微分方程数值求解基础

ZHONGSHAN XU

2020/7

微分方程的分类

$$A\frac{\partial^2 u}{\partial x^2}+2B\frac{\partial^2 u}{\partial x\partial y}+C\frac{\partial^2 u}{\partial y^2}+(\text{lower order derivative terms})=0$$

- $AC-B^2>0$：椭圆方程（elliptical equation）
  - Poisson equation：$u_{xx}+u_{yy}=s$
- $AC-B^2<0$：双曲方程（hyperbolic equation）
  - Wave equation：$u_{tt}-c^2u_{xx}=0$
- $AC-B^2=0$：抛物方程（parabolic equation）
  - Heat equation：$u_t-\alpha u_{xx}=0$

微分方程转化为代数方程

- 解微分方程最终转化为三类线性代数问题
  - 线性方程组求解：$Ax = b$
    - 直接法：LU分解（Gauss 消元）
    - 迭代法：经典迭代（基于矩阵分裂）和现代迭代（基于Krylov子空间）
  - 线性本证值问题：$Ax = \lambda Bx$
  - 矩阵乘法（张量缩并）和 element-wise operations

离散化

- 变量$x \Rightarrow \boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T$

$$a = x_1, x_2, \cdots, x_j, \cdots, x_{n-1}, x_n = b$$

- 函数$f(x) \Rightarrow \boldsymbol{f} = [f_1 \quad f_2 \quad \cdots \quad f_n]^T$

$$f_1 \equiv f(x_1), f_2 \equiv f(x_2), \cdots, f_j \equiv f(x_j), \cdots, f_{n-1} \equiv f(x_{n-1}), f_n \equiv f(x_n)$$

- 导数$f^{(k)}(x) \Rightarrow \boldsymbol{f}^{(k)} = \left[f_1^{(k)} \quad f_2^{(k)} \quad \cdots \quad f_n^{(k)}\right]$

$$f_1^{(k)} \equiv f^{(k)}(x_1), f_2^{(k)} \equiv f^{(k)}(x_2), \cdots, f_j^{(k)} \equiv f^{(k)}(x_j), \cdots, f_{n-1}^{(k)} \equiv f^{(k)}(x_{n-1}), f_n^{(k)} \equiv f^{(k)}(x_n)$$

- 有限差分法和伪谱法(pseudo-spectral method)通过格点函数值的线性组合来近似导数，即：

$$f_i^{(k)} = \sum_j D_{ij}^{(k)} f_j \Leftrightarrow \boldsymbol{f}^{(k)} = D^{(k)} \boldsymbol{f}$$

离散化

- 微分矩阵

$$\begin{bmatrix} f_1^{(k)} \\ f_2^{(k)} \\ \vdots \\ f_j^{(k)} \\ \vdots \\ f_{n-1}^{(k)} \\ f_n^{(k)} \end{bmatrix} = \begin{bmatrix} D_{11}^{(k)} & D_{12}^{(k)} & \cdots & D_{1j}^{(k)} & \cdots & D_{1,n-1}^{(k)} & D_{1n}^{(k)} \\ D_{21}^{(k)} & D_{21}^{(k)} & \cdots & D_{2j}^{(k)} & \cdots & D_{2,n-1}^{(k)} & D_{2,n}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{j1}^{(k)} & D_{j2}^{(k)} & \cdots & D_{jj}^{(k)} & \cdots & D_{j,n-1}^{(k)} & D_{j,n}^{(k)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ D_{n-1,1}^{(k)} & D_{n-1,2}^{(k)} & \cdots & D_{n-1,j}^{(k)} & \cdots & D_{n-1,n-1}^{(k)} & D_{n-1,n}^{(k)} \\ D_{n1}^{(k)} & D_{12}^{(k)} & \cdots & D_{nj}^{(k)} & \cdots & D_{n,n-1}^{(k)} & D_{nn}^{(k)} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_j \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

有限差分微分矩阵

- 一阶微分矩阵

$$f_j^{(1)} \approx \frac{f_{j+1} - f_{j-1}}{2\Delta x}, j = 2,3,\cdots,n-1$$

$$f_1^{(1)} = \frac{-3f_1 + 4f_2 - f_3}{2\Delta x}, f_n^{(1)} = \frac{f_n - 4f_{n-1} + 3f_{n-2}}{2\Delta x}$$

$$D^{(1)} = \frac{1}{2\Delta x}\begin{bmatrix} -3 & 4 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & -1 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 3 & -4 & 1 \end{bmatrix}$$

$$\boldsymbol{f}^{(1)} = D^{(1)}\boldsymbol{f}$$

有限差分微分矩阵

- 二阶微分矩阵

$$f_j^{(2)} = \frac{f_{j-1} - 2f_j + f_{j+1}}{\Delta x^2}, j = 2,3,\cdots,n-1$$

$$f_1^{(2)} = \frac{f_1 - 2f_2 + f_3}{\Delta x^2}, f_N^{(2)} = \frac{f_{n-2} - 2f_{n-1} + f_n}{\Delta x^2}$$

$$D^{(2)} = \frac{1}{\Delta x^2}\begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ -1 & -2 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & -2 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & -2 & 1 & 0 \\ 0 & 0 & \cdots & 0 & -1 & -2 & 1 \\ 0 & 0 & \cdots & 0 & 1 & -2 & 1 \end{bmatrix}$$

$$\boldsymbol{f}^{(2)} = D^{(2)}\boldsymbol{f}$$

微分方程离散化

- 一维线性微分方程

$$A\partial_x^2 f + B\partial_x f + Cf = R$$
$$B_a\partial_x f + C_a f = R_a$$
$$B_b\partial_x f + C_b f = R_b$$

- 离散化

$$AD^{(2)}\boldsymbol{f} + BD^{(1)}\boldsymbol{f} + CI\boldsymbol{f} = \underbrace{\left(AD^{(2)} + B\mathrm{D} + CI\right)}_{M}\boldsymbol{f} = \boldsymbol{R}$$
$$\left(B_a D_1^{(1)} + C_a I_1\right)\boldsymbol{f} = R_a$$
$$\left(B_b D_n^{(1)} + C_b I_n\right)\boldsymbol{f} = R_b$$

- 边界条件替换

$$M_1 = B_a D_1^{(1)} + C_a I_1, M_n = \left(B_b D_n^{(1)} + C_b I_n\right); R_1 = R_a, R_n = R_b$$
$$\boldsymbol{f} = M^{-1}\boldsymbol{R}$$

牛顿迭代——将非线性问题转化为线性问题！

- 牛顿迭代（本质用线性化后方程的解去逐步逼近）
  - 一维情形：$f(x)=0$

线性化

$$f(x_0)+f'(x_0)(x-x_0)=0 \Rightarrow f'(x_0)(x-x_0)=-f(x_0)$$

$$\Rightarrow x \approx x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

逐步逼近

$$f'(x_n)(x_{n+1}-x_n)=-f(x_n)$$

  - 高维情形：$\boldsymbol{F}(\boldsymbol{x})=\boldsymbol{0}$

$$J_F(\boldsymbol{x}_n)\cdot(\boldsymbol{x}_{n+1}-\boldsymbol{x}_n)=-\boldsymbol{F}(\boldsymbol{x}_n)$$

$$\boldsymbol{x}_{n+1}=\boldsymbol{x}_n-J_F^{-1}(\boldsymbol{x}_n)\cdot\boldsymbol{F}(\boldsymbol{x}_n)$$

$$\boldsymbol{x}_0 \rightarrow \{F(\boldsymbol{x}_0), J_F(\boldsymbol{x}_0)\} \rightarrow (\boldsymbol{x}_1-\boldsymbol{x}_0) \rightarrow \boldsymbol{x}_1 \rightarrow \cdots$$

微分方程情形

- 对方程作变分后线性化

$$E[x,f]=0$$

$$E[x,f_0]+\left.\frac{\delta E}{\delta f}\right|_{f_0}\underbrace{(f-f_0)}_{\delta f}=0$$

- 例如非线性微分方程

$$E[x,f]=f''+f'e^f+\sin f+x\ln f=0$$

- 变分

$$\delta E=\delta f''+e^f\delta f'+f'e^f\delta f+\cos f\,\delta f+\frac{x}{f}\delta f$$

$$=\left[D^{(2)}+e^{\boldsymbol{f}}D^{(1)}+\left(D^{(1)}\boldsymbol{f}e^{\boldsymbol{f}}\right)I+(\cos\boldsymbol{f})I+\left(\frac{\boldsymbol{x}}{\boldsymbol{f}}\right)I\right]\delta f$$

- 线性化后的微分方程

$$E[x,f_0]+\delta E[x,f_0,\delta f]=0\Rightarrow\delta E=-E[x,f_0]$$

非线性微分方程的求解过程

二维求导

- 二维格点分布

$$\begin{bmatrix} f(x_1,y_1) & f(x_1,y_2) & \cdots & f(x_1,y_n) \\ f(x_2,y_1) & f(x_2,y_2) & \cdots & f(x_2,y_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_m,y_1) & f(x_m,y_2) & \cdots & f(x_m,y_n) \end{bmatrix}$$

- 对x方向求偏导

$$D_x^{(k)} f$$

二维求导

- 二维格点转置

$$\begin{bmatrix} f(x_1,y_1) & f(x_2,y_1) & \cdots & f(x_m,y_1) \\ f(x_1,y_2) & f(x_2,y_2) & \cdots & f(x_m,y_2) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_1,y_n) & f(x_2,y_n) & \cdots & f(x_m,y_n) \end{bmatrix}$$

- 对y方向求偏导

$$\left[D_y^{(k)} f^T\right]^T = f\left[D_y^{(k)}\right]^T$$

- 混合求导

$$f^{(k,l)}(x,y) \Rightarrow D_x^{(k)} f\left[D_y^{(l)}\right]^T$$

Poisson方程

- 二维Poisson方程

$$\partial_x^2 f + \partial_y^2 f = s$$
$$f\Big|_{bdy} = 0$$

- 离散化

$$D_x^{(2)} f + f\left[D_y^{(2)}\right]^T \overset{?}{=} s \Rightarrow Mf = s$$

- 采用二维格点离散化无法将最终结果表示成常规线性系统的形式！
- 如何才能获得M的具体形式？

按行展开（矢量化）

$$\begin{bmatrix} f(x_1,y_1) & f(x_1,y_2) & \cdots & f(x_1,y_n) \\ f(x_2,y_1) & f(x_2,y_2) & \cdots & f(x_2,y_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_m,y_1) & f(x_m,y_2) & \cdots & f(x_m,y_n) \end{bmatrix} \xrightarrow{\text{Flatten}} \begin{bmatrix} f(x_1,y_1) \\ f(x_1,y_2) \\ \cdots \\ f(x_1,y_n) \\ f(x_2,y_1) \\ f(x_2,y_2) \\ \cdots \\ f(x_2,y_n) \\ \vdots \\ f(x_m,y_1) \\ f(x_m,y_2) \\ \cdots \\ f(x_m,y_n) \end{bmatrix}$$

微分矩阵的扩展

$$\underbrace{\begin{bmatrix} I_{x,11}D_y^{(k)} & & & \\ & I_{x,22}D_y^{(k)} & & \\ & & \ddots & \\ & & & I_{x,mm}D_y^{(k)} \end{bmatrix}}_{I_x \otimes D_y^{(k)}} \begin{bmatrix} f(x_1,y_1) \\ f(x_1,y_2) \\ \cdots \\ f(x_1,y_n) \\ --- \\ f(x_2,y_1) \\ f(x_2,y_2) \\ \cdots \\ f(x_2,y_n) \\ --- \\ \vdots \\ --- \\ f(x_m,y_1) \\ f(x_m,y_2) \\ \cdots \\ f(x_m,y_n) \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} D_{x,11}^{(k)}I_y & D_{x,12}^{(k)}I_y & \cdots & D_{x,1m}I_y \\ D_{x,21}^{(k)}I_y & D_{x,22}^{(k)}I_y & \cdots & D_{x,2m}^{(k)}I_y \\ \vdots & \vdots & \ddots & \vdots \\ D_{x,m1}^{(k)}I_y & D_{x,m2}^{(k)}I_y & \cdots & D_{x,mm}^{(k)}I_y \end{bmatrix}}_{D_x^{(k)} \otimes I_y} \begin{bmatrix} f(x_1,y_1) \\ f(x_1,y_2) \\ \cdots \\ f(x_1,y_n) \\ --- \\ f(x_2,y_1) \\ f(x_2,y_2) \\ \cdots \\ f(x_2,y_n) \\ --- \\ \vdots \\ --- \\ f(x_m,y_1) \\ f(x_m,y_2) \\ \cdots \\ f(x_m,y_n) \end{bmatrix}$$

Poisson方程

- 二维Poisson方程

$$\partial_x^2\psi + \partial_y^2\psi = s$$
$$\psi\Big|_{\text{bdy}} = 0$$

- 离散化

$$\left[D_x^{(k)} \otimes I_y\right]\Psi + \left[I_x \otimes D_y^{(k)}\right]\Psi = \underbrace{\left[D_x^{(k)} \otimes I_y + I_x \otimes D_y^{(k)}\right]}_{M}\Psi = S$$

- 边界条件替换

$$M(\text{IndexBdy}, :) = I(\text{IndexBdt}, :), S(\text{IndexBdy}, :) = 0$$
$$\Psi = M^{-1}S$$

三维离散化

- 张量缩并

对第一个指标缩并	对第二个指标缩并	对第三个指标缩并
$D_{ij}\psi_{jkl}$	$D_{ik}\psi_{\underbrace{kj}_{j\leftrightarrow k}l}$	$D_{il}\psi_{\underbrace{lkj}_{j\leftrightarrow l}}$ or $\psi_{jkl}D^T_{li}$

- 微分矩阵

$$D_x^{(k)} \otimes I_y \otimes I_z$$
$$I_x \otimes D_y^{(k)} \otimes I_z$$
$$I_x \otimes I_y \otimes D_z^{(k)}$$

- 先解决存储问题

$$m \times n \times l \times \text{方程数} \sim 10^5$$

迭代法

- 对于线性方程

$$\mathcal{L}f = b \Rightarrow \partial_t f = b - \mathcal{L}f$$

- 显然，方满足$\mathcal{L}f = b$时，时间演化停止，将右端离散化（时间导数采用单步向前欧拉法）

$$\boldsymbol{f}_{n+1} = \boldsymbol{f}_n + \tau(\boldsymbol{b} - L\boldsymbol{f}_n)$$

- 收敛分析

$$\begin{aligned}\boldsymbol{e}_{n+1} \equiv \boldsymbol{f} - \boldsymbol{f}_{n+1} &= \boldsymbol{f} - [\boldsymbol{f}_n + \tau(\boldsymbol{b} - L\boldsymbol{f}_n)] \\ &= \boldsymbol{f} - \boldsymbol{f}_n - \tau L(\boldsymbol{f} - \boldsymbol{f}_n) \\ &= \underbrace{(I - \tau L)}_{G}\boldsymbol{e_n} = G^2\boldsymbol{e}_{n-1} = G^n\boldsymbol{e}_1\end{aligned}$$

$$G = P^{-1}\Lambda P \Rightarrow G^n = P^{-1}\Lambda^n P$$

$$\rho \equiv \max\{|\Lambda|\}, \rho < 1 \Rightarrow L\text{正定}$$

预处理（preconditioning）

- 任何迭代法总要结合适当的预处理

$$Lx = b$$
$$(P^{-1}L)x = (P^{-1}b)$$

- 预处理的原则：
  - 预处理子P^{-1}较容易获得
  - 预处理后的矩阵条件数较小
  - 一般来说P是L的低阶近似
- 预处理后的迭代

$$x_{n+1} = x_n + \tau(P^{-1}b - P^{-1}Lx_n) = x_n + \tau \underbrace{P^{-1}\overbrace{(b - Lx_n)}^{r_n}}_{z_n}$$

$$G = I - \tau(P^{-1}L)$$

直线法（Method of Lines）

- 将空间导数离散化，时间方向导数采用步进的方式演化！
- 时间方向步进一般用Runge–Kutta（如经典RK4）
- 考虑初值问题

$$\frac{dy}{dt} = f(t, y), y(t_0) = y_0$$

- RK4的演化步骤（h为演化步长）

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \text{其中} \begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}h\right) \\ k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}h\right) \\ k_4 = f(t_n + h, y_n + hk_3) \end{cases}$$

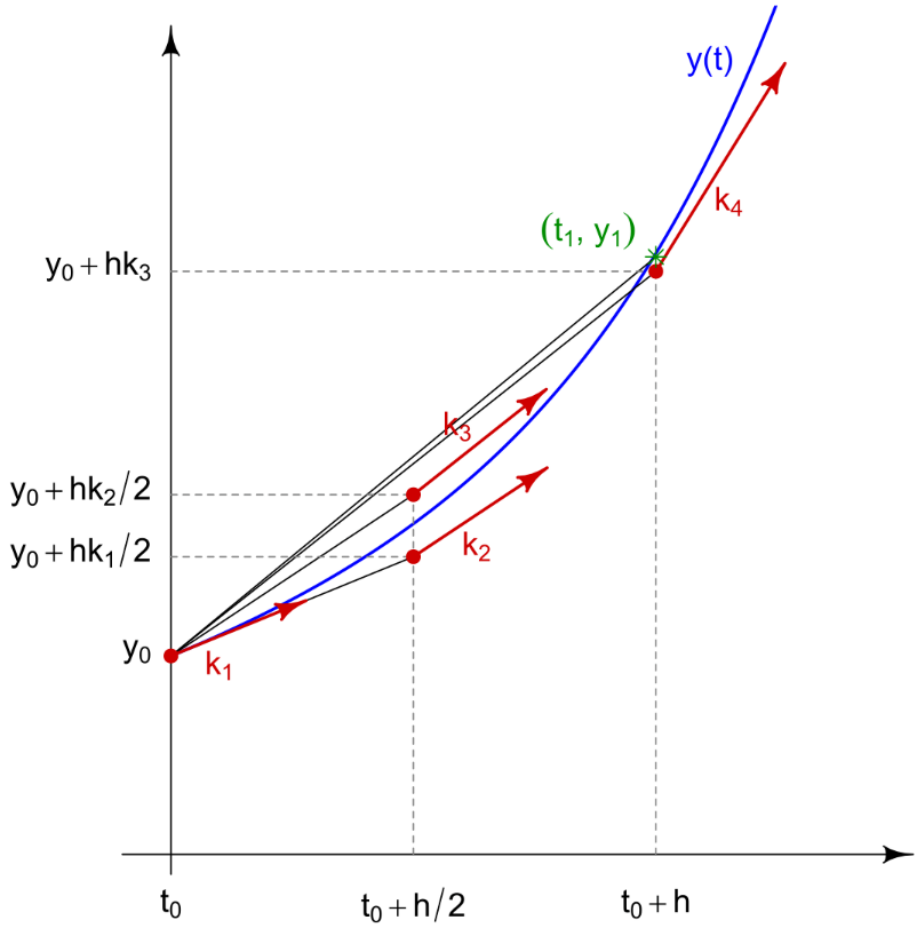
$$t_{n+1} = t_n + h$$

Runge-Kutta

- Runge-Kutta的几何解释
  - k_1是t_0处的斜率
  - k_2是t_0到t_1中点的斜率，采用斜率k_1来决定y在中点值
  - k_3也是中点处的斜率，但是这次采用斜率k_2决定y值
  - k_4是$t_1 = t_0 + h$处的斜率，其y值用k_3决定
- 当四个斜率取平均时，中点的斜率有更大的权值：

$$k = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

$$y_1 = y_0 + kh$$



GPU计算

- GPU 本身是一种并行架构，采用所谓的单程序多数据（SPMD）并行模式，特别适合做一些数据密集且宜并行的简单运算，如带批处理的矩阵乘法或者更一般的张量缩并、逐元（element-wise）数组运算等
- 对于较为复杂的操作，特别是涉及较多的分支（控制）时，GPU 的计算效率是相对较低的，原因在于 GPU通常不像 CPU 那样具有复杂的分支预测功能
- 市面上消费级别的显卡，典型的 NVIDIA GeForce 系列是不具备双精度加速计算的能力，这类显卡主要用于图形渲染，本身不需要较高的双精度计算
- 为了做科学计算需要购买专门的计算卡，如我们这里的 TITAN V 或者 NVIDIA Tesla 系列显卡。尤其后者是专门为科学计算而设计，其显存具备纠错码（ECC）机制，可以有效降低因环境噪声导致的误码率
- 需要提醒的是，在测量 GPU 执行计算时间时，要注意 CPU 与 GPU 的计算同步化（synchronize）

GPU计算

	Results for data-type 'double' (In GFLOPS)			Results for data-type 'single' (In GFLOPS)		
	MTimes	**Backslash**	**FFT**	**MTimes**	**Backslash**	**FFT**
Your GPU (TITAN V)	**6694.45**	**1259.51**	**551.51**	**13440.17**	**2438.90**	**935.72**
Tesla P100-PCIE-12GB	4518.23	878.97	313.43	8807.20	1439.15	676.20
Tesla K40c	1189.54	677.12	135.88	3187.76	1334.17	294.86
Tesla K20c	1004.06	641.42	106.09	2657.01	1230.28	235.20
TITAN Xp	422.47	371.37	207.24	11607.69	1426.76	763.56
GeForce GTX 1080	280.84	223.05	137.66	7707.01	399.37	424.60
Your CPU	**1089.10**	**245.14**	**10.94**	**2132.53**	**512.82**	**19.64**
Quadro K620	25.45	22.77	12.75	716.71	350.31	75.00
Quadro 600	19.71	17.55	7.62	117.99	88.64	38.58

非精确牛顿迭代（In-exact Newton Iteration）

- 牛顿迭代需要在每一步迭代求解线性方程组：

$$J_F(\boldsymbol{x}_n)(\boldsymbol{x}_{n+1}-\boldsymbol{x}_n)=-\boldsymbol{F}(\boldsymbol{x}_n)$$

$$\boldsymbol{x}_{n+1}=\boldsymbol{x}_n-J_F^{-1}(\boldsymbol{x}_n)\cdot\boldsymbol{F}(\boldsymbol{x}_n)$$

- 该线性方程组的求解，通常采用高斯消元法直接求解。实际上这种常规方法当矩阵规模 N 很大时，计算代价是相当大的，我们知道高斯消元的计算复杂度为$O(n^3)$

- 而对于牛顿迭代内部的线性方程组，实际上我们没必要求其严格解，只需最终的迭代解收敛到要求精度范围内即可。因此，最好的办法是用迭代法去求近似解（如GMRES），即所谓的非精确牛顿迭代 (Inexact Newton Iteration)

- Jacobi-free算法

$$J\boldsymbol{v}=\frac{F(\boldsymbol{x}+\sigma\boldsymbol{v})-F(\boldsymbol{x})}{\sigma}$$

准牛顿法（quasi-Newton）

- 另一种近似牛顿法是所谓的准牛顿法，该方法的核心是简化Jacobi矩阵的计算
- 用低阶法离散化Jacobi，高阶法离散化方程！

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - J_F^{-1}(\boldsymbol{x}_n)\cdot \boldsymbol{F}(\boldsymbol{x}_n) \xleftarrow[\text{步长取1}]{\text{一步向前欧拉法}} \frac{d\boldsymbol{x}}{dt} = -J_F^{-1}(\boldsymbol{x}_n)\cdot \boldsymbol{F}(\boldsymbol{x}_n)$$

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \tau J_F^{-1}(\boldsymbol{x}_n)\cdot \boldsymbol{F}(\boldsymbol{x}_n)$$

$$\underbrace{\overbrace{J(\boldsymbol{x}_n)}^{\text{Low Order}}}_{A}\underbrace{(\boldsymbol{x}_{n+1}-\boldsymbol{x}_n)}_{y} = \underbrace{-\tau\ \overbrace{F(\boldsymbol{x}_n)}^{\text{High Order}}}_{b}$$

- 最终的求解精度由高阶法决定！

域分解（ Domain decomposition ）与有限元（ finite element ）

- 有限差分一般只适用于形状规则的几何区域，实际的建模往往是较为复杂的几何
- 对于高阶方法（如伪谱法），当需要较大采点规模时，单域求解会使得最终的Jacobi矩阵较难处理
- 将区域分解为有限个部分，每一个部分称为一个element！用分片（piece-wise）多项式近似每个element中的未知函数
- 我国数学家冯康先生独立国外开创了有限元分析！