计算物理大作业-PDE 有限差分法的稳定性分析

2020 年 12 月 19 日

作者: 吴远清
学号: 2018300001031
邮箱: yuanqingwu@whu.edu.cn
学院: 武汉大学, 弘毅学堂
电话: 17352918073

摘要

有限差分法是对 PDE 进行数值求解的一种重要方法, 也是本课程中主要学习的方法. 但是, 本课程对于有限差分法的数值稳定性没有作较为详细而严谨的介绍. 本文以一维扩散方程为例, 介绍了对有限差分法稳定性分析常用的 Von Neumann 分析法, 分别对于显示差分和隐式差分, 求解了参数与数值稳定性间的关系, 并做了验证.

关键词: 一维扩散方程, 数值方法,lax 等价定理,Von Neumann 稳定性分析

1 研究背景

由于偏微分方程解析求解面临着极大的困难,PDE 的数值解法就显得尤为重要. 其中, 有限差分法是最简单, 应用也最广泛的方法. 本课程也主要教授有限差分法相关的内容. 教材中指出, 有些算法的数值稳定性存在问题 [2]. 但并未给出详细的讨论和证明. 事实上, 对于不同的算法, 其数值稳定性可能依赖于离散化网格的参数, 也可能对于几乎所有参数都非常稳定或者非常不稳定 [5]. 这种差异和对参数的依赖使得对有限差分法的稳定性的研究具有非常重要的意义. 我接下来将介绍一种数学上严格的稳定性分析方法, 并使用对同一问题的两种离散方式来说明算法稳定性对数值计算的重要性.

2 理论基础

首先, 我需要指出, 本文中介绍的分析方法只适用于部分偏微分方程. 更准确的说, 是满足 **Lax–Richtmyer 条件**的偏微分方程.Lax 和 Richtmyer 证明了如下结论 [3]:

对于一个良好的线性初始值问题的一致的有限差分法，当且仅当它是稳定的时候，该方法是收敛的.

这个定理表明, 对于一些 PDE 系统, 系统的稳定性与收敛性是等价的.Lax-Richtmyer 条件为,PDE 是线性的, 且与差分格式相容. 事实上, 对于我们课程内求解的绝大多数体系,Lax-Richtmyer 条件均满足. 在此条件的基础上,Von Neumann 提出了一套可以分析差分格式稳定性的方法 [1]. 大致流程如下:

(1) 将数值解与真实解之差 ϵ 做傅里叶变换.

(2) 计算每次迭代的误差涨幅 $G = \epsilon^{n+1}/\epsilon^n$.

(3) 若 $|G| \leq 1$, 则该差分格式是稳定的, 否则, 该差分格式是不稳定的.

这就是所谓的 Von Neumann 稳定性分析 (也称傅里叶稳定性分析). 下面, 我将以经典的一维扩散方程为例, 利用 Von Neumann 稳定性分析来计算两种不同的差分格式的稳定性差别.

3 稳定性分析

我在此研究一维扩散问题, 方程可写为 [4]:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial t^2} \tag{3.0.1}$$

其中 α 为一常数 ($\alpha > 0$).

3.1 显式格式差分

首先, 最自然的, 我们使用显示格式差分:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \alpha \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} \tag{3.1.1}$$

重排, 得到:

$$u_j^{n+1} = u_j^n + \frac{\alpha \Delta t}{(\Delta x)^2}(u_{j+1}^n - 2 * u_j^n + u_{j-1}^n) \tag{3.1.2}$$

该差分格式的数值解 N_j^n 和体系的真实解 T_j^n 均满足方程 (3.1.2), 并且由于其是线性方程, 误差 $\epsilon_j^n = N_j^n - T_j^n$ 也满足此方程, 代入得:

$$\epsilon_j^{n+1} = \epsilon_j^n + \frac{\alpha \Delta t}{(\Delta x)^2}(\epsilon_{j+1}^n - 2\epsilon_j^n + \epsilon_{j-1}^n) \tag{3.1.3}$$

对误差 ϵ 做傅里叶展开, 有:

$$\epsilon(x,t) = \sum_{m=1}^{N_x} e^{at} e^{ik_m x} \tag{3.1.4}$$

其中,a 为常数,$k_m = \frac{\pi m}{L}$,N_x 为差分格式的 x 网格数. 并且, 再次由线性性质可知:

$$G = \frac{\epsilon(x, t + \Delta t)}{\epsilon(x,t)} = \frac{e^{a(t+\Delta t)e^{ik_m x}}}{e^{at}e^{ik_m x}} = e^{a\Delta t} \tag{3.1.5}$$

回到 (3.1.3):

$$e^{a\Delta t} = 1 + \frac{\alpha \Delta t}{\Delta x^2}\left(e^{ik_m \Delta x} + e^{-ik_m \Delta x} - 2\right) \tag{3.1.6}$$

由 Euler 公式:

$$\sin^2 \frac{k_m \Delta x}{2} = -\frac{1}{4}\left(e^{ik_m \Delta x} + e^{-ik_m \Delta x} - 2\right) \tag{3.1.7}$$

由此有:

$$G = 1 - \frac{4\alpha \Delta t}{\Delta x^2} \sin^2 \left(k_m \Delta x/2\right) \tag{3.1.8}$$

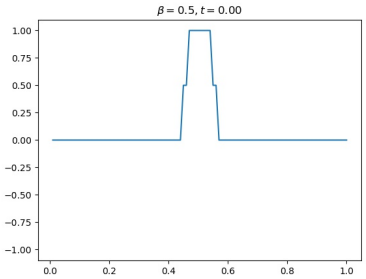
若要差分格式稳定, 即 $|G| \leq 1$:

$$\frac{\alpha \Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{3.1.9}$$

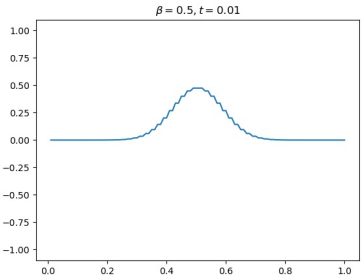
至此, 我们通过 Von Neumann 方法, 成功的讨论了一维扩散方程的显式格式差分的稳定性条件. 可见, 随着体系网格的缩小, 时间步长也需要减小, 以维持计算稳定, 这符合我们的预期.

接下来, 我们进行实际计算来测试方程 (3.1.9) 的结论是否正确, 我们定义 $\alpha \Delta t/(\Delta x)^2$ 为 β, 调节不同的 Δt 和 Δx 的值, 来验证体系的稳定性, 代码见附录 (代码 1).

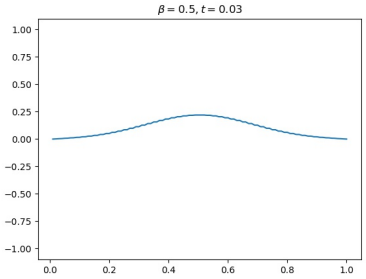
首先, 我们计算了两组不同的 Δx 和 Δt 下的一维扩散方程, 但均保持 $\beta = 0.5$, 结果如下图 (见图 1, 图 2), 可见, 体系均能保持稳定.



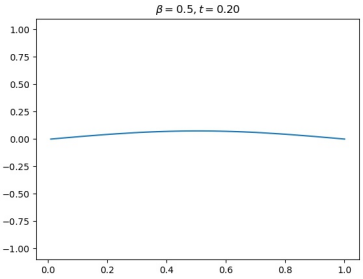
(a) t=0



(b) t=0.01

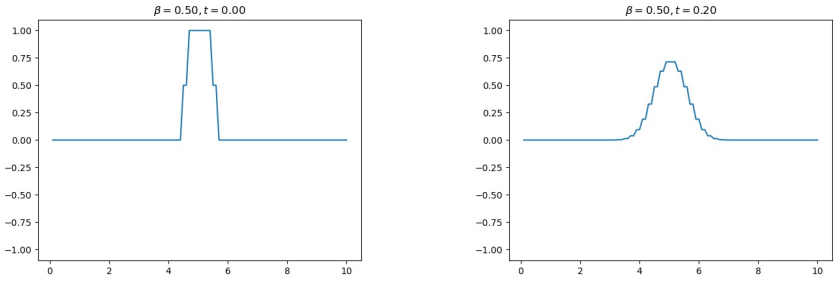


(c) t=0.03

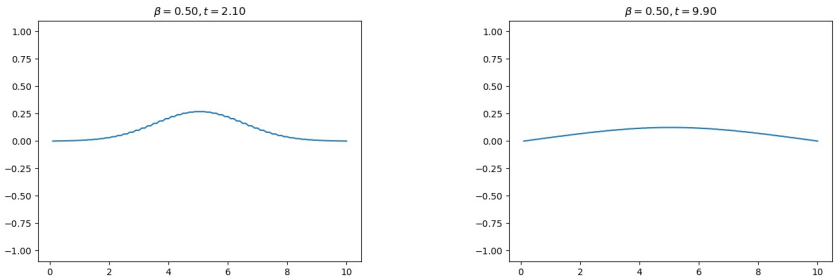


(d) t=0.20

图 1: 显式差分,$\Delta x = 0.01, \Delta t = 0.0001, \beta = 0.5$



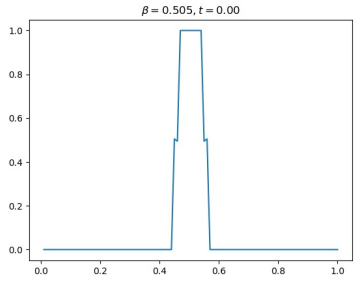
(a) t=0 (b) t=0.20



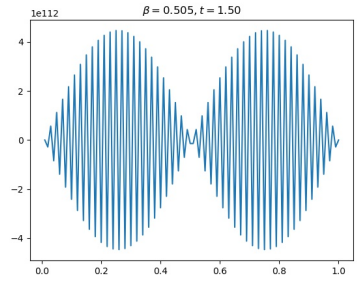
(c) t=2.10 (d) t=9.90

图 2: 显式差分,$\Delta x = 0.1, \Delta t = 0.01, \beta = 0.5$

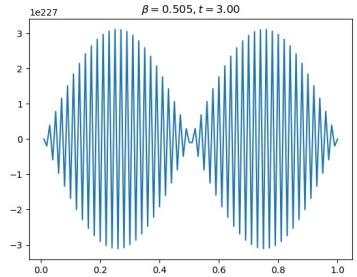
接下来, 我们稍微改变一点点 Δt 值, 使得 β 值略大于 0.5, 可以发现, 此时, 差分格式立刻变得不稳定了 (见图 3, 图 4). 扩散行为表现出奇异现象, 并且数值急剧增大, 并最终溢出.



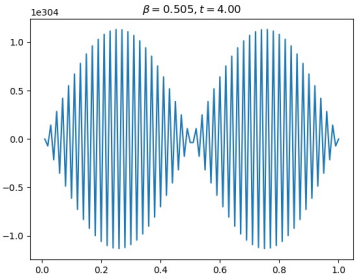
(a) t=0



(b) t=0.20

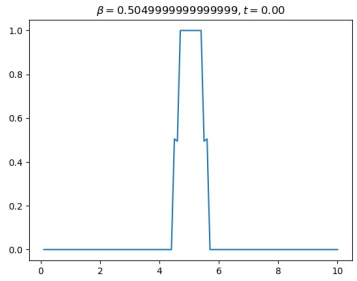


(c) t=1.50

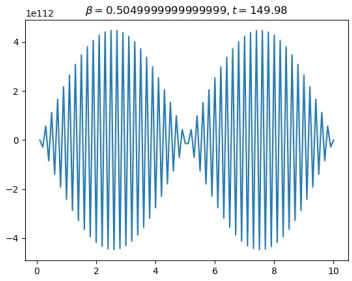


(d) t=4.00

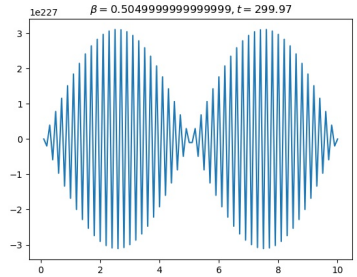
图 3: 显式差分,$\Delta x = 0.01, \Delta t = 0.000101, \beta = 0.505$



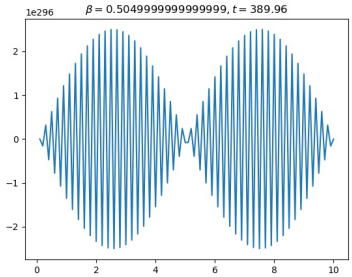
(a) t=0



(b) t=150



(c) t=300



(d) t=390

图 4: 显式差分,$\Delta x = 0.01, \Delta t = 0.000101, \beta = 0.505$

3.2 隐式格式差分

下面我来介绍隐式格式差分. 差分方程如下:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \alpha \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{(\Delta x)^2} \tag{3.2.1}$$

转化可得:

$$u_j^{n+1} - \frac{\alpha \Delta t}{(\Delta x)^2}(u_{j+1}^{n+1} + u_{j-1}^{n+1} - 2u_j^{n+1}) = u_j^n \tag{3.2.2}$$

表面上看起来, 左侧包含多个 $n+1$ 时刻的值, 难以求解, 实际上, 将左侧写成矩阵的形式:

$$A\mathbf{u}^{n+1} = \mathbf{u}^n \qquad A = 1 - \frac{\alpha \Delta t}{(\Delta x)^2} M \tag{3.2.3}$$

其中 M 为:

$$M = \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -2 \end{pmatrix}$$

由此, 我们可以看出, 只要求的矩阵 A 的逆, 就能由 $\mathbf{u}^n$ 求得 $\mathbf{u}^{n+1}$:

$$\mathbf{u}^{n+1} = A^{-1}\mathbf{u}^n \tag{3.2.4}$$

讨论完隐式格式差分的求解, 我们来看看隐式格式差分的稳定性. 明显的, 差分格式的误差迭代系数 G 具有和 (3.1.5) 一样的形式:

$$G = e^{a\Delta t} \tag{3.2.5}$$

同样的, 对差分方程的分析将给出类似于 (3.1.6) 的结果, 除了方程右侧是 $n+1$ 时刻的迭代值, 因此含有一个 $e^{a\Delta t}$ 因子:

$$e^{a\Delta t} = 1 + \frac{\alpha\Delta t}{(\Delta x)^2} e^{a\Delta t}\left(e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2\right) \tag{3.2.6}$$

重排:

$$e^{a\Delta t} = \left(1 - \frac{\alpha\Delta t}{(\Delta x)^2}\left(e^{ik_m\Delta x} + e^{-ik_m\Delta x} - 2\right)\right)^{-1} \tag{3.2.7}$$

差分格式的稳定性要求:

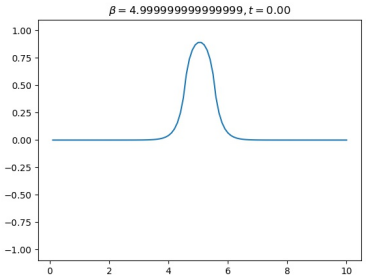
$$|G| = |e^{a\Delta t}| \leq 1 \tag{3.2.8}$$

同时代入 Euler 方程, 得:

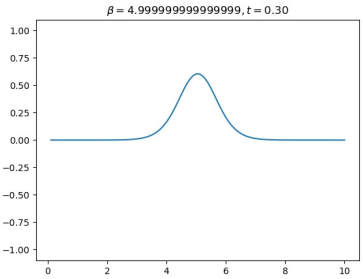
$$1 + \frac{4\alpha\Delta t}{\Delta x^2}\sin^2\left(k_m\Delta x/2\right) \geq 1 \tag{3.2.9}$$

明显的, 由于 $\alpha > 0, \sin^2\left(k_m\Delta x/2\right) > 0$,(3.2.9) 对于任何 $\Delta t, \Delta x$ 的组合均成立, 即意味着, 隐式格式差分的稳定性不依赖于网格参数的选择.

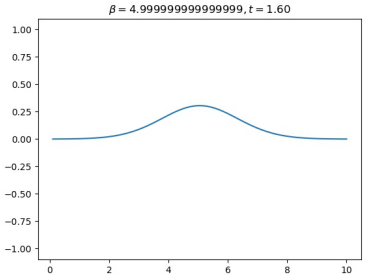
我实现了隐式格式差分计算 (代码见附录代码 2), 并计算了两组不同的 $\Delta x - \Delta t$ 值的组合下体系的行为. 结果如图 5, 图 6 所示. 可以看出, 即使体系的 β 远大于显示格式差分的稳定阈值 (我们最大测试了 $beta = 50$ 的情况, 此时时间步长为 $\Delta t = 1s$), 体系仍然能够保持稳定. 这也验证了我的预测.



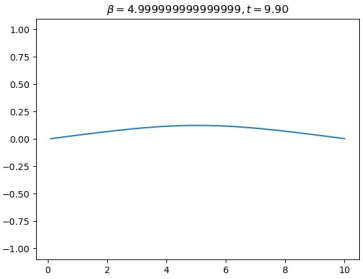
(a) t=0



(b) t=0.3

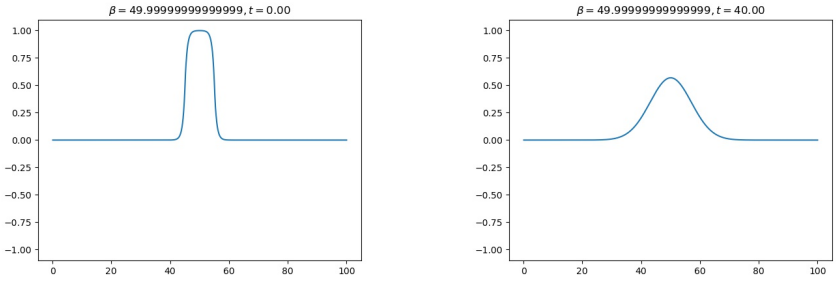


(c) t=1.6



(d) t=9.9

图 5: 隐式差分,$\Delta x = 0.1, \Delta t = 0.1, \beta = 5$



(a) t=0 (b) t=40

(c) t=150 (d) t=990

图 6: 隐式差分,$\Delta x = 0.1, \Delta t = 1, \beta = 50$

4 总结

本文中, 我介绍了分析 PDE 的有限差分法的稳定性的方法, 并以一维扩散方程为例, 分别计算了显式和隐式差分方程的稳定性条件. 结果表明, 对于显式方程, 随着空间网格的减小, 时间步长也必须随着减小, 二者之间关系必须满足 (3.1.9) 式, 否则计算误差将会发散. 而对于隐式方程, 则是无条件稳定的. 我也使用代码分别计算了几种情况下算法的稳定性, 结果与理论预测一致.

5 附录

Listing 1: 显式形式求解

```
1 import numpy as np
```

```
import matplotlib.pyplot as plt
from tqdm import tqdm

class heat():
    def __init__(self, k, dx, dt, nx, totT, x1, x2):
        self.dx = dx
        self.dt = dt
        self.nx = nx
        self.nt =  int(totT/dt)
        self.u = np.zeros(nx)
        self.u[x1:x2] = 1
        self.alpha = k*dt/(dx**2)
        self.x = np.linspace(1,self.nx,self.nx)*self.dx
    def update(self):
        self.u[1:-1] += self.alpha*(self.u[2:] + self.u[:-2] - 2*self.u
            [1:-1])
    def calculate(self, savefig = False):
        img_num = 0
        for i in tqdm( range(self.nt)):
            self.update()
            if savefig and (i% int(self.nt/100)==0):
                plt.clf()
                #plt.ylim(-1.1,1.1)
                plt.title("$\\beta={},t={:.2f}$".
                    format(self.alpha,self.dt*i))
                plt.plot(self.x, self.u)
                plt.savefig("./img/{}.jpg". format(img_num))
                img_num += 1
    def snapshot(self):
        plt.ylim(-1.1,1.1)
        plt.plot(self.x, self.u)
        plt.show()

h = heat(0.5,0.1,0.0101,100,1000,45,55)
h.calculate(True)
```

Listing 2: 隐式形式求解

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

```
3  from tqdm import tqdm
4
5  class heat():
6      def __init__(self, k, dx, dt, nx, totT, x1, x2):
7          self.dx = dx
8          self.dt = dt
9          self.nx = nx
10         self.nt =  int(totT/dt)
11         self.u = np.zeros(nx)
12         self.u[x1:x2] = 1
13         self.alpha = k*dt/(dx**2)
14         print(self.alpha)
15         self.x = np.linspace(1,self.nx,self.nx)*self.dx
16         self.M = np.eye(nx)*-2
17         for i in  range(nx):
18             if i != 0:
19                 self.M[i][i-1] = 1
20             if i != (nx-1):
21                 self.M[i][i+1] = 1
22         self.M = np.linalg.inv(np.eye(self.nx)-self.alpha*self.M)
23     def update(self):
24         self.u = np.dot(self.M, self.u)
25         #print(self.u)
26     def calculate(self, savefig = False):
27         img_num = 0
28         for i in tqdm( range(self.nt)):
29             self.update()
30             if savefig and (i% int(self.nt/100)==0):
31                 plt.clf()
32                 plt.ylim(-1.1,1.1)
33                 plt.title("$\\beta={},t={:.2f}$".
                       format(self.alpha,self.dt*i))
34                 plt.plot(self.x, self.u)
35                 plt.savefig("./img/{}.jpg". format(img_num))
36                 img_num += 1
37     def snapshot(self):
38         plt.ylim(-1.1,1.1)
39         plt.plot(self.x, self.u)
40         plt.show()
```

```
41
42
43 h = heat(0.5,0.1,1,1000,1000,450,550)
44 h.calculate(True)
```

参考文献

[1] J. G. Charney, R. Fjörtoft, and J. V. Neumann. Numerical Integration of the Barotropic Vorticity Equation, Nov. 1950. ISSN: 2153-3490 Issue: 4 Pages: 237-254 Publisher: John Wiley & Sons, Ltd Volume: 2.

[2] N. Giordano and H. Nakanishi. *Computational Physics*. Pearson/Prentice Hall, 2006.

[3] P. D. Lax and R. D. Richtmyer. Survey of the stability of linear finite difference equations, May 1956. ISSN: 1097-0312 Issue: 2 Pages: 267-293 Publisher: John Wiley & Sons, Ltd Volume: 9.

[4] P. Scherer. *Computational Physics: Simulation of Classical and Quantum Systems*. Springer Science & Business Media, July 2013.

[5] J. Thijssen. *Computational Physics*. Cambridge University Press, 2007.