的网格中。
- 延拓 (prolongation): 将粗网格计算的修正值插值到更精细的网格中
- 修正 (Correction): 将延拓的修正值添加到精细网格中。

多重网格方法有多种变体，但均包含以上步骤，在本文中运用的是最基础的 **V 循环多重网格方法**，因其迭代过程为逐次预光滑降低到最粗糙的网格中，再逐次求解插值到最精细的网格中，所形成的迭代图像为一个“V”字形，故得名 V 循环多重网格方法，其重要步骤如算法2，也可以用图1来表示：

算法2中，需要先对进入迭代的预测解 φ 进行一次预平滑，即使用求解器对其进行较少次数的迭代，使其较靠近精确解，本文使用的求解器为共轭梯度方法，在编写中一般要求在预平滑中迭代 3 ~ 5 次即可；插值后进行的后平滑一般按照设定的精度进行，每相邻两级网格之间的网格精度一般是翻一倍，对于求解精度每一级网格设置为相同即可；对于限制算子 (Restriction) 和延拓 (Prolongation) 均采用双线性插值方法，其为一维两点线性插值的二维推广方法，限于篇幅，此处不再过多赘述。

2.6 可视化与误差分析

对于电势的可视化，本文使用 matplotlib.pyplot 库中，利用不同颜色来标注大小的 pcolor 函数来绘制电势在空间中的分布，再使用 contour 函数绘制其等势线；对于电场的可视化，将在运用 pcolor 函数来绘制电场强度大小在空间中的分布之外，利用 quiver 函数辅以 19×19 个箭头来表示电场强度矢量的方向。误差分析方面，将对一组拥有解析解情况的泊松方程进行数值计算，并与精确解进行对比，计算均方误差 RMS。

Input: φ : 初始预测值

f : 常数项

h : 最精细网格精度

h_m : 最粗糙网格精度

1 **Function** VCycle(φ, f, h):

```

2    $\varphi = \text{smoothing}(\varphi, f, h)$  ' 预平滑 (pre-smoothing)
3   res = residual( $\varphi, f, h$ )
4   rhs = restriction (res)
5   eps = zeros (size (rhs))
6   if  $h < h_m$  then
7       | VCycle( $\varphi, f, 2h$ )
8   else
9       | smoothing( $\varphi, f, 2h$ )
10  end
11   $\varphi = \varphi + \text{prolongation} (\text{eps})$ 
12   $\varphi = \text{smoothing}(\varphi, f, h)$  ' 后平滑 (post-smoothing)
13  return  $\varphi$ 
14 end
```

Output: φ : 线性方程组的解

Algorithm 2: 多重网格方法

Multigrid V-Cycle: Solving \mathbf{PHI} in PDE $\mathbf{f}(\mathbf{PHI}) = \mathbf{F}$

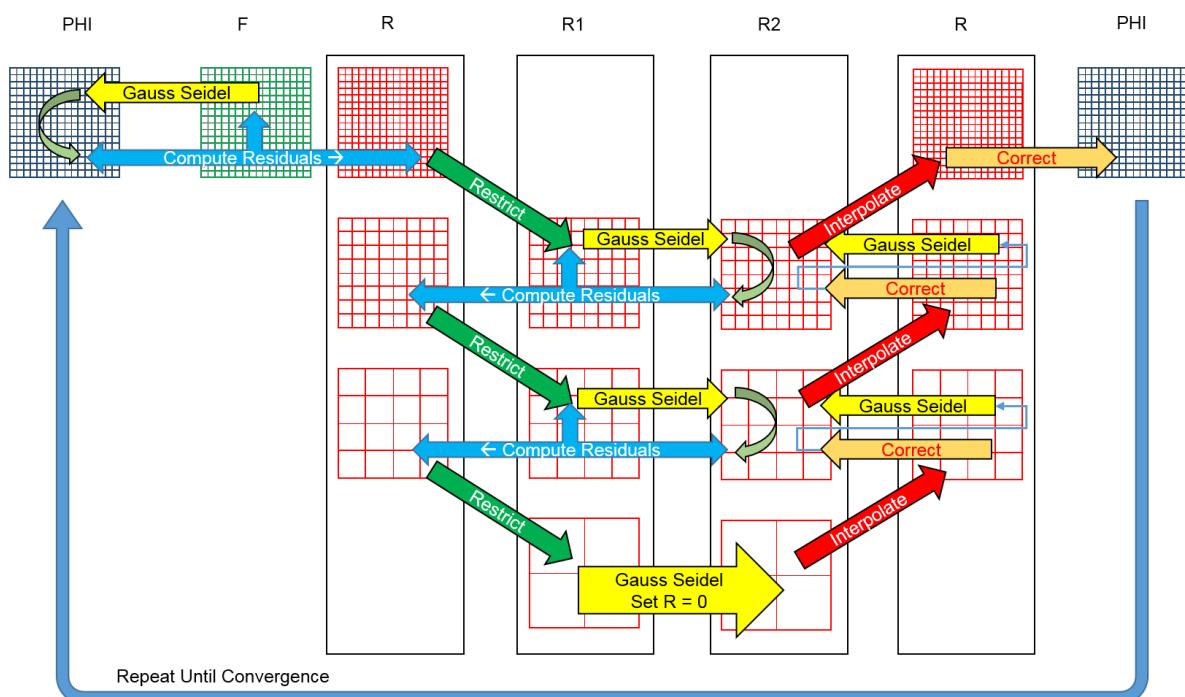


图 1: V 循环多重网格方法的示意图

3 结果

该部分展示了一些数值计算的例子，为了证明该方法的有效性，将会给予一组具有解析解的电荷分布及边值条件进行求解，并与理论上的解析解的结果进行对比，除此之外还将展示点电荷、电偶极子、带点细棍等所产生的电场在有限空间中的分布。

3.1 二维正弦电荷分布

利用多重网格-五点差分-共轭梯度法和直接共轭梯度方法（只在一级网格中进行迭代）对以下边值问题进行求解，

$$\left\{ \begin{array}{l} \frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = -\frac{\rho}{\epsilon} = -2\pi^2 \sin(\pi x) \sin(\pi y) \\ \varphi = 0, x = \pm 1 \end{array} \right. \quad (19a)$$

$$\left\{ \begin{array}{l} \varphi = 0, y = \pm 1 \end{array} \right. \quad (19b)$$

$$\left\{ \begin{array}{l} \varphi = 0, y = \pm 1 \end{array} \right. \quad (19c)$$

显然该系统具有解析解，并可推导得到电场强度矢量的表达式，

$$\varphi = \sin(\pi x) \sin(\pi y) \quad E = -\pi \cos(\pi x) \sin(\pi y) \vec{e}_x + \pi \sin(\pi x) \cos(\pi y) \vec{e}_y \quad (20)$$

在进行数值计算时，设置求解区域为 $S = \{(x, y) | -1 \leq x \leq 1, -1 \leq y \leq 1\}$ 的正方形区域，在两个维度上的网格精度均为 10^{-3} ，即求解区域的网格大小为 2000×2000 ，调节求解参数，多重网格等级为 3，预平滑迭代次数为 3，后平滑迭代求解精度为 10^{-10} 。最终求解结果如图2(a) 所示，实际计算误差 $RMS = 4.112377 \times 10^{-7}$ ，数值的后平滑迭代过程在三级网格中经历了 1 次，在二级网格中经历了 7 次，在一级网格中经历了 43 次，包括预平滑、限制、插值、后平滑等所有过程在内的 10 次平均总耗时约为 6.89692s，与直接共轭梯度方法的 55 次迭代，10 次平均总耗时 10.2639s 相比，误差降低的速度要高出许多，误差下降曲线如图2(b) 所示，蓝色曲线从最高一级粗网格后迭代开始是记录误差，从第九个数据点开始回到一级精细网格，虽然此处误差较大，但是可以以很快的速度降低误差，因为前 8 次迭代是在粗化的网格下进行，所以所报告的算法总体效率要高于直接共轭梯度法。

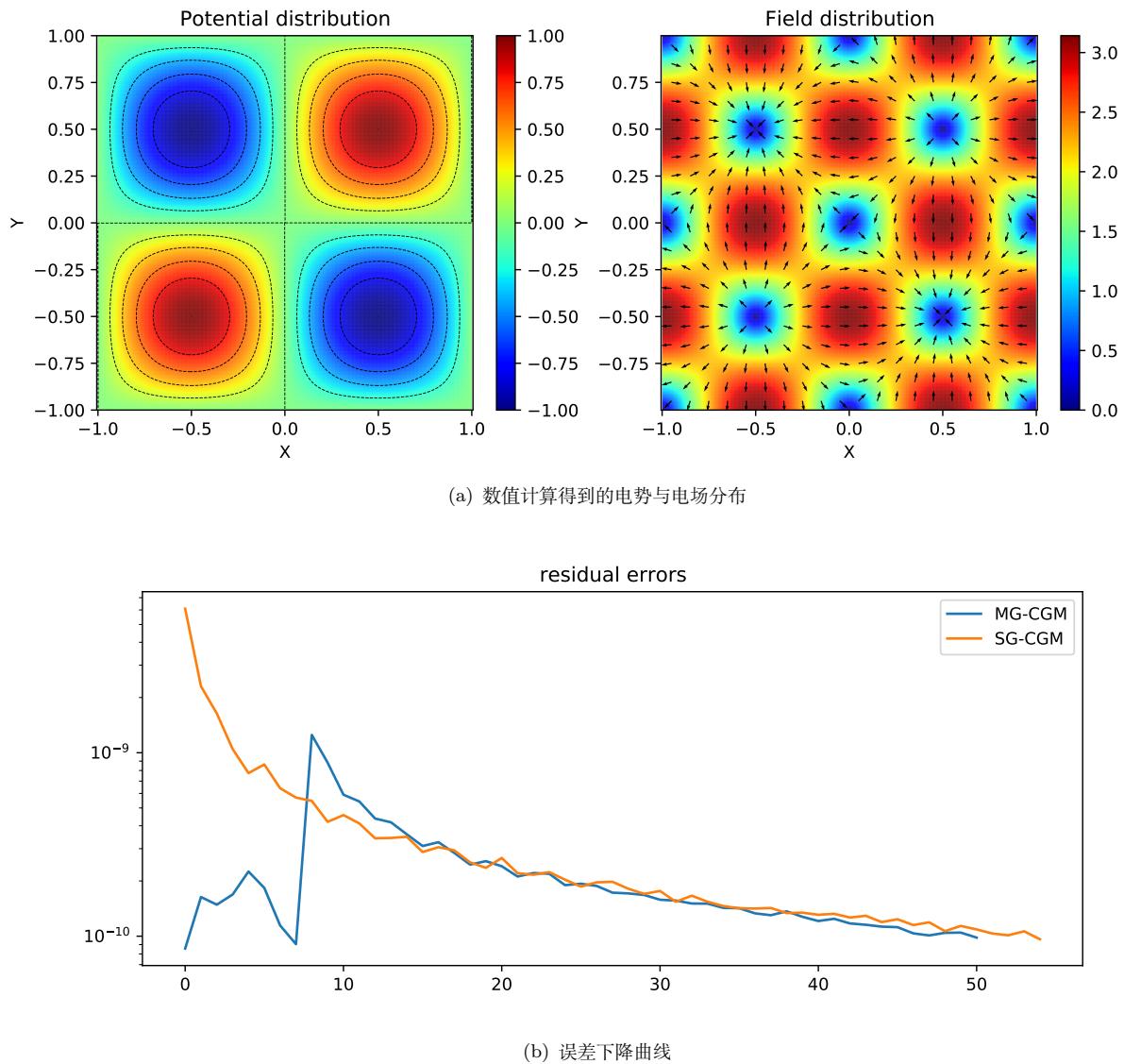
3.2 点电荷分布

设置求解区域为 $S = \{(x, y) | -1 \leq x \leq 1, -1 \leq y \leq 1\}$ 的正方形区域，在两个维度上的网格精度均为 2×10^{-3} ，即求解区域的网格大小为 1000×1000 ，调整多重网格等级为 4，预平滑迭代次数为 4，后平滑迭代求解精度为 10^{-5} 。在坐标原点放置一个 $Q/\epsilon = 1/h_x h_y = 0.25 \times 10^6$ 的点电荷，边界条件电势为零，即 (19b)(19c) 式，计算其电场分布，迭代结果如图3(a) 所示。在相同的网格和求解参数下，另取两个绝对值相等的正负电荷组成电偶极子，分别放置在 $(-0.2, 0), (0.2, 0)$ 的位置进行求解，其结果如图3(b) 所示。

由于该问题难以给出解析解，在此处只给出定性的法分析，可以看到在正电荷处，电场方向背离电荷，在负电荷处电场方向指向电荷，在越靠近电荷的位置电场越强；在靠近边界处，电场方向均垂直于边界，符合所给定的边界条件，边界附近电场方向变化大的位置，电场强度大小较小。

3.3 连续体电荷分布

求解区域以及迭代参数与点电荷分布的数值实验设置相同，放置一根长 0.8，中心位于原点的沿 x 方向的带电细棒，电荷密度为 $\rho/(\epsilon h_x h_y) = 1$ ，边界条件设置为电势为零，即 (19b)(19c) 式，电场的计算结果如图3(3) 所示。在相同的求解参数下，求解三根带电细棒在有限空间中的电场分布，计算结果如图3(4) 所示，第一、二根细棒长 0.6，沿 y 轴方向，电荷密度为 $\rho/(\epsilon h_x h_y) = 1$ ，中心分别位于 $(-0.4, 0.3), (0.4, 0.3)$ 处，第三根细棒长 0.8，沿 x 轴方向，电荷密度为 $\rho/(\epsilon h_x h_y) = -2$ ，中心位于 $(0, -0.6)$ 。数值实验的结果符合静电场规律，在边界处电场方向垂直与边界，在带电棒附近，电场方向几乎垂直与细棒。



(a) 中左图展示的是电势在求解空间中的分布情况，黑色虚线表示等势面；右图展示的是电场强度在求解空间中的分布情况，颜色的不同表示电场强度大小，箭头表示箭头根部所在的位置的电场强度矢量的方向。(b) 为多重网格-共轭梯度法 (MG-CGM) 后迭代过程 (蓝色)、直接共轭梯度法 (SG-CGM) (橙色) 的误差下降曲线。

图 2: 二维正弦电荷分布的数值计算结果

4 讨论

本文报告了一种数值求解泊松方程的方法，并举例与单网格算法相比，说明了其高效性与实用性，可视化图像也清晰的展现出了求解结果。目前对于该方法的讨论还不够全面，对于其在一些复杂的电荷分布的可用性还待进一步论证。目前所运用的多重网格方法循环模式是最基础的 V 循环，还有 F 循环和 W 循环在特定条件下会展现出更快的收敛速度。对于插值算法，报告的方法中采用的是最简单的双性插值，从图2(b)可以看出插值的误差还处于一个较低的水准，每次提升网格精度都会导致误差大幅度的提高，可以看到在蓝色折线的第二个点与第八个点都有急剧的误差升高现象，这是由于插值不够平滑所带来的，可以考虑运用双三次插值的方法进行插值，可以在细化网格时获得更低的误差值。

多重网格方法作为预调节器，除了可以用来提高偏微分方程的求解速度以外，对于其他的线性系统，乃至非线性系统的迭代提速，降低误差都有很大的实用价值。对于一般的非稀疏线性方程组，可以配合连续过松弛迭代方法进行求解，同样可以获得十分可观的结果。

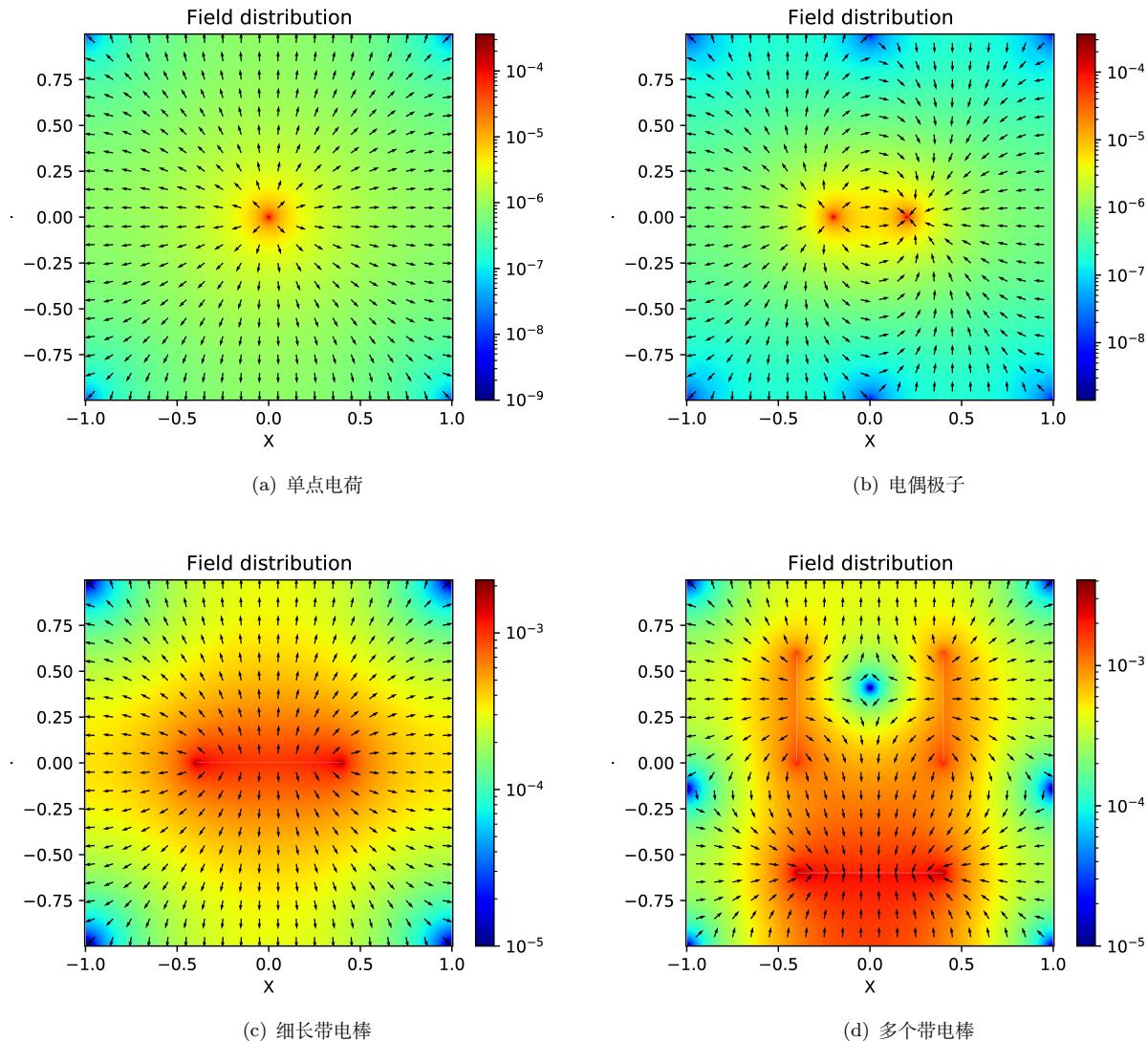


图 3: 二维点电荷分布及连续体电荷分布的数值计算电场可视化

5 附录

本文二维正弦电荷分布的数值求解的完整 Python 代码如下，其余的代码可以通过公开的 GitHub 数据库查看<https://github.com/812610357/Electrodynamics>，欢迎标星收藏跟随关注。

```
1 import time
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import pandas as pd
5 from scipy import sparse
6
7 ##### 设置求解参数 #####
8 postIterError = 1e-10 # 后平滑共轭梯度迭代误差
9 mainIterTimes = 3 # 多重网格迭代次数，即网格粗化等级
10 preIterTimes = 3 # 预平滑共轭梯度迭代次数
11 error = np.array([])
12
13 ##### 定义求解区域 #####
14 xMin = -1
```

```
15     xMax = 1
16     yMin = -1
17     yMax = 1
18     d = np.array([1e-3, 1e-3]) # [y,x]
19     x = np.arange(xMin, xMax+d[1], d[1])
20     y = np.arange(yMin, yMax+d[0], d[0])
21     xx, yy = np.meshgrid(x, y)
22
23 ##### 定义电荷分布 #####
24 epsilon0 = 1
25 rho = 2*np.pi**2*np.sin(np.pi*xx)*np.sin(np.pi*yy)
26
27 ##### 多重网格-五点差分-共轭梯度-求解泊松方程 #####
28 t = time.time()
29
30
31 def smoothing(phi, f, d, pre=False):
32     """
33         平滑算子，共轭梯度迭代，返回迭代结果和剩余误差
34     """
35     size = np.array(f.shape, dtype='int')
36
37     def coefficientMatrix():
38         """
39             展平到一维空间，根据五点差分法，使用稀疏矩阵，初始化系数矩阵
40         """
41         #### 主对角线 ####
42         C = np.ones(sizeA)*-2*(1/d[1]**2+1/d[0]**2)
43         #### x方向差分 ####
44         Dx = np.ones(sizeA-1)/d[1]**2
45         Dx[np.arange(1, size[0], 1, dtype='int')*size[1]-1] = 0
46         #### y方向差分 ####
47         Dy = np.ones(sizeA-size[1])/d[0]**2
48         A = sparse.diags([C, Dx, Dx, Dy, Dy], [0, 1, -1, size[1], -size[1]])
49         return A
50
51     def leftBoundary(phi):
52         """
53             x=xMin, 左侧一类边值条件
54         """
55         b[:size[1]] = phi
56         A.data[0, :size[1]] = 1
57         A.data[[1, 2], :size[1]] = 0
58         A.data[3, :size[1]*2] = 0
59         pass
60
61     def rightBoundary(phi):
62         """
63             x=xMax, 右侧一类边值条件
64         """
65         b[-size[1]:] = phi
66         A.data[0, -size[1]:] = 1
67         A.data[[1, 2], -size[1]:] = 0
68         A.data[4, -size[1]*2:] = 0
69         pass
70
```

```
71     def bottomBoundary(phi):
72         """
73             y=yMin, 下侧一类边值条件
74         """
75             b[np.arange(size[0])*size[1]] = phi
76             A.data[0, np.arange(size[0])*size[1]] = 1
77             A.data[1, np.arange(size[0])*size[1]+1] = 0
78             A.data[3, np.arange(size[0])*size[1]] = 0
79             A.data[4, np.arange(size[0])*size[1]] = 0
80             pass
81
82     def topBoundary(phi):
83         """
84             y=yMax, 上侧一类边值条件
85         """
86             b[np.arange(size[0])*size[1]-1] = phi
87             A.data[0, np.arange(size[0])*size[1]-1] = 1
88             A.data[2, np.arange(size[0])*size[1]-2] = 0
89             A.data[3, np.arange(size[0])*size[1]-1] = 0
90             A.data[4, np.arange(size[0])*size[1]-1] = 0
91             pass
92
93     def conjGradMethod(A, b, x):
94         """
95             共轭梯度迭代求解
96         """
97             global error
98             r = b - sparse.dia_matrix.dot(A, x)
99             d = r
100            CGMstep = 0
101            while CGMstep < size[0]*size[1]:
102                Ad = sparse.dia_matrix.dot(A, d)
103                alpha = np.dot(r.T, r) / np.dot(d.T, Ad)
104                x = x + alpha*d
105                beta = np.dot((r - alpha*Ad).T, r - alpha*Ad) / np.dot(r.T, r)
106                r = r - alpha*Ad
107                d = r + beta*d
108                CGMstep += 1
109                Error = np.linalg.norm(r, ord=np.inf) # 后向误差
110                print('VMGstep=%d,CGMstep=%d,log(r)=%6f' %
111                     (VMGstep, CGMstep, np.log10(Error)))
112                error = np.append(error, Error) # 存储余项误差数据
113                if pre and CGMstep >= preIterTimes: # 预迭代次数上限
114                    break
115                if Error < postIterError: # 误差上限
116                    break
117            return x
118
119 ###### 组装求解矩阵 #####
120 sizeA = size[0]*size[1]
121 A = coefficientMatrix()
122 b = np.reshape(f, (sizeA, 1))
123
124 ##### 定义边界条件 #####
125 leftBoundary(0)
126 rightBoundary(0)
```

```

127     bottomBoundary(0)
128     topBoundary(0)
129
130     ##### 解线性方程组 #####
131     phi = np.reshape(phi, (sizeA, 1)) # 展平到一维空间
132     phi = conjGradMethod(A, b, phi)
133
134     ##### 计算剩余误差 #####
135     res = b-sparse.dia_matrix.dot(A, phi)
136     phi = np.reshape(phi, (size[0], size[1])) # 剩余误差重整回二维空间
137     res = np.reshape(res, (size[0], size[1])) # 求解结果重整回二维空间
138
139     return phi, res
140
141
142 def restriction(res):
143     """
144         限制算子，将网格间距为h的剩余误差投影到间距为2h的网格
145     """
146     xi = np.arange(0, res.shape[1]+1, 2, dtype='int')
147     yi = np.arange(0, res.shape[0]+1, 2, dtype='int')
148     xic = xi[1:-1]
149     yic = yi[1:-1]
150     xii, yii = np.meshgrid(xi, yi)
151     xiic = xii[1:-1, 1:-1]
152     yiic = yii[1:-1, 1:-1]
153     rhs = np.zeros(((res.shape[0]-1)//2+1, (res.shape[1]-1)//2+1))
154     rhs[1:-1, 1:-1] = (4*res[xic, yic]+2*(res[xic-1, yic]+res[xic+1, yic]+res[xic, yic-1]+
155             res[xic, yic+1]) +
156             res[xic-1, yic-1]+res[xic-1, yic+1] + res[xic+1, yic-1]+res[xic+1,
157             yic-1])/16 # 非边界
158     rhs[1:-1, 0] = (2*res[yic, 0]+res[yic-1, 0]+res[yic+1, 0])/4 # 左边界
159     rhs[1:-1, -1] = (2*res[yic, -1]+res[yic-1, -1]+res[yic+1, -1])/4 # 右边界
160     rhs[0, 1:-1] = (2*res[0, xic]+res[0, xic-1]+res[0, xic+1])/4 # 下边界
161     rhs[-1, 1:-1] = (2*res[-1, xic]+res[-1, xic-1]+res[-1, xic+1])/4 # 上边界
162     rhs[[0, 0, -1, -1], [0, -1, 0, -1]] = res[[0, 0, -1, -1], [0, -1, 0, -1]] # 四个角
163     return rhs
164
165
166 def prolongation(eps):
167     """
168         延拓算子，将网格间距为2h的低精度解插值到间距为h的网格
169     """
170     xi = np.arange(0, eps.shape[1], 1, dtype='int')
171     yi = np.arange(0, eps.shape[0], 1, dtype='int')
172     xic = xi[:-1]
173     yic = yi[:-1]
174     xii, yii = np.meshgrid(xi, yi)
175     xiic = xii[:-1, :-1]
176     yiic = yii[:-1, :-1]
177     phi = np.zeros(((eps.shape[0]-1)*2+1, (eps.shape[1]-1)*2+1))
178     phi[xii*2, yii*2] = eps[xii, yii] # 非插值点
179     phi[xiic*2+1, yiic*2+1] = (eps[xic, yic]+eps[xic+1, yic] +
180             eps[xic+1, yic+1]+eps[xic+1, yic-1])/4 # 中心点
181     phi[xiic*2+1, yiic*2] = (eps[xic, yic]+eps[xic+1, yic])/2 # 左右点
182     phi[yic*2+1, -1] = (eps[yic, -1]+eps[yic+1, -1])/2

```

```
181     phi[xiic*2, yiic*2+1] = (eps[xiic, yiic]+eps[xiic, yiic+1])/2 # 上下点
182     phi[-1, xic*2+1] = (eps[-1, xic]+eps[-1, xic+1])/2
183     return phi
184
185
186 def VCycleMultiGrid(phi, f, d):
187     """
188     V循环多重网格法主程序
189     """
190     global VMGstep
191     phi, res = smoothing(phi, f, d, pre=True) # 预平滑
192     rhs = restriction(res)
193     eps = np.zeros_like(rhs)
194     VMGstep += 1
195     if VMGstep < mainIterTimes:
196         VCycleMultiGrid(eps, rhs, 2*d)
197     else:
198         eps = smoothing(eps, rhs, 2*d, pre=True)[0] # 到达最大主迭代次数
199     VMGstep -= 1
200     phi += prolongation(eps)
201     phi = smoothing(phi, f, d)[0] # 后平滑
202     return phi
203
204
205 VMGstep = 1
206 phi = np.zeros((len(y), len(x))) # 等号左侧自变量的预测
207 f = -rho/epsilon0 # 等号右侧的函数表达式
208 phi = VCycleMultiGrid(phi, f, d)
209
210 ##### 三点差分-计算梯度 #####
211 Ex = (phi[1:-1, 2:]-phi[1:-1, :-2])/2/d[1]
212 Ey = (phi[2:, 1:-1]-phi[:-2, 1:-1])/2/d[0]
213 Exy = -np.stack((Ey, Ex), axis=2)
214 print('迭代耗时: %6fs' % (time.time()-t))
215
216 ##### 计算误差分析 #####
217 phi0 = np.sin(np.pi*xx)*np.sin(np.pi*yy)/epsilon0 # 该系统的解析解
218 rms = np.linalg.norm(np.reshape(
219     phi-phi0, (phi.size, 1)), ord=2)/np.sqrt(phi.size) # 均方根误差
220 print('均方误差: %e' % rms)
221 dataframe = pd.DataFrame(data={'error': error}) # 导出迭代余项误差
222 dataframe.to_csv("error.csv", index=False, mode='w', sep=',')
223
224 ##### 电势电场求解误差的可视化 #####
225
226
227 def plotElectricPotential():
228     """
229     绘制电势大小
230     """
231     plt.pcolor(x, y, phi, cmap='jet')
232     plt.colorbar()
233     plt.xlabel('X')
234     plt.ylabel('Y')
235     plt.title('Potential distribution')
236     pass
```

```
237
238
239     def plotEquipotentialSurface():
240         """
241             绘制等势线
242         """
243         plt.contour(x, y, phi, 10, colors='k',
244                     linestyles='dashed', linewidths=0.5)
245         pass
246
247
248     def plotElectricFieldIntensity():
249         """
250             绘制电场强度大小
251         """
252         E = np.linalg.norm(Exy, ord=2, axis=2)
253         E[200-1, 300-1:701-1] = np.NaN
254         E[500-1:801-1, 300-1] = np.NaN
255         E[500-1:801-1, 700-1] = np.NaN
256         plt.pcolor(x[1:-1], y[1:-1], E, cmap='jet')
257         plt.colorbar()
258         plt.xlabel('X')
259         plt.ylabel('Y')
260         plt.title('Field distribution')
261         pass
262
263
264     def plotElectricFieldDirection():
265         """
266             绘制电场强度方向
267         """
268         xi = np.arange(0, 20, 1, dtype='int') * (len(x)-1)//20+(len(x)-1)//40
269         yi = np.arange(0, 20, 1, dtype='int') * (len(y)-1)//20+(len(y)-1)//40
270         xii, yii = np.meshgrid(xi, yi)
271         xa = x[xi]
272         ya = y[yi]
273         E = np.linalg.norm(Exy[yii-1, xii-1, :], ord=2, axis=2)
274         dx = Exy[yii-1, xii-1, 1]/E
275         dy = Exy[yii-1, xii-1, 0]/E
276         plt.quiver(xa, ya, dx, dy)
277         pass
278
279
280     plt.figure(figsize=(10, 8))
281     plt.subplot(2, 2, 1)
282     plt.axis('equal')
283     plt.axis([-1, 1, -1, 1])
284     plotElectricPotential()
285     plotEquipotentialSurface()
286     plt.subplot(2, 2, 2)
287     plt.axis('equal')
288     plt.axis([-1, 1, -1, 1])
289     plotElectricFieldIntensity()
290     plotElectricFieldDirection()
291     plt.subplot(2, 1, 2) # 绘制误差下降曲线
292     plt.yscale('log')
```

```
293     plt.plot(error)
294     plt.xlabel('Iterating times')
295     plt.ylabel('Residual errors')
296     plt.title('Residual errors')
297     plt.show()
```
