

微分方程数值求解——有限差分法



科研民工
科研新手村玩家-Lv0

1. 引言

有限差分法（Finite Difference Method, FDM）是一种求解微分方程数值解的近似方法，其主要原理是对微分方程中的微分项进行直接差分近似，从而将微分方程转化为代数方程组求解。有限差分法的原理简单，粗暴有效，最早由远古数学大神欧拉（L. Euler 1707-1783）提出，他在1768年给出了一维问题的差分格式。1908年，龙格（C. Runge 1856-1927）将差分法扩展到了二维问题【对，就是龙格-库塔法中的那个龙格】。但是在那个年代，将微分方程的求解转化为大量代数方程组的求解无疑是将一个难题转化为另一个难题，因此并未得到大量的应用。随着计算机技术的发展，快速准确地求解庞大的代数方程组成为可能，因此逐渐得到大量的应用。发展至今，有限差分法已成为一个重要的数值求解方法，在工程领域有着广泛的应用背景。本文将从有限差分法的原理、基本差分公式、误差估计等方面进行概述，给出其基本的应用方法，对于一些深入的问题不做讨论。

2. 有限差分方法概述

首先，有限差分法是一种求解微分方程的数值方法，其面对的对象是微分方程，包括常微分方程和偏微分方程。此外，有限差分法需要对微分进行近似，这里的近似采取的是离散近似，使用某一点周围点的函数值近似表示该点的微分。下面将对该方法进行概述。

2.1. 有限差分法的基本原理

这里我们使用一个简单的例子来简述有限差分法的基本原理，考虑如下常微分方程

$$\begin{cases} u'(x) + c(x)u(x) = f(x), & x \in [a, b]; \\ u(x=a) = d \end{cases} \quad (1)$$

微分方程与代数方程最大的不同就是其包含微分项，这也是求解微分方程最难处理的地方。有限差分法的基本原理即使用近似方法处理微分方程中的微分项。为了得到微分的近似，我们最容易想到的即导数定义

$$u'(x) = \lim_{\Delta x \rightarrow 0} \frac{u(x + \Delta x) - u(x)}{\Delta x} \approx \frac{u(x + \Delta x) - u(x)}{\Delta x} \quad (2)$$

上式后面的近似表示使用割线斜率近似替代切线斜率， Δx 即为步长，如图 1(a)所示。式(2)表明函数在某一点的微分可以由相邻点的函数值近似确定。显然，这里微分近似的精度与步长的选取有关，步长越短则越精确。

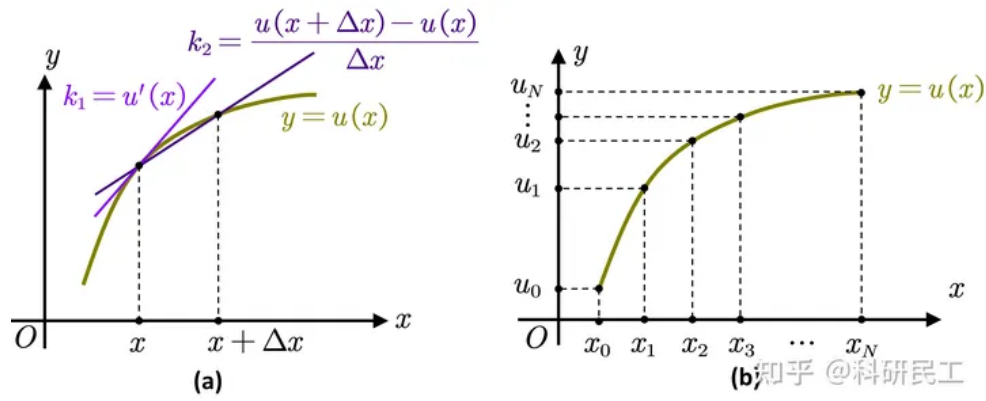


图 1. (a) 微分的近似表示； (b) 一维区间的离散表示

因此，这里首先需要对求解域进行离散，然后分别得到各离散点上的微分近似。对于示例的一维问题，将求解区间等分为 N 个区间，步长为 h ，分别将包含首尾的各结点记为 $x_0, x_1, x_2, \dots, x_N$ ，对应的函数值为 $u_0, u_1, u_2, \dots, u_N$ ，对应各点的一阶微分记为 $u'_0, u'_1, u'_2, \dots, u'_N$ ，如图 1(b) 所示。这样，就把原问题的求解转化为了各结点函数值 u_i 的求解。式(2)的离散形式表示为

$$u'_i = \frac{1}{h}(u_{i+1} - u_i), \quad i = 0, 1, 2, \dots, N-1 \quad (3)$$

将式 (3) 代入方程 (1) 可以得到

$$\begin{cases} (u_{i+1} - u_i)/h + c(x_i)u_i = f(x_i); & i = 1, 2, \dots, N-1 \\ u_0 = d \end{cases} \quad (4)$$

这里的结点坐标 $x_i = a + ih$, ($i = 0, 1, 2, 3, 4$)，步长 $h = (b - a)/N$ 均为已知。记 $c_i = c(x_i)$, $f_i = f(x_i)$ ，将式(4)合并同类项可以得到如下递推关系

$$u_{i+1} = (1 - hc_i)u_i + hf_i; \quad u_0 = d, \quad i = 0, 1, 2, 3, \dots, N-1 \quad (5)$$

上式共有 N 个方程 ($i = 0, 1, 2, \dots, N-1$)，包括 N 个未知数 (u_1, u_2, \dots, u_N)，刚好可以求解得到各结点上的待求函数的值，从而得到原问题在求解域上的近似解。由于该问题中初值已给定，按照各结点依次迭代就可以得到该问题的解。此外，为了给出更一般化的求解方法，可以将(5)写成矩阵的形式，即

$$\mathbf{A}\mathbf{u} = \mathbf{F} \quad (6)$$

即

$$\begin{bmatrix} 1 & & & \\ C_1 & 1 & & \\ & \ddots & \ddots & \\ & & C_{N-1} & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_{N-1} \end{bmatrix} - \begin{bmatrix} C_0 u_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (7)$$

其中 $\mathbf{u} = \{u_1, \dots, u_{N-1}, u_N\}^T$ 总共包含 N 个待求结点值。 $C_i = hc_i - 1$, $F_i = hf_i$ 。上述方程组的系数矩阵 \mathbf{A} 显然可逆，则上述问题的解总是可以表示为 $\mathbf{u} = \mathbf{A}^{-1}\mathbf{F}$ 。为了验证上述结果的正确性，我们取 $c(x) = 1$, $f(x) = \sin(x) + \cos(x)$ ，求解区间为 $x \in [0, 2\pi]$ ，且满足边界条件 $u(0) = 0$ ，则原问题(1)可以写为如下形式

$$\begin{cases} u'(x) + u(x) = \sin(x) + \cos(x), & x \in [0, 2\pi]; \\ u(x = 0) = 0 \end{cases} \quad (8)$$

则该问题对应方程组(6)中的 $C_i = h - 1$, $F_i = h[\sin(x_i) + \cos(x_i)]$, $d = 0$ 。将上述表达式代入式(7)并利用 $\mathbf{u} = \mathbf{A}^{-1}\mathbf{F}$ 即可以得到该问题的近似解。此外，上述方程的解析解可以容易

给出为 $u(x) = \sin(x)$ 。分别取 $N = 5, 10, 20, 100$ 时有限差分法结果与解析解对比如图(2)所示，可以看到网格划分越精细则求解结果越精确，与解析解的误差越小。当 $N = 100$ 时，两者的误差已经极小，表明有限差分法是有效的。

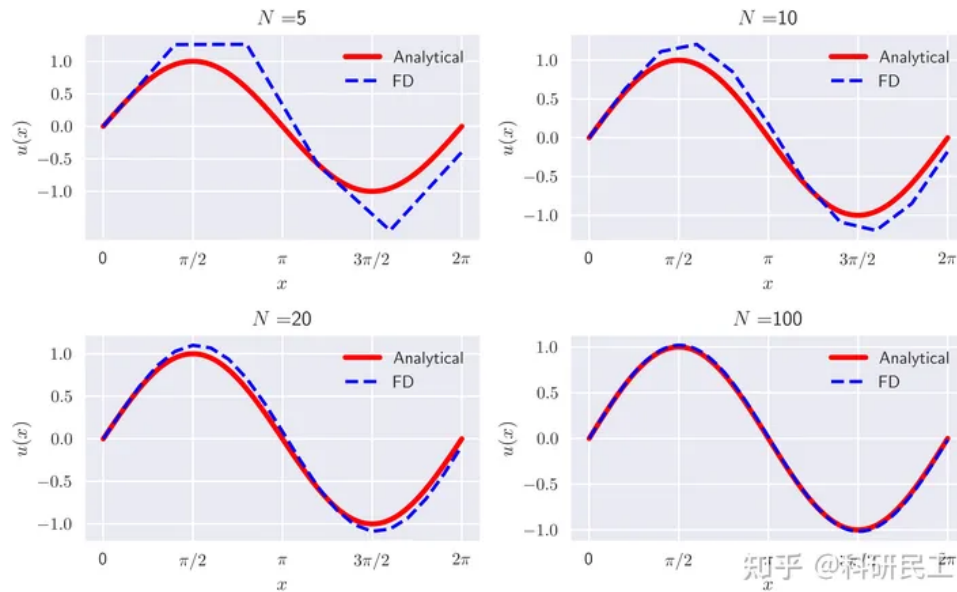


图2. 有限差分法与解析解的结果对比

求解该问题时的Python计算代码如下：

```
from numpy import *

h = 2*pi/N
x = linspace(0, 2*pi, N+1)
A = zeros((N, N))
F = zeros((N, 1))
u = zeros((N+1, 1))
for i in range(N):
    A[i, i] = 1
    if i > 0: A[i, i-1] = h-1
    F[i] = h*(sin(x[i]) + cos(x[i]))
u[1:] = dot(linalg.inv(A), F)
```

2.2. 微分的近似表示

上一节以一个一维一阶微分方程为例简单描述了有限差分的基本原理，下面我们对其更广泛的使用进行扩展。注意到式 (3) 中的一阶微分是使用相邻结点的函数值来表示，一般地，假设函数 $u(x)$ 在求解域内连续可导，则由泰勒级数我们可以得到

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \cdots + \frac{h^k}{k!}u^{(k)}(x) + \cdots \quad (9)$$

基于泰勒级数并做适当的截断，我们就可以得到各阶微分的近似表达式，或者记做“差分公式”。下面主要对一阶和二阶微分的差分公式进行讨论，更高阶的微分可以同理推导得到。

2.2.1. 一阶微分

我们将(8)式另写作

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u'''(x+\xi) \quad (10)$$

这里 $\xi \in (0, h)$ 。对上式变形即可以得到一阶微分的向前差分公式：

$$u'_F(x) = \frac{u(x+h) - u(x)}{h} - \frac{h}{2}u''(x+\xi) \quad (11)$$

将(10)式中的 h 用 $-h$ 替代，则可以得到一阶微分的向后差分公式：

$$u'_B(x) = \frac{u(x) - u(x-h)}{h} + \frac{h}{2}u''(x+\xi) \quad (12)$$

联立式(11)与(12)，则可以得到一阶微分的中心差分公式：

$$u'_C(x) = \frac{u(x+h) - u(x-h)}{2h} - \frac{h^2}{6}u^{(3)}(x+\xi) \quad (13)$$

式(11)-(13)是几种常用的一阶微分的差分公式，式(11)-(13)中右边的最后一项即为截断误差。由于这里的误差与步长相关，因此可以看到向前和向后差分公式具有一阶近似精度 $[O(h)]$ ，而中心差分则具有二阶近似精度 $[O(h^2)]$ ，三种差分公式的区别如图(3)所示。

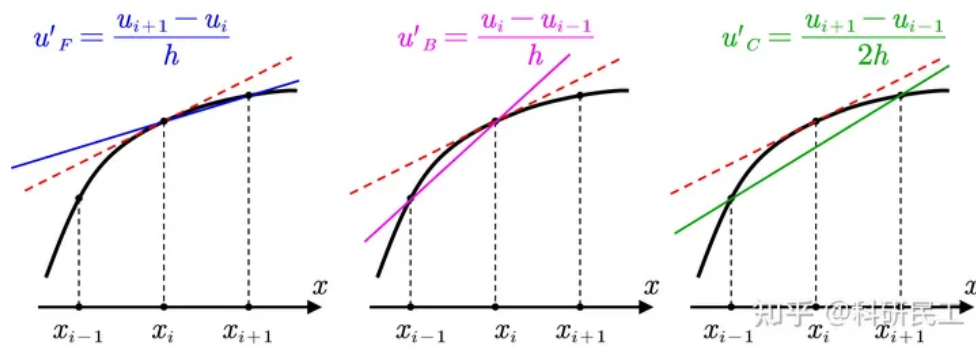


图3. 三种不同差分方法的示意图

2.2.2. 二阶微分

类似地，将式(8)多写几项表示为

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + \frac{h^3}{6}u^{(3)}(x) + \frac{h^4}{12}u^{(4)}(x+\xi) \quad (14)$$

类似的，在式(14)中令 $h \rightarrow -h$ ，并联立原式可以得到二阶微分的中心差分公式

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} - \frac{h^2}{12}u^{(4)}(x+\xi) \quad (15)$$

这里列出的差分公式仅仅只是一些常用的形式，不同的差分公式具有不同的求解误差，会产生不一样的求解效果。对于一般问题，由于中心差分公式的精度较高，因此使用较多。

2.2.3. 差分公式的推导

前面给出了常用一阶和二阶差分公式，而实际问题中经常需要给定差分项的差分公式，这时可以采用待定系数法来给出。例如，对于一阶微分，我们想要在差分公式中同时包含 $u(x)$ ， $u(x+h)$ ， $u(x+2h)$ 三项，即需要同时使用 u_i ， u_{i+1} 和 u_{i+2} 来表示 u'_i ，则令

$$u'(x) = au(x) + bu(x+h) + cu(x+2h) \quad (16)$$

由泰勒公式

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + O(h^3) \quad (17)$$

$$u(x+2h) = u(x) + 2hu'(x) + 2h^2u''(x) + O(h^3)$$

将式(17)代入(16)并对比系数可以得到

$$\begin{cases} a+b+c=0 \\ (b+2c)h=1 \\ (b/2+2c)h^2=0 \end{cases} \Rightarrow \begin{cases} a=-3/(2h) \\ b=2/h \\ c=-1/(2h) \end{cases} \quad (18)$$

即

$$u'(x) = \frac{-3u(x) + 4u(x+h) - u(x+2h)}{2h} \quad (19)$$

这样就得到了指定差分项的差分公式。通过计算可知，上式给出的差分公式具有二阶近似精度，比向前差分与向后差分公式的精度都要高。同样地，其它差分公式也可以通过类似的方法推导得到。通过上述方法我们可以根据实际求解问题的不同给出不同的差分公式以达到最高的求解效率。

2.3. 偏微分方程和时域问题的有限差分法

通常工程中遇到的许多问题都是偏微分方程，即包含两个及以上自变量变化的问题。对于偏微分方程，有限差分法也可以进行类似处理。对于二维、三维或者时域问题，其处理思路与方法类似，这里为了简便，我们以二维问题为例来进行讨论，三维问题和时域问题可以同理进行推广。对于二维偏微分方程，其包含两个自变量的变化，求解域为面域。我们还是以一个实例来演示其基本方法，比如，考虑如下二维泊松方程

$$\begin{cases} \Delta u = f(x, y); & x \in [0, a], y \in [0, b] \\ u(0, y) = c(y), u(a, y) = d(y) \\ u(x, 0) = g(x), u(x, b) = h(x) \end{cases} \quad (20)$$

首先，将求解域进行离散化。由于求解域为矩形，分别将 x 和 y 方向等分为 M 与 N 份，从而构建网格，两个方向的离散坐标分别记为 $x_i (i = 0, 1, 2, \dots, M)$ 和 $y_j (j = 0, 1, 2, \dots, N)$ ，各结点的函数值简记为 $u(x_i, y_j) = u_{i,j}$ ，如图(4)所示。

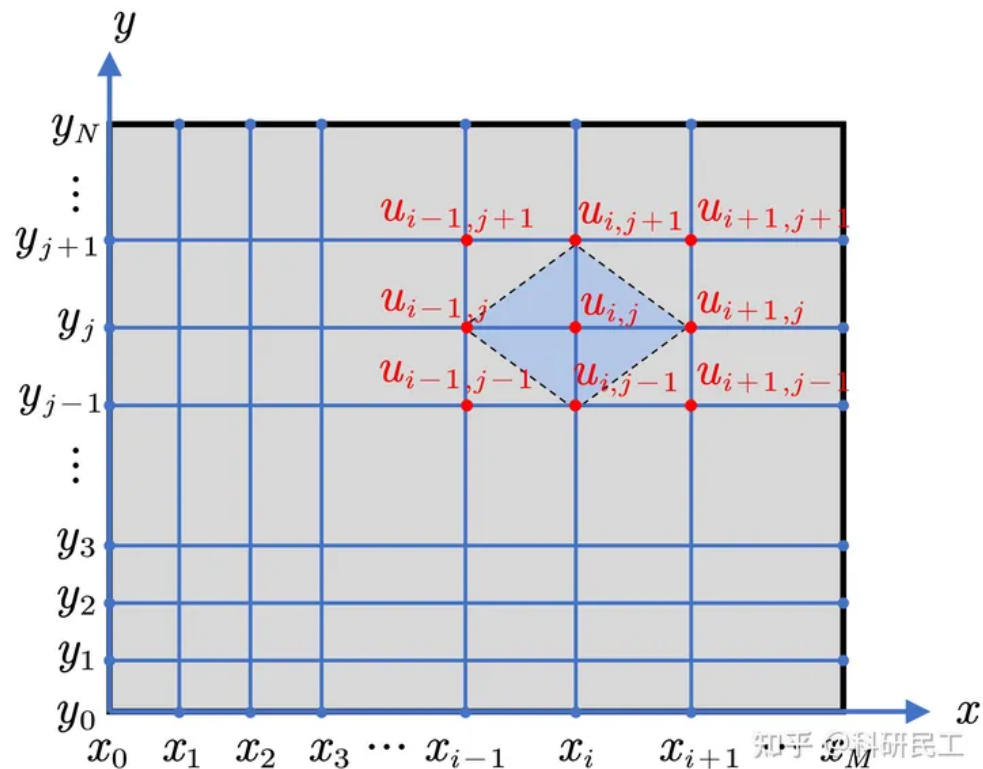


图4. 二维区域的离散网格

进一步，对方程(20)中的二阶微分使用中心差分公式有

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2}, \quad \frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} \quad (21)$$

其中 $h_x = a/M$ 与 $h_y = b/N$ 分别为 x 方向与 y 方向的步长。记 $p = 1/h_x^2$, $q = 1/h_y^2$, $r = -2(p+q)$, $f(x_i, y_j) = f_{i,j}$, 将式(21)代入原方程得到

$$\begin{cases} qu_{i,j-1} + (pu_{i-1,j} + ru_{i,j} + pu_{i+1,j}) + qu_{i,j+1} = f_{i,j} \\ u_{0,j} = c_j, u_{M,j} = d_j \\ u_{i,0} = g_i, u_{i,N} = h_i \end{cases}; \quad \begin{cases} i = 1, 2, \dots, M-1 \\ j = 1, 2, \dots, N-1 \end{cases} \quad (22)$$

递推式(22)表明每一次迭代中有4个结点值是相互独立的，即

$u_{i+1,j}$, $u_{i-1,j}$, $u_{i,j}$, $u_{i,j+1}$, $u_{i,j-1}$, 这5个点组成一个菱形，根据其中任意4个结点值可以算出最后一个结点的值。由此，不断迭代可以求得所有结点的值。根据所划分的网格，除去所求矩形区域的四个顶点（由边界条件给出），图4网格中的所有结点将会被菱形覆盖，根据边界条件可以依次向内求解得到全域的解。类似地，可以将上述差分方程写作矩阵形式

$$\begin{bmatrix} \mathbf{U} & \mathbf{V} & & & \\ \mathbf{V} & \mathbf{U} & \mathbf{V} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{V} & \mathbf{U} & \mathbf{V} \\ & & & \mathbf{V} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \vec{u}_{i,1} \\ \vec{u}_{i,2} \\ \vdots \\ \vec{u}_{i,N-2} \\ \vec{u}_{i,N-1} \end{bmatrix} = \begin{bmatrix} \vec{f}_{i,1} \\ \vec{f}_{i,2} \\ \vdots \\ \vec{f}_{i,N-2} \\ \vec{f}_{i,N-1} \end{bmatrix} - q \begin{bmatrix} \vec{u}_{i,0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} - q \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \vec{u}_{M,0} \end{bmatrix}$$

其中

$$\mathbf{U} = \begin{bmatrix} r & p & & & \\ p & r & p & & \\ & \ddots & \ddots & \ddots & \\ & & p & r & p \\ & & & p & r \end{bmatrix}_{(M-1) \times (M-1)}, \quad \mathbf{V} = \begin{bmatrix} q & & & & \\ & q & & & \\ & & \ddots & & \\ & & & q & \\ & & & & q \end{bmatrix}_{(M-1) \times (M-1)}$$

$$\vec{u}_{i,j} = \{u_{1,j}, u_{2,j}, \dots, u_{M-1,j}\}^T, \quad \vec{f}_{i,j} = \{f_{1,j}, f_{2,j}, \dots, f_{M-1,j}\}^T$$

$$\vec{u}_{i,0} = \{u_{1,0}, u_{2,0}, \dots, u_{M-1,0}\}^T, \quad \vec{u}_{i,N} = \{u_{1,N}, u_{2,N}, \dots, u_{M-1,N}\}^T$$

$$\vec{u}_{0,j} = \{u_{0,j}, 0, \dots, 0\}^T, \quad \vec{u}_{M,j} = \{0, 0, \dots, u_{M,j}\}^T$$

这里，根据边界条件，四个顶点的值均为常数，此外，向量 $\vec{u}_{i,0}$, $\vec{u}_{i,M}$, $\vec{u}_{0,j}$ 和 $\vec{u}_{M,j}$ 均由边界条件给定。可以看出，相比于常微分方程，由于求解结点的增多，偏微分方程的求解复杂度大大增加。同样地，为了验证计算结果，我们取 $f(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y)$, $c(y) = d(y) = g(x) = h(x) = 0$, $(x, y) \in [0, 1] \times [0, 1]$, 使用有限差分方法对其进行求解。同时，其对应的解析解为 $u(x, y) = \sin(\pi x) \sin(\pi y)$, 在求解网格为 100×100 时两者的结果对比如图5所示。

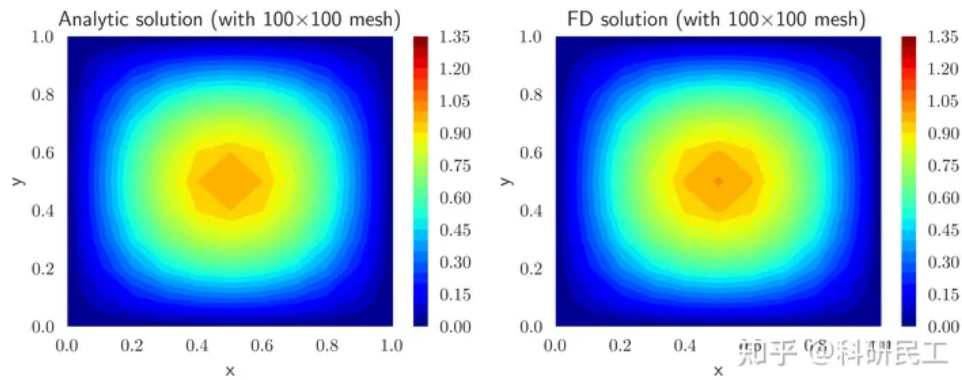


图5. 有限差分法计算二维偏微分问题与解析解的对比

可以看到有限差分法求解得到的结果与解析解基本一致。我们选取 x 与 y 方向相同的网格密度，即令 $M = N = S$ ，得到其最大相对误差随网格密度 S 的变化如图6所示。可以看到在 50×50 计算网格下的计算误差已经可以忽略不计，因此对该问题的有限差分法求解是可靠的。

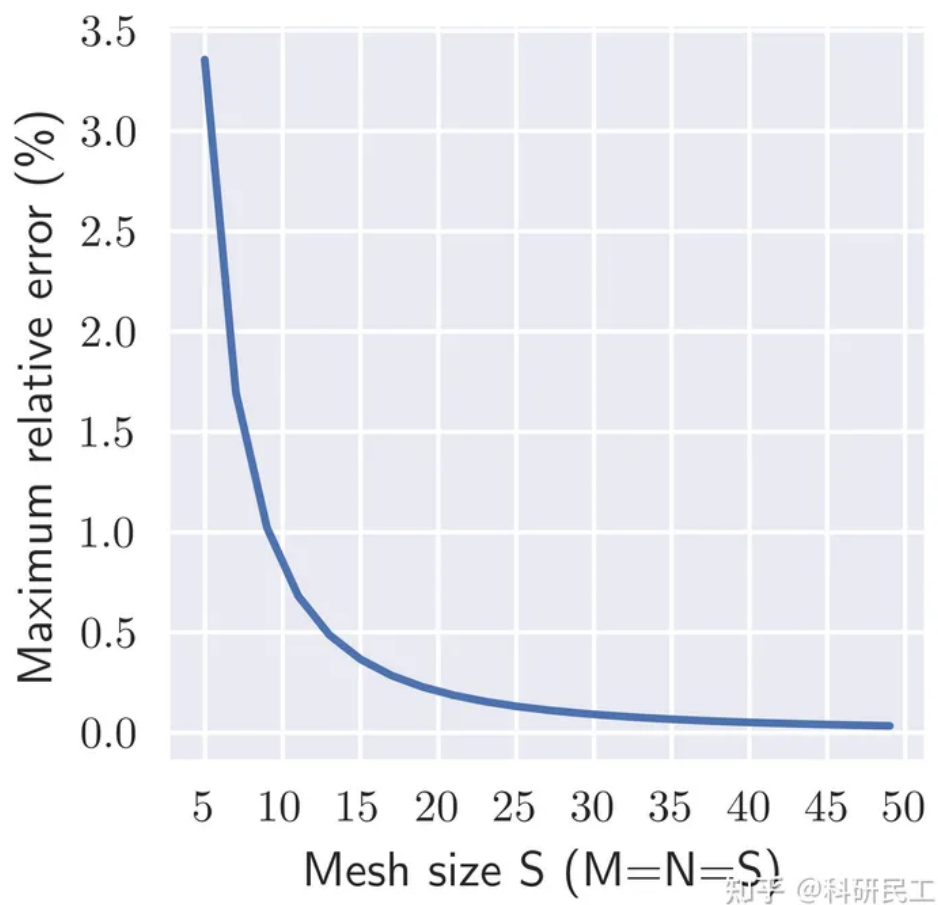


图6. 有限差分法的计算误差随网格密度变化

对应该问题的Python求解代码如下：

```
from numpy import *

M, N = 100, 100
a, b = 1, 1
hx, hy = a/M, b/N
p, q = 1/hx**2, 1/hy**2
r = -2*(p + q)

U = zeros((M-1, M-1))
```



```

for i in range(M-1):
    U[i, i] = r
    if i < M-2: U[i, i+1] = p
    if i > 0:    U[i, i-1] = p
V = diag([q]*(M-1))
Zero_mat = zeros((M-1, M-1))

A_blc = empty((N-1, N-1), dtype=object) # 矩阵A的分块形式
for i in range(N-1):
    for j in range(N-1):
        if i == j:
            A_blc[i, j] = U
        elif abs(i-j) == 1:
            A_blc[i, j] = V
        else:
            A_blc[i, j] = Zero_mat

A = vstack([hstack(A_i) for A_i in A_blc]) # 组装得到矩阵A

x_i = linspace(0, a, M+1)
y_i = linspace(0, b, N+1)
F = vstack([-2*pi**2 * sin(pi*x_i[1:M].reshape((M-1, 1))) * sin(pi*j) for j in

u = dot(linalg.inv(A), F).reshape(M-1, N-1)
u_f = vstack([zeros((1,M+1)), # 最后组装边界条件得到全域的解
               hstack([zeros((N-1,1)), u, zeros((N-1,1))]),
               zeros((1,M+1))])

```

3. 应用实例

根据上面给出的有限差分法的基本方法，这里给出一个具体的应用实例，即枝晶固化生长的相场模拟，使用的也是经典的 Kobayashi 模型^[1]。该问题也是一个典型的相变问题，枝晶固化生长即由液态变为固态的过程。使用变量 $p(\mathbf{r}, t) \in [0, 1]$ 来表示不同相，这里 \mathbf{r}, t 分别为位置变量和时间变量。并且， $p = 0$ 表示液相（未相变）， $p = 1$ 表示固相（已相变）。其中变量 p 满足 Ginzburg-Landau 方程

$$\tau \frac{\partial p}{\partial t} = -\frac{\delta \Phi}{\delta p} \quad (25)$$

这里 τ 为小常数， δ 为变分符号； Φ 为自由能，表达式为：

$$\Phi(p; m) = \int_V \left[\frac{1}{2} \varepsilon^2 |\nabla p|^2 + \frac{1}{4} p^4 - \left(\frac{1}{2} - \frac{m}{3} \right) p^3 + \left(\frac{1}{4} - \frac{m}{2} \right) p^2 \right] dV \quad (26)$$

其中 ε 为梯度能系数，与界面的厚度相关。为了表征界面的各向异性，假设 ε 为界面外法线方向 \mathbf{v} 的函数，即 $\varepsilon(\mathbf{v}) = \varepsilon(-\nabla p)$ ，将式 (26) 带入 (25) 可以得到

$$\tau \frac{\partial p}{\partial t} = -p(p-1) \left(p - \frac{1}{2} + m \right) - \nabla \cdot \left(|\nabla p|^2 \varepsilon \frac{d\varepsilon}{d\mathbf{v}} \right) + \nabla \cdot (\varepsilon^2 \nabla p) \quad (27)$$

这里的推导用到了如下泛函的变分公式：若

$$\Phi[p(\mathbf{r})] = \int_V \mathcal{L}[\mathbf{x}, p(\mathbf{r}), \nabla p(\mathbf{r})] d\mathbf{r} \quad (28)$$

则

$$\frac{\delta\Phi}{\delta p} = \frac{\partial\mathcal{L}}{\partial p} - \nabla \cdot \frac{\partial\mathcal{L}}{\partial\nabla p} \quad (29)$$

这里我们以二维的情况为例进行讨论，引入偏移角 θ 表示 \mathbf{v} 与 \mathbf{x} 轴正向的夹角，即

$$\theta = \arctan\left(\frac{\partial p/\partial y}{\partial p/\partial x}\right) \quad (30)$$

此时的梯度能系数 ε 可以进一步表示为

$$\varepsilon(\theta) = \bar{\varepsilon} \{1 + \sigma \cos[J(\theta - \theta_0)]\} \quad (31)$$

其中 σ 为界面各项异性的强度 (strength of anisotropy) ; J 为界面的各向异性振型数 (mode number of anisotropy) ; θ_0 为初始偏移角。由此，式 (27) 可以进一步变形为

$$\tau \frac{\partial p}{\partial t} = -p(p-1) \left(p - \frac{1}{2} + m \right) - \frac{\partial}{\partial x} \left(\varepsilon \frac{d\varepsilon}{d\theta} \frac{\partial p}{\partial y} \right) + \frac{\partial}{\partial y} \left(\varepsilon \frac{d\varepsilon}{d\theta} \frac{\partial p}{\partial x} \right) + \nabla \cdot (\varepsilon^2 \nabla p) \quad (32)$$

此外，自由能中的参数 m 是界面运动的驱动力，与局部温度 T 相关，

$$m(T) = \left(\frac{\alpha}{\pi} \right) \arctan[\gamma(T_{eq} - T)] \quad (33)$$

其中 α 为正常数， T_{eq} 为平衡温度。同时，局部温度分布满足方程

$$\frac{\partial T}{\partial t} = \nabla^2 T + \kappa \frac{\partial p}{\partial t} \quad (34)$$

这样，式(27)-(34)就给出了该问题的控制方程，包含两个变量 p 和 T 需要求解。需要说明的是，上述给出的控制方程均已经归一化，相关的参数选择为 $\tau = 3 \times 10^{-4}$, $\bar{\varepsilon} = 0.01$, $\sigma = 0.02$, $J = 4$, $\theta_0 = 0.2$, $\alpha = 0.9$, $\gamma = 10$, $T_{eq} = 1$, $\kappa = 1.8$ 。这里我们假设空间求解域为方形区域 $\mathbf{r} = [x, y]$, x, y 方向上的分别划分为 M, N 段；时间域 t 上划分为 K 段，在各方向上的步长分别对应记为 $\Delta x, \Delta y, \Delta t$ 。初始 $t = 0$ 时，在求解域中间取一个区域作为固化的初始点，记为 SN ，如图 7 所示；则在固化点 SN 区域上 $p = 1$ ，在其余区域 $p = 0, T = 0$ ，并随着时间增长枝晶从固化点开始生长。

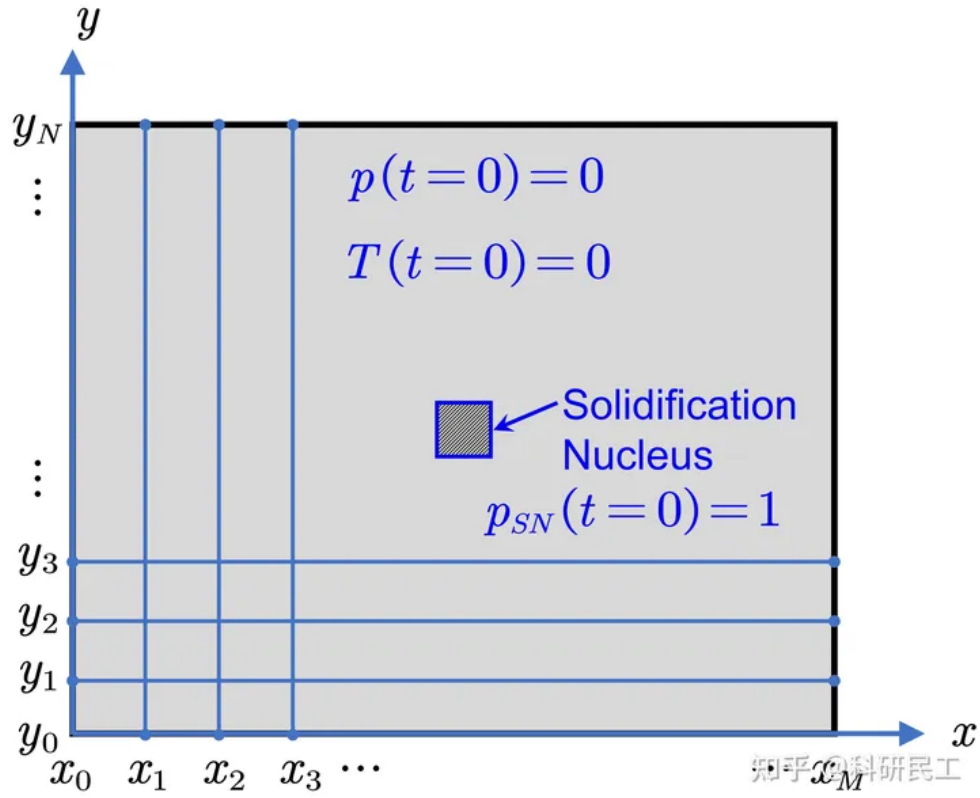


图7. 求解域与初始固化区域

对于这个问题，由于全域的初始值已给定，因此使用迭代法来求解各时间步内的解比较方便。为了方便，使用如下记号： $p_{i,j}^k = p(x_i, y_j, t_k)$ ，其中 i, j, k 分别对应 x, y, t 方向上的结点。下面使用有限差分法对控制方程 (27) 与 (34) 进行离散，其中时域使用经典欧拉显示格式来离散，即

$$\frac{\partial p}{\partial t} = \frac{p_{i,j}^{k+1} - p_{i,j}^k}{\Delta t}, \quad \frac{\partial T}{\partial t} = \frac{T_{i,j}^{k+1} - T_{i,j}^k}{\Delta t} \quad (35)$$

空间域则使用前面讲到的一般性离散方法，例如

$$\frac{\partial p}{\partial x} = \frac{p_{i+1,j}^k - p_{i,j}^k}{\Delta l}, \quad \frac{\partial p}{\partial y} = \frac{p_{i,j+1}^k - p_{i,j}^k}{\Delta l}$$

$$\nabla^2 T = \frac{T_{i+1,j}^k + T_{i-1,j}^k + T_{i,j+1}^k + T_{i,j-1}^k - 4T_{i,j}^k}{(\Delta l)^2} \quad (36)$$

则原控制方程的迭代方程可以推导为：

$$p_{i,j}^{k+1} = p_{i,j}^k + \frac{\Delta t}{\tau} \left[-p_{i,j}^k (p_{i,j}^k - 1) (p_{i,j}^k - 0.5 + m_{i,j}^k) - (\partial g / \partial x)_{i,j}^k + (\partial h / \partial y)_{i,j}^k + (\varepsilon_{i,j}^k)^2 (\nabla^2 p)_{i,j}^k \right] \quad (37)$$

$$T_{i,j}^{k+1} = T_{i,j}^k + \Delta t (\nabla^2 T)_{i,j}^k + \kappa (p_{i,j}^{k+1} - p_{i,j}^k) \quad (38)$$

$$g = \varepsilon \frac{d\varepsilon}{d\theta} \frac{\partial p}{\partial y} = -\varepsilon \bar{\varepsilon} \sigma J \sin[J(\theta - \theta_0)] \frac{\partial p}{\partial y} \quad (39)$$

$$h = \varepsilon \frac{d\varepsilon}{d\theta} \frac{\partial p}{\partial x} = -\varepsilon \bar{\varepsilon} \sigma J \sin[J(\theta - \theta_0)] \frac{\partial p}{\partial x} \quad (40)$$

取 $M = N = 300, K = 4000; \Delta x = \Delta y = 0.03, \Delta t = 1 \times 10^{-4}$ ，求解得到枝晶的生长如图 8 所示，该结果与文献中的实验结果一致，表明有限差分法在该问题的求解中是有效的。

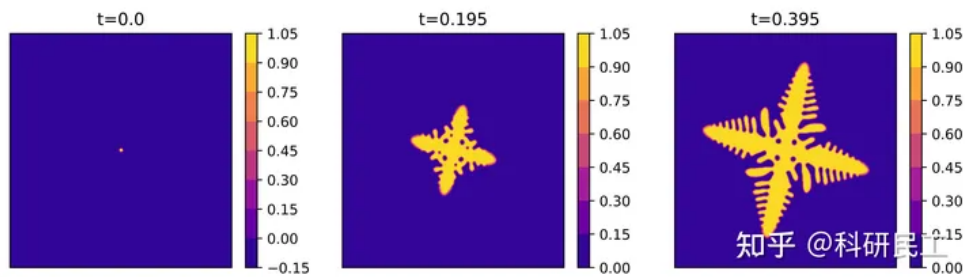


图8. 枝晶生长问题的有限差分法求解结果

可以看到，枝晶边缘的变化曲线比较模糊，这是由于我们划分的网格相对比较粗糙。这里为了计算速度，我们并未取较细的网格。枝晶生长过程的动画如图 9 所示。需要注意的是，在枝晶生长的前 10 计算步中出现了一些 $p < 0$ 的情况，这里为了方便直接取这些元素为零。

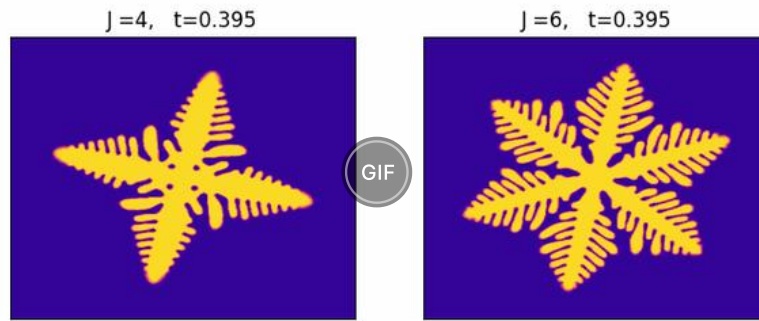


图9. J=4 与 J=6 时的枝晶生长动画

该问题的Python求解代码如下：

```
from numpy import *

# Space and time domain
M, N, K = 300, 300, 4000
Dx, Dy, Dt = 0.03, 0.03, 1e-4

# Material Properties
tau = 3e-4
eps_bar = 0.01
sigma = 0.02
J = 4.
theta_0 = 0.2
alpha = 0.9
gamma = 10.
T_eq = 1.
kappa = 1.8

#%% Evolution
p = zeros((M, N))
T = zeros((M, N))

# Initial Solidification area
for i in range(M):
    for j in range(N):
        if (i - M/2)**2 + (j - N/2)**2 < 5.0:
            p[i, j] = 1.0

# Define Laplacian operator
def Lap(p):
    p_i_j = delete(delete(p, [0, -1], axis=0), [0, -1], axis=1)
    p_im_j = delete(delete(p, [0, -1], axis=0), [-1, -2], axis=1)
```

```

p_ip_j = delete(delete(p, [0, -1], axis=0), [0, 1], axis=1)
p_i_jm = delete(delete(p, [0, -1], axis=1), [0, 1], axis=0)
p_i_jp = delete(delete(p, [0, -1], axis=1), [-1, -2], axis=0)
Lap_p = (p_im_j + p_ip_j + p_i_jm + p_i_jp - 4*p_i_j)/Dx**2
Lap_pj = vstack((Lap_p[0,:], Lap_p, Lap_p[-1,:]))
return hstack((Lap_pj[:,0].reshape(N,1), Lap_pj, Lap_pj[:, -1].reshape(N,1))

# Phase field evolution
def Phase_field(p, T):
    theta = arctan2.gradient(p, Dy, axis=1), gradient(p, Dx, axis=0))
    eps = eps_bar * (1. + sigma * cos(J * (theta - theta_0)))
    g = -eps * eps_bar * sigma * J * sin(J * (theta - theta_0)) * gradient(p, D
    h = -eps * eps_bar * sigma * J * sin(J * (theta - theta_0)) * gradient(p, D
    m = alpha/pi * arctan(gamma * (T_eq - T))
    term_1 = - p*(p - 1.)*(p - 0.5 + m)
    term_2 = - gradient(g, Dx, axis=0)
    term_3 = gradient(h, Dy, axis=1)
    term_4 = eps**2 * Lap(p)
    p_ev = Dt / tau * (term_1 + term_2 + term_3 + term_4)
    return p + p_ev

# Temperature evolution
def Temp(T, p_new, p_old):
    T_ev = Dt*Lap(T) + kappa*(p_new - p_old)
    return T + T_ev

# Evolution process
p_hist = []
T_hist = []
p_old = p; T_old = T
for t_step in range(K):
    p_new = Phase_field(p_old, T_old)
    T_new = Temp(T_old, p_new, p_old)
    p_old = p_new
    T_old = T_new
    if t_step % 50 == 0:
        p_hist.append(p_new)
        T_hist.append(T_new)
        print('step finished:', t_step, '/', str(K))

```

供稿：科研民工；排版：科研民工；审核：科研民工

本文内容仅供学习研究，请勿用作其它商业用途，转载请先联系作者获取授权。

参考

1. [^] R. Kobayashi, Modeling and numerical simulations of dendritic crystal growth, Physica D: Nonlinear Phenomena. 63 (1993) 410–423. [https://doi.org/10.1016/0167-2789\(93\)90120-P](https://doi.org/10.1016/0167-2789(93)90120-P)

编辑于 2021-12-11 14:05

有限差分法

数值模拟

相场模拟

写下你的评论...

67 条评论

默认

最新



文婷

不愧是我的男人👍👍👍

2021-12-15

回复 40

 **Liu Huo** 🏠

路过帮顶，没想到逛个知乎还能遇到同行。
我们给学生上课也是用的这个例子，不过用的是Ingo Steinbach的模型。

2021-12-14


回复 12

 **科研民工** (作者)

👍👍👍👍👍👍

2021-12-15

回复 2

 **静寂的延续**

由浅入深，非常棒👍

2022-01-04

回复 5

 **苏叶**

二阶微分那里，式14和15最后一项分母是不是写错了啊？4的阶乘不是24吗，咋个是12呢，不懂求教

2022-03-17

回复 4

 **放开**

式子15那个分母最后算出来的是12，式子14那个确实错了

2022-09-01

回复 2

 **科研民工** (作者)

是的👍，我写错了。

2022-05-31

回复 喜欢

 **李明健**

高老师是你吗

2022-06-21

回复 4

 **科研民工** (作者)

李老师好👍👍👍

2022-06-21


回复 3

 **黑夜的鲨**

公式 (23) 里面的 $u_{(m,0)}$ 是不是应该想表达 $u_{(i,m)}$

2022-05-30

回复 4

 **不知名大学生**

这里碰见校友😎

2022-08-31

回复 喜欢

 **科研民工** (作者)

是的👍，复制粘贴忘了改了

2022-05-31


回复 喜欢

 **大后生**

牛翻了，5年前自己试着用C++写，没有搞定。

2021-12-12

回复 2

 **你被群主移出群聊**

大神能否把所有的Python代码换成matlab代码😂

2022-06-13

回复 3

 **freely163**

```
clear
clc
M = 100; N = 100;
a = 1; b = 1;
hx = a/M; hy = b/N;
p = 1/hx^2; q = 1/hy^2;
```

```

r = -2 * (p + q);
U = diag(ones(1,M-1)*r,0) + diag(ones(1,M-2)*p,-1) + diag(ones(1,M-2)*p, 1);
V = eye(M-1);
W = ones(N-1,2);
W1 = full(spdia(W,[-1,1],N-1,N-1));
W2 = full(spdia(W,[0,N-1],N-1));
X = kron(W1,V) + kron(W2,U);
F = zeros((M-1),(N-1));
for j = 1:N-1;
for i = 1:M-1;
F(i,j) = -2*pi*pi*sin(pi*i/(M-1))*sin(pi*j/(N-1));
end
end
F1 = F(:);
C = inv(X) * F1;
Z = reshape(C,M-1,N-1)
D1 = zeros(1,M-1);
Z1 = [D1 ; Z; D1]
D2 = zeros(N+1,1)
Z2 = [D2,Z1]
Z3 = [Z2,D2]
[A B] = meshgrid(0:hx:1,0:hy:1)
surface(A,B,Z3);

```

凑合看吧

03-29

回复 4



Bigfish

我也直接粘贴到python里面然后跑不出来😂

04-20

回复 喜欢

展开其他 2 条回复 >



吾心似秋月

绿宝书最后几题过来的

部分绘图代码

取N = 5,10,20,100, 计算并绘制图形到一个2*2的子图中

N = [5, 10, 20, 100]

for n in N:

x, u = fdm(n)

subplot(2, 2, N.index(n)+1)

plot(x, u, 'r-')

plot(x, sin(x), 'b-')

绘制图形

from matplotlib import cm

fig = figure()

ax = fig.gca(projection='3d')

X, Y = meshgrid(x_i, y_i)

surf = ax.plot_surface(X, Y, u_f, rstride=1, cstride=1, cmap=cm.coolwarm,

linewidth=0, antialiased=False)

show()

绘制解析解图形

u_a = sin(pi*x_i.reshape((M+1, 1))) * sin(pi*y_i)

fig = figure()

ax = fig.gca(projection='3d')

X, Y = meshgrid(x_i, y_i)

surf = ax.plot_surface(X, Y, u_a, rstride=1, cstride=1, cmap=cm.coolwarm,

linewidth=0, antialiased=False)

show()

02-16

回复 2



跟着璇推学Comsol

向您学习, 我以后也要把我会的东西, 写文章发出来帮助大家

2022-06-01

回复 1



科研民工 作者



2022-06-01

回复 喜欢

[点击查看全部评论 >](#)

文章被以下专栏收录

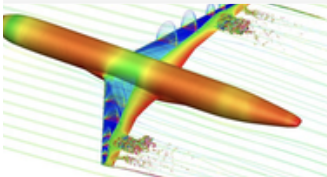


1



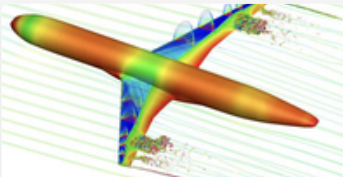
论文
棒

推荐阅读



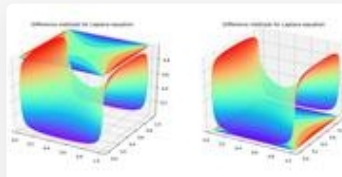
微分方程数值解法1.1:有限差分方法_定义与误差分析

此心安处是... 发表于微分方程数...



微分方程数值解法1.2:有限差分方法_局部误差和全局误差、...

此心安处是... 发表于微分方程数...



数值偏微分方程-差分法 (Python)

孤光一点萤 发表于计算&物理...

常系数差分方程的一般解法

对于形如 $ay_{n-1}+by_{n-2}=d$ (式1) 非齐次常系数二阶差分方程步骤1: 求解 $ay_{n-1}+by_{n-2}=0$ (式2) 齐次常系数二阶差分方程的同解1.1 先求特征方程...

格式化 发表于别了网工, ...