偏微分方程及其解的讨论

$$F(x_1,\ x_2,\ \cdots,\ x_n,\ u_1,\ u_{x_1},\ \cdots,\ u_{x_n},\ u_{x_1x_2},\ \cdots) = 0$$

热传导方程（一维）：

Heat equation(1 dimension)

$$\frac{\partial T}{\partial t}(x,\ t) = \alpha \cdot \frac{\partial^2 T}{\partial x^2}(x,\ t)$$

热传导方程（三维）：

Heat equation(3 dimensions)

$$\frac{\partial T}{\partial t} = \alpha\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right)$$

我们常研究的就是二元函数的二阶偏微分方程：其基本形式：

$$A\frac{\partial^2 f}{\partial x^2} + 2B\frac{\partial^2 f}{\partial x \partial y} + C\frac{\partial^2 f}{\partial y^2} + D\frac{\partial f}{\partial x} + E\frac{\partial f}{\partial y} + Ff = 0$$

如果 *A*、*B*、*C* 三个常系数不全为 0，定义判别式，当判别式大于 0 称其为双曲线式方程；若判别式等于 0，则称其为抛物线式方程；若判别式小于 0，贝则称其为椭圆式方程。

偏微分方程主要有三类：椭圆方程、抛物方程和双曲方程。

- 双曲方程描述变量以一定速度沿某个方向传播，常用于描述振动与波动问题；
- 椭圆方程描述变量以一定深度沿所有方向传播，常用于描述静电场、引力场等稳态问题；
- 抛物方程描述变量沿下游传播，常用于描述热传导和扩散等瞬态问题。

偏微分方程的定解问题通常 很难求出解析解 ，只能通过数值计算方法对偏微分方程的近似求解。

- 常用偏微分方程数值解法有：有限差分方法、有限元方法、有限体方法、共轭梯度法，等等。
- 通常先对问题的求解区域进行网格剖分，然后将定解问题离散为代数方程组求出在离散网格点上的近似值。

- 大名鼎鼎的“韦神”韦东奕在流体流速方程中做出的巨大突破——有关纳维·斯托克斯方程的研究：

$$\rho\left(\frac{\partial v}{\partial t}+v\cdot\nabla v\right)=-\nabla p+\mu\nabla^2 v+f$$

- 这一方程更多的是描述流体流速与流体密度、压力、外阻力之间的关系，在机械工程、能源工程等制造领域有着重要应用。

- 另一个例子是电磁学中重要的麦克斯韦方程组

$$\text{Gauss's law: } \nabla\cdot\mathbf{E}=\frac{\rho}{\varepsilon_0}$$

$$\text{Gauss's law for magnetism: } \nabla\cdot\mathbf{B}=0$$

$$\text{Maxwell–Faraday equation: } \nabla\times\mathbf{E}=-\frac{\partial\mathbf{B}}{\partial t}$$

$$\text{Ampère's circuital law: } \nabla\times\mathbf{B}=\mu_0\left(\mathbf{J}+\varepsilon_0\frac{\partial\mathbf{E}}{\partial t}\right)$$

求解步骤

有限差分法：

- 导入 `numpy`、`matplotlib` 包
- 定义初始条件 $U(x, 0)$
- 输入模型参数 v, p，定义求解的时间域 $(tStart, tEnd)$ 和空间域 $(xMin, xMax)$，设置差分步长 dt、$nNodes$；
- 初始化；
- 递推求解差分方程再区间 $[x_a, x_b]$ 的数值解，获得网格节点处的 $u(t)$ 值。

计算原理

一阶微分线性方程的计算原理：

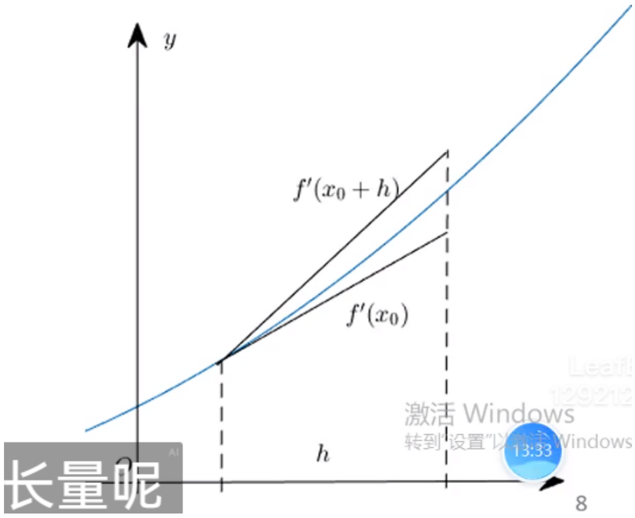
将一阶微分方程离散化，写成迭代、差分的形式：

$$\begin{cases}\dfrac{dy}{dt}=f(y, t)\\ y(t_0)=y_0\end{cases}\longrightarrow\begin{cases}\dfrac{y_{n+1}-y_n}{\Delta t}=\dfrac{f(y_n, t_n)+f(y_{n+1}, t_{n+1})}{2}\\ y(0)=y_0\end{cases}$$

$f(y, t)$ 是一阶导也是一阶微分方程。（优化的欧拉法）

从而我们得到递推关系：

$$\begin{cases}y_{n+1}=y_n+\dfrac{\Delta t}{2}(f(y_n, t_n)+f(y_{n+1}, t_{n+1}))\\ y(0)=y_0\end{cases}$$



例 1

对于一个 *RC* 回路，我们有：

$$IR + \frac{Q}{C} = 0$$

其中，I、R、Q、C 分别为电流、电阻、电量和电容，利用 $I = \frac{dQ}{dt}$，并定义 $\tau \equiv RC$ ，我们得到一个含初始条件的一阶常微分方程：

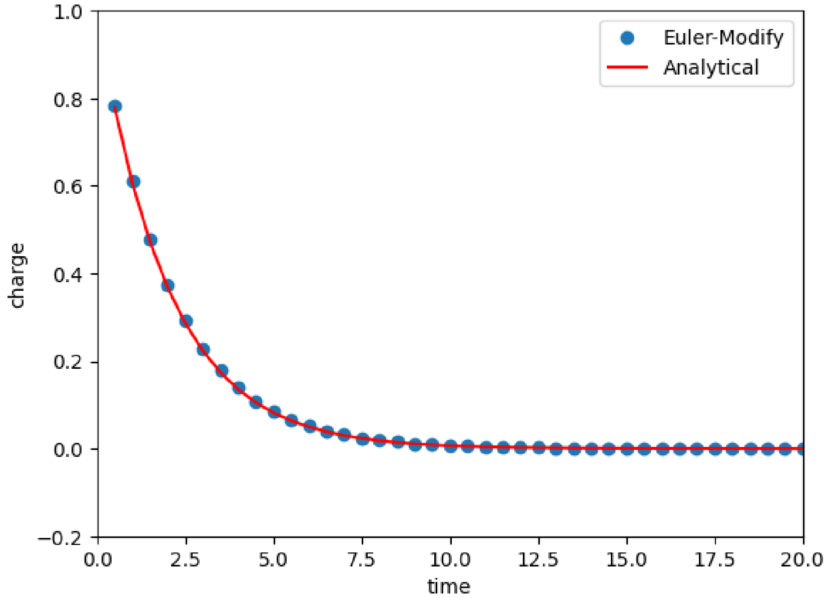
$$\begin{cases} \frac{dQ}{dt} = -\frac{Q}{\tau} \\ Q(t_0) = Q_0 \end{cases}$$

可以解析求解，得到 $Q = Q_0 e^{-\frac{t}{\tau}}$；使用差分法：

$$\begin{cases} Q_{n+1} = Q_n - \frac{1}{2}\left(\frac{Q_n}{\tau} + \frac{Q_{n+1}}{\tau}\right)\Delta t \\ Q(0) = Q_0 \end{cases}$$

```
rc = 2.0  # 设置常量
dt = 0.5  # 设置步长
n = 1000  # 设置分割段数
t = 0.0  # 设置初始时间
q = 1.0  # 设置初始电量

# 先定义三个空列表
qt = []  # 用来盛放差分得到的 q 值
qt0 = []  # 用来盛放解析得到的 q 值
time = []  # 用来盛放时间值

for i in range(n):
    t = t + dt
    q1 = q - q * dt / rc  # qn+1 的近似值
    q = q - 0.5 * (q1 * dt / rc + q * dt / rc)  # 差分递推关系
    q0 = np.exp(-t / rc)  # 解析关系
```

```
    qt.append(q)  # 差分得到的 q 值列表
    qt0.append(q0)  # 解析得到的 q 值列表
    time.append(t)  # 时间列表

plt.plot(time, qt, "o", label="Euler-Modify")  # 差分得到的电量随时间的变化
plt.plot(time, qt0, "r-", label="Analytical")  # 解析得到的电量随时间的变化
plt.xlabel("time")
plt.ylabel("charge")
plt.xlim(0, 20)
plt.ylim(-0.2, 1.0)
plt.legend(loc="upper right")
plt.show()
```



上面的差分方法为梯形法，即采用该点与后一点的斜率的平均值来表示当前点的变化。而欧拉方法（*Euler*）仅采用当前点的值。

一般二阶抛物型偏微分方程

一维热传导问题：

$$\begin{cases} \dfrac{\partial u(x,\ t)}{\partial t} = \lambda \dfrac{\partial^2 u(x,\ t)}{\partial x^2} & 0 \le t \le T,\ 0 \le x \le l \\ u(x,\ 0) = f(x) \\ u(0,\ t) = g_1(t) \\ u(l,\ t) = g_2(t) \end{cases} \longrightarrow \begin{cases} \dfrac{\partial^2 u(x,\ t)}{\partial x^2}|_{i,\ k} = \dfrac{u_{i-1,\ k} - 2u_{i,\ k} + u_{i+1,\ k}}{h^2} \\ \dfrac{\partial u(x,\ t)}{\partial t}|_{i,\ k} = \dfrac{u_{i,\ k+1} - u_{i,\ k}}{\tau} \end{cases}$$

其中 h 和 t 分别为空间步长和时间步长，i 和 j 分别标记空间指标和时间指标，则得到差分方程：

$$
\begin{cases}
\dfrac{u_{i,\,k+1}-u_{i,\,k}}{\tau}=\lambda\dfrac{u_{i-1,\,k}-2u_{i,\,k}+u_{i+1,\,k}}{h^2} \\
u_{i,\,0}=f(ih) \quad i=1,\,2,\,\ldots,\,N-1,\,N=\dfrac{l}{h} \\
u_{0,\,k}=g_1(k\tau) \quad k=0,\,1,\,\ldots,\,M-1,\,M=\dfrac{T}{\tau} \\
u_{N,\,k}=g_2(k\tau)
\end{cases}
$$

$u_{i,\,0}$ 代表 $t=0$ 时，温度的变化随位置的变化；$u_{0,\,k}$ 和 $u_{N,k}$ 代表最左和最优的温度随时间的变化。

而 $\dfrac{\partial^2 u(x,\,t)}{\partial x^2}|_{i,\,k}=\dfrac{u_{i-1,\,k}-2u_{i,\,k}+u_{i+1,\,k}}{h^2}$ 的得法：

$$\frac{\partial u(x,\,t)}{\partial x}|_{i,\,k}=\frac{u_{i+1,\,k}-u_{i,\,k}}{h}$$

$$\frac{\partial^2 u(x,\,t)}{\partial x^2}|_{i,\,k}=\frac{\frac{u_{i+1,\,k}-u_{i,\,k}}{h}-\frac{u_{i,\,k}-u_{i-1,\,k}}{h}}{h}=\frac{u_{i-1,\,k}-2u_{i,\,k}+u_{i+1,\,k}}{h^2}$$

例 2

一维热传导方程是一个典型的抛物型二阶偏微分方程。设 $u(x,t)$ 表示在时间 t，空间 x 处的温度，则根据傅里叶定律（单位时间内流经单位面积的热量和该处温度的负梯度成正比），可以导出热传导方程【2】

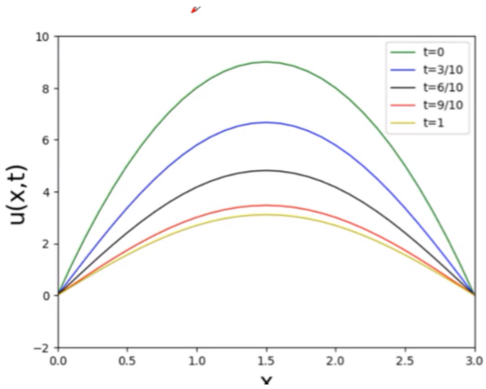
$$\frac{\partial u}{\partial t}=\lambda\frac{\partial^2 u}{\partial x^2}$$

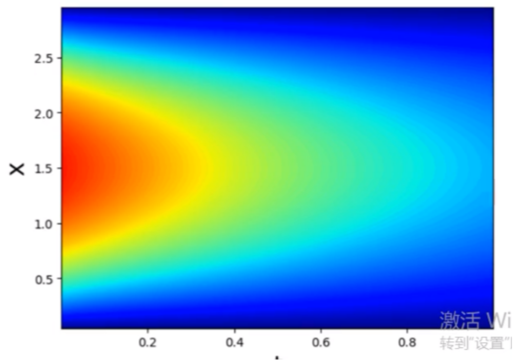
其中 $\lambda\equiv\kappa/c\rho$ 称为热扩散率，κ,c,ρ 分别为热导率，比热和质量密度，都是由系统本身确定的常量。

为了具体，设 $\lambda=1,l=3,T=1$，设边界条件为

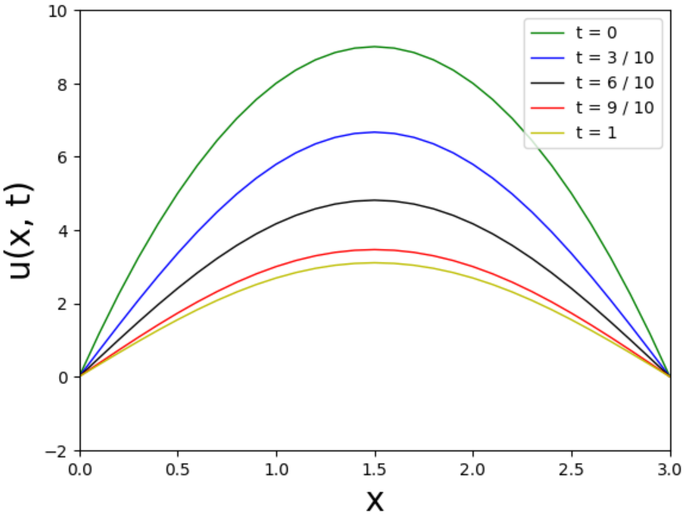
$$
\begin{cases}
\dfrac{\partial u(x,t)}{\partial t}=\dfrac{\partial^2 u(x,t)}{\partial x^2} \qquad 0\le t\le 1000,0\le x\le 1 \\
u(x,0)=4x(3-x) \\
u(0,t)=0 \\
u(3,t)=0
\end{cases}
$$

我们首先把位于 $0-3$ 中的空间等分为 30 份，位于 $0-1$ 的时间等分为 10000 份，然后通过设置初始条件、边界条件和递推关系并借助 `for` 循环就得到了 1 个 $30*10000$ 的二维数组，里面放着每个离散的时空点的温度值。

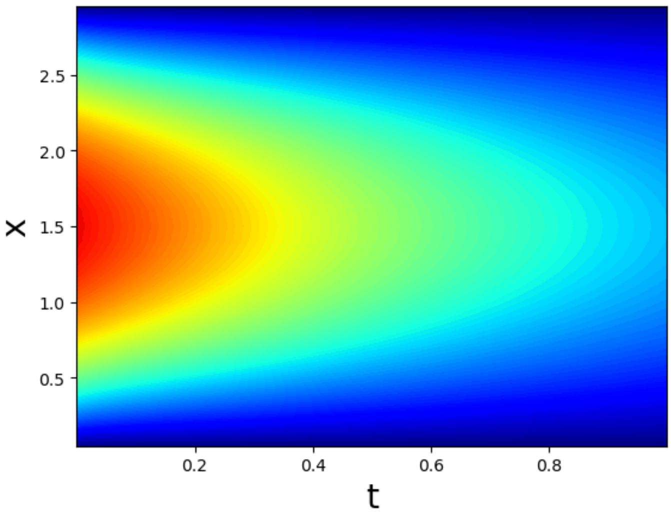





```
h = 0.1  # 空间步长
N = 30  # 空间步数
dt = 0.0001  # 时间步长
M = 10000  # 时间步数
A = dt / (h**2)  # lambda*tau/h^2
U = np.zeros([N + 1, M + 1])  # 建立二维空数组
Space = np.arange(0, (N + 1) * h, h)  # 建立空间等差数列，从 0 到 3，公差是 h

for k in np.arange(0, M + 1):
    U[0, k] = 0.0
    U[N, k] = 0.0

for i in np.arange(0, N + 1):
    U[i, 0] = 4 * i * h * (3 - i * h)

# 递推关系
for k in np.arange(0, M):
    for i in np.arange(1, N):
        U[i, k + 1] = A * U[i + 1, k] + (1 - 2 * A) * U[i, k] + A * U[i - 1, k]

plt.plot(Space, U[:, 0], "g-", label="t = 0", linewidth=1.0)
plt.plot(Space, U[:, 3000], "b-", label="t = 3 / 10", linewidth=1.0)
plt.plot(Space, U[:, 6000], "k-", label="t = 6 / 10", linewidth=1.0)
plt.plot(Space, U[:, 9000], "r-", label="t = 9 / 10", linewidth=1.0)
plt.plot(Space, U[:, 10000], "y-", label="t = 1", linewidth=1.0)
plt.ylabel("u(x, t)", fontsize=20)
plt.xlabel("x", fontsize=20)
plt.xlim(0, 3)
plt.ylim(-2, 10)
plt.legend(loc="upper right")
plt.show()
```




```
# 温度等高线随时空坐标的变化，温度越高，颜色越偏红
extent = [0, 1, 0, 3]  # 时间和空间的取值范围
levels = np.arange(0, 10, 0.1)  # 温度等高线的变化范围 0-10，变化间隔为 0.1
plt.contourf(U, levels, origin="lower", extent=extent, cmap=plt.cm.jet)
plt.ylabel("x", fontsize=20)
plt.xlabel("t", fontsize=20)
plt.show()
```



1. `extent = [0, 1, 0, 3]`：定义了时间和空间的取值范围，分别为 x 轴从 0 到 1，y 轴从 0 到 3。
2. `levels = np.arange(0, 10, 0.1)`：定义了温度等高线的变化范围，从 0 到 10，间隔为 0.1。
3. `plt.contourf(U, levels, origin="lower", extent=extent, cmap=plt.cm.jet)`：这一行是绘制等高线图的关键。`U` 应该是包含温度数据的二维数组，其形状应与定义的时间和空间取值范围一致。`levels` 定义了等高线的值，`origin="lower"` 表示原点在左下角，`extent=extent` 指定了时间和空间的取值范围，`cmap=plt.cm.jet` 指定了等高线的颜色映射，这里使用了 "jet" 色彩映射，表示温度越高颜色越偏红。

例 3 一维平流分程

平流过程是大气运动中重要的过程。平流方程（Advection Equation）描述某一物理量的平流作用而引起局部变化的物理过程，最简单的形式是一维平流方程。

$$\begin{cases} \frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = 0 \\ u(x,\, 0) = F(x) \\ 0 \leq t \leq t_e \\ x_a \leq x \leq x_b \end{cases}$$

式中 u 为某物理量，v 为系统速度，x 为水平方向分量，t 为时间。该方程求得解析解：

$$u(x,\, t) = F(x - vt),\ 0 \leq t \leq t_e$$

考虑一维线性平流微分方程的数值解法，采用有限差分法求解。简单地，采用一阶迎风格式的差分方法（First-order Upwind），一阶导数的差分表达式为：

$$(\frac{\partial u}{\partial x})_{i,\, j} = \frac{u_{i+1,\, j} - u_{i,\, j}}{\Delta x} + o(\Delta x)$$

于是得到解析形式：

$$u_{i,\, j+1} = u_{i,\, j} - \frac{vdt}{dx(u_{i,\, j} - u_{i-1,\, j})}$$

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

# 初始条件函数 U(x, 0)
def initial_condition(x, p):
    u0 = np.sin(2 * (x - p) ** 2)
    return u0

# 输入参数
velocity = 1.0  # 平流方程参数，系统速度
p = 0.25  # 初始条件函数 U(x, 0) 中的参数

# 时间参数
start_time = 0  # 开始时间
end_time = 1.0  # 终止时间：(0, end_time)
dt = 0.02  # 时间步长
n_nodes = 100  # 空间网格数

# 初始化
n_steps = round(end_time / dt)
dx = np.pi / n_nodes
x = np.arange(0, np.pi + 2 * dx, dx)
u_curr = initial_condition(x, p)
u_next = u_curr.copy()

```

```python
# 时域差分
for _ in range(n_steps):
    plt.clf()  # 清除当前 figure 的所有 axes，但是保留当前窗口

    # 计算 u(j + 1)
    for i in range(1, n_nodes + 1):
        u_next[i] = u_curr[i] - (velocity * dt / dx) * (u_curr[i] - u_curr[i - 1])

    # 更新边界条件
    u_next[0] = u_next[n_nodes]
    u_next[n_nodes + 1] = u_next[1]
    u_curr = u_next.copy()

    # 更新时间
    start_time += dt

    # 绘图
    plt.plot(x, u_curr, "b-", label="Velocity = 1.0")
    plt.axis((0 - 0.1, np.pi + 0.1, -1.1, 1.1))
    plt.xlabel("x")
    plt.ylabel("U(x)")
    plt.legend(loc=(0.05, 0.05))
    plt.title(f"Advection equation with finite difference method, t = {start_time + dt:.1f}")
    plt.text(0.05, 0.9, "ypucans-xupt", color="gainsboro")
    plt.pause(0.001)

plt.show()

```

例 4 二维波动方程

波动方程（Wave Equation）是典型的双曲形偏微分方程，广泛应用于声学、电磁学和流体力学等领域，描述自然界中的各种波动现象，包括横波和纵波，例如声波、光波和水波。

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = c^2(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}) \\ \dfrac{\partial}{\partial t}u(0,\ x,\ y) = 0 \\ u(x,\ y,\ 0) = u_0(x,\ y) \\ u(0,\ y,\ t) = u_a(t),\ u(1,\ y,\ t) = u_b(t) \\ u(x,\ 0,\ t) = u_c(t),\ u(x,\ 1,\ t) = u_d(t) \\ 0 \le t \le t_e,\ 0 < x < 1,\ 0 < y < 1 \end{cases}$$

式中：u 为振幅；c 为波的传播速率，c 可以是固定常数，或位置的函数 $c(x,\ y)$，也可以是振幅的函数 $c(u)$。

考虑二维波动偏微分方程的数值解法，采用有限差分法求解。简单地，采用迎风法的三点差分格式：

$$\left(\frac{u_{i,\,j}^{k+1}-2u_{i,\,j}^{k}+u_{i,\,j}^{k-1}}{\Delta t^2}\right)=c^2\left(\frac{u_{i-1,\,j}^{k}-2u_{i,\,j}^{k}+u_{i+1,\,j}^{k}}{\Delta x^2}+\frac{u_{i-1,\,j}^{k}-2u_{i,\,j}^{k}+u_{i+1,\,j}^{k}}{\Delta y^2}\right)$$

化简：

$$\begin{cases} u_{i,\,j}^{k+1}=r_x(u_{i-1,\,j}^{k}+u_{i+1}^{k})+2(1-r_x-r_y)u_{i,\,j}^{k}+r_y(u_{i,\,j-1}^{k}+u_{i,\,j+1}^{k})-u_{i,\,j}^{k-1} \\ r_x=c^2\dfrac{\Delta t^2}{\Delta x^2} \\ r_y=c^2\dfrac{\Delta t^2}{\Delta y^2} \end{cases}$$

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

# 模型参数
c = 1.0  # 波的传播速度
tc, te = 0.0, 1.0  # 时间范围：(0, te)
xa, xb = 0.0, 1.0  # 空间范围：(xa, xb)
ya, yb = 0.0, 1.0  # 空间范围：(ya, yb)

# 初始化
c2 = c * c  # 方程参数
dt = 0.01  # 时间步长
dx = dy = 0.02  # 空间步长
t_nodes = round(te / dt)  # t轴 时序网格数
x_nodes = round((xb - xa) / dx)  # x轴 空间网格数
y_nodes = round((yb - ya) / dy)  # y轴 空间网格数
t_zone = np.arange(0, (t_nodes + 1) * dt, dt)  # 建立空间网格
x_zone = np.arange(0, (x_nodes + 1) * dx, dx)  # 建立空间网格
y_zone = np.arange(0, (y_nodes + 1) * dy, dy)  # 建立空间网格
xx, yy = np.meshgrid(x_zone, y_zone)  # 生成网格点的坐标 xx, yy (二维数组)

# 步长比检验（r > 1，则算法不稳定）
r = 4 * c2 * dt * dt / (dx * dx + dy * dy)
print("dt = {:.2f}, dx = {:.2f}, dy = {:.2f}, r = {:.2f}".format(dt, dx, dy, r))
assert r < 1.0, "Error: r > 1, unstable step ratio of dt2 / (dx2 + dy2)."
rx = c2 * dt**2 / dx**2
ry = c2 * dt**2 / dy**2

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d")

```

```python
# 计算初始值
U = np.zeros([t_nodes + 1, x_nodes + 1, y_nodes + 1])  # 建立三维数组
U[0] = np.sin(6 * np.pi * xx) + np.cos(4 * np.pi * yy)  # U[0, :, :]
U[1] = np.sin(6 * np.pi * xx) + np.cos(4 * np.pi * yy)  # U[1, :, :]
surf = ax.plot_surface(xx, yy, U[0, :, :], rstride=2, cstride=2, cmap=plt.cm.coolwarm)

for k in range(2, t_nodes + 1):
    for i in range(1, x_nodes):
        for j in range(1, y_nodes):
            U[k, i, j] = (
                rx
                * (
                    U[k - 1, i - 1, j]
                    + U[k - 1, i + 1, j]
                    + ry * (U[k - 1, i, j - 1])
                    + U[k - 1, i, j + 1]
                )
                + 2 * (1 - rx - ry) * U[k - 1, i, j]
                - U[k - 2, i, j]
            )
    surf = ax.plot_surface(xx, yy, U[k, :, :], rstride=2, cstride=2, cmap="rainbow")
    ax.set_title(f"2D wave equation (t = {k * dt:.2f})")
    plt.pause(0.001)

plt.show()
```

例 5 二维抛物方程

热传导方程（Heat Equation）是典型的抛物形偏微分方程，也成为扩散方程。广泛应用于声学，电磁学，和流体力学等领域，描述自然界中的各种的波动现象，包括横波和纵波，例如声波、光波和水波。

$$\begin{cases} \dfrac{\partial u}{\partial t} = \lambda\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + q_v\right) \\ u(x,\ y,\ 0) = u_0(x,\ y) \\ u(0,\ y,\ t) = u_a(t),\ u(1,\ y,\ t) = u_b(t) \\ u(x,\ 0,\ t) = u_c(t),\ u(x,\ 1,\ t) = u_d(t) \\ 0 \le t \le t_e,\ 0 < x < 1,\ 0 < y < 1 \end{cases}$$

式中 λ 为热扩散率，取决于材料本身的热传导率、密度和热容；q_v 是热源强度，可以是恒定值，也可以是时间、空间的函数。

有：

$$\begin{cases} \left(\dfrac{u_{i,\,j}^{k+1}-u_{i,\,j}^{k}}{\Delta t}\right)=\lambda\left(\dfrac{\partial^2 u^k}{\partial x^2}+\dfrac{\partial^2 u^k}{\partial y^2}\right)+q_v \\ \dfrac{\partial^2 u_{i,\,j}}{\partial x^2}=\dfrac{(u_{i,\,j-1}-2u_{i,\,j}+u_{i,\,j+1})}{\Delta x^2} \\ \dfrac{\partial^2 u_{i,\,j}}{\partial y^2}=\dfrac{(u_{i,\,j-1}-2u_{i,\,j}+u_{i,\,j+1})}{\Delta y^2} \\ u_0=u_0(x,\;y) \\ u(x,\;y,t_{k+1})=u^{k+1} \end{cases}$$

$$\begin{cases} U_{k+1}=U_k+r_x\cdot A\cdot U_k+r_y\cdot B\cdot U_k+q_v\cdot dt \\ A=\begin{bmatrix} -2 & 1 & \cdots & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -2 \end{bmatrix}_{(N_{x+1},\,N_{x+1})} \qquad B=\begin{bmatrix} -2 & 1 & \cdots & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -2 \end{bmatrix}_{(N_{y+1},\,N_{y+1})} \end{cases}$$

化简后:

$$\begin{cases} u_{i,\,j}^{k+1}=r_x(u_{i-1,\,j}^k+u_{i+1,j}^k)+2(1-r_x-r_y)u_{i,\,j}^k+r_y(u_{i,\,j-1}^k+u_{i,\,j+1}^k)-u_{i,\,j}^{k-1} \\ r_x=\lambda\dfrac{\Delta t}{\Delta x^2} \\ r_y=\lambda\dfrac{\Delta t}{\Delta y^2} \end{cases}$$

例 6 二维椭圆方程

椭圆型偏微分方程是一类重要的偏微分方程，主要用来描述物理的平衡稳定状态，如定常状态下的电磁场、引力场和反应扩散现象等，广泛应用于流体力学、弹性力学、电磁学、几何学和变分法中。

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=f(x,\;y)$$

- 考虑二维椭圆型偏微分方程的数值解法，采用有限差分法求解。简单地，采用五点差分格式表示二阶导数的差分表达式，将上述的偏微分方程离散为差分方程：
$$(\mathrm{u}_{\mathrm{i-1,j}}-2\mathrm{u}_{\mathrm{i,j}}+\mathrm{u}_{\mathrm{i+1,j}})/\Delta\mathrm{h}^2+(\mathrm{u}_{\mathrm{i,j-1}}-2\mathrm{u}_{\mathrm{i,j}}+\mathrm{u}_{\mathrm{i,j+1}})/\Delta\mathrm{h}^2=\mathrm{f}_{\mathrm{i,j}}$$
- 椭圆型偏微分描述不随时间变化的均衡状态，没有初始条件，因此不能沿时间步长递推求解。对上式的差分方程，可以通过矩阵求逆方法求解，但当 h 较小时网格很多，矩阵求逆的内存占用和计算量极大。于是，可以使用迭代松弛法递推求得二维椭圆方程的数值解：

$$\mathrm{u}_{\mathrm{i,j}}^{\mathrm{k+1}}=(1-\mathrm{w})*\mathrm{u}_{\mathrm{i,j}}^{\mathrm{k}}+\mathrm{w}/4*(\mathrm{u}_{\mathrm{i,j+1}}^{\mathrm{k}}+\mathrm{u}_{\mathrm{i,j-1}}^{\mathrm{k}}+\mathrm{u}_{\mathrm{i-1,j}}^{\mathrm{k}}+\mathrm{u}_{\mathrm{i+1,j}}^{\mathrm{k}}-\mathrm{h}^2*\mathrm{f}_{\mathrm{i,j}})$$

偏微分方程及其解的讨论

少数方程可以用 `sympy` 中的 `pdsolve` 求解:

`pdsolve(eq, func=None, hint='default', dict=False, solvefun=None, **kwargs)[source]`

只有少部分方程是存在符号解的。