计算流体力学　第一次作业

张柏铭　3230100298

2025 年 2 月 25 日

1. 从流体力学专业期刊，例如 Journal of Computational Physics、Journal of Fluid Mechanics、Physics of Fluids、Physical Review Fluids、力学学报等找一篇自己感兴趣的 CFD 为主的论文(近 5 年内)，完整阅读，总结是用什么数值方法、解决了什么问题、有什么需要改进或者值得未来进一步研究的（这一条选做）。

Symplectic neural networks in Taylor series form for Hamiltonian systems

Yunjin Tong [a,1], Shiying Xiong [a,*,1], Xingzhe He [a,b], Guanghan Pan [a,c], Bo Zhu [a]

[a] Dartmouth College, Hanover, NH 03755, United States
[b] Rutgers University, New Brunswick, NJ 08854, United States
[c] Middlebury College, Middlebury, VT 05753, United States

1　数值方法[1]

本文提出了一种新型的神经网络架构——**Symplectic Taylor Neural Networks (Taylor-nets)**，用于预测哈密顿动力系统的长期动态行为。其核心方法如下：

1.1　泰勒级数展开形式的神经网络

Taylor-nets 通过构建两个子网络，分别学习哈密顿系统的位置和动量梯度。网络结构基于泰勒级数展开，确保每个项的对称性。具体来说，网络的设计如下：

- **对称网络结构**：为了学习哈密顿量关于广义坐标和动量的梯度，作者设计了对称的神经网络结构：

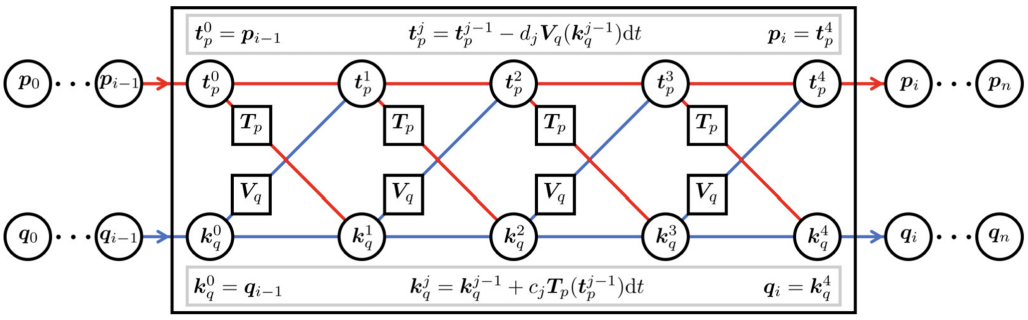
$$T_p(p,\theta_p)=\sum_{i=1}^{M}\left(A_i^T f_i(A_i p)-B_i^T f_i(B_i p)\right)+b, \tag{1}$$

其中 A_i, B_i 是全连接层的权重矩阵，$f_i(x)=\frac{x^i}{i!}$ 是泰勒级数的第 i 项，M 是泰勒级数的项数，b 是偏置项。这种结构确保了网络的雅可比矩阵是对称的，从而满足哈密顿系统的辛结构要求。

[1] https://doi.org/10.1016/j.jcp.2021.110325

- **辛结构保持**：通过设计对称的网络结构，Taylor-nets 保证了哈密顿系统的辛结构得以保持。辛结构是哈密顿系统的一个重要特性，它保证了系统的相空间体积守恒。

1.2 四阶辛积分器



为了实现哈密顿系统的连续时间演化，作者引入了四阶辛积分器。该积分器结合了神经常微分方程（Neural ODEs）框架，能够精确地模拟系统的长期动态行为。积分器的具体实现如下：

- **递归关系**：给定初始条件 (q_0, p_0)，通过递归关系计算系统的演化：

$$\begin{cases} q_i = q_{i-1} + c_j T_p(p_{i-1}, \theta_p)\Delta t, \\ p_i = p_{i-1} - d_j V_q(q_i, \theta_q)\Delta t, \end{cases} \tag{2}$$

  其中 c_j 和 d_j 是四阶辛积分器的系数。

- **辛结构保持**：四阶辛积分器确保了系统的辛结构在时间演化过程中得以保持，从而提高了模型的长期预测精度。

1.3 数据驱动的训练方法

Taylor-nets 采用数据驱动的方法进行训练，仅需系统的初始和最终状态数据，无需中间数据。具体训练方法如下：

- **训练数据生成**：通过辛积分器生成训练数据，仅使用系统的初始状态 (q_0, p_0) 和最终状态 (q_n, p_n)，而不依赖于中间状态。

- **损失函数**：采用 L_1 损失函数进行训练，损失函数定义为：

$$L_{\text{train}} = \frac{1}{N_{\text{train}}} \sum_{s=1}^{N_{\text{train}}} \left(\|\hat{p}_n^{(s)} - p_n^{(s)}\|_1 + \|\hat{q}_n^{(s)} - q_n^{(s)}\|_1 \right), \tag{3}$$

  其中 N_{train} 是训练样本的数量。

2 解决的问题

2.1 哈密顿系统的长期预测

传统方法在长期预测时容易出现误差累积，导致预测结果偏离真实轨迹。Taylor-nets 通过辛结构保持和精确的时间演化，显著提高了长期预测的准确性。例如，在摆动系统中，训练周期 $T_{\text{train}} = 0.01$，预测周期 $T_{\text{predict}} = 20\pi$，模型仍能准确预测系统的动态行为。

2.2 数据稀疏性问题

在实际物理系统中，获取大量数据往往成本高昂。Taylor-nets 仅需少量数据（如15个样本）即可实现高精度预测，且训练周期远短于预测周期（6000倍以上）。例如，在摆动系统中，仅需15个样本即可实现 $L_{\text{validation}} \sim 10^{-4}$ 的验证损失。

2.3 复杂系统的建模

该方法能够处理包括摆动系统、Lotka-Volterra系统、开普勒系统和Hénon-Heiles系统在内的多种哈密顿动力系统，表现出良好的泛化能力。

3 改进方向

尽管 Taylor-nets 在哈密顿系统预测中表现出色，但仍有一些潜在的改进方向和值得进一步研究的领域：

3.1 非可分离哈密顿系统的建模

当前方法主要针对可分离哈密顿系统（即动能和势能可分离的系统）。对于非可分离哈密顿系统，需要进一步探索如何设计网络架构以适应其复杂性。

NONSEPARABLE SYMPLECTIC NEURAL NETWORKS

Shiying Xiong[a*], **Yunjin Tong**[a], **Xingzhe He**[a], **Shuqi Yang**[a], **Cheng Yang**[b], **Bo Zhu**[a]
a Dartmouth College, Hanover, NH, United States
b ByteDance AI Lab, Beijing, China

3.2 更高维度系统的扩展

虽然 Taylor-nets 已经在多体问题中展示了良好的泛化能力，但在处理更高维度的复杂系统时，可能需要优化网络结构或引入更高效的辛积分器。

3.3 计算效率的优化

尽管 Taylor-nets 的训练效率较高，但随着系统复杂度的增加，计算成本仍可能成为瓶颈。探索更高效的训练算法或硬件加速技术，将有助于进一步提升模型的实用性。或者引入量子计算方法，使其可以应用到量子模拟中。

4 未来研究

本来是打算拓展到扩散模型中，当时阅读了熊老师的四篇文献，并做了相应的总结和构想。见下PDF整理。

Reviews of different networks

Zhang Baiming Time: 2025-1-23

Fundamental Work: HNN

Hamiltonian Neural Networks

Sam Greydanus
Google Brain
sgrey@google.com

Misko Dzamba
PetCube
mouse9911@gmail.com

Jason Yosinski
Uber AI Labs
yosinski@uber.com

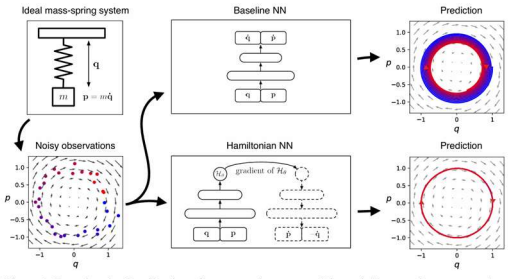


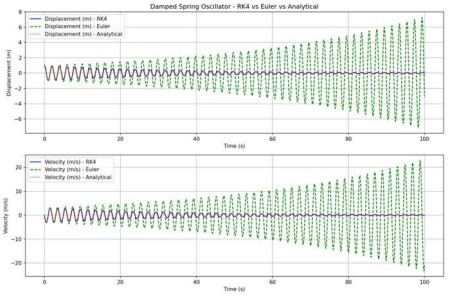
Figure 1: Learning the Hamiltonian of a mass-spring system. The variables q and p correspond to position and momentum coordinates. As there is no friction, the baseline's inner spiral is due to model errors. By comparison, the Hamiltonian Neural Network learns to *exactly* conserve a quantity that is analogous to total energy.

$$\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}.$$

$$(\mathbf{q}_1, \mathbf{p}_1) = (\mathbf{q}_0, \mathbf{p}_0) + \int_{t_0}^{t_1} \mathbf{S}(\mathbf{q}, \mathbf{p})\ dt$$

$$\mathbf{S}_{\mathcal{H}} = \left(\frac{\partial \mathcal{H}}{\partial \mathbf{p}}, -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}\right)$$

$$\mathcal{L}_{HNN} = \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} - \frac{\partial \mathbf{q}}{\partial t} \right\|_2 + \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} + \frac{\partial \mathbf{p}}{\partial t} \right\|_2$$



Four Nets from Shiying Xiong

2021.4	**TaylorNet**	$\boldsymbol{T}_p(\boldsymbol{p},\boldsymbol{\theta}_p)=\left(\sum_{i=1}^{M}\boldsymbol{A}_i^T\circ f_i\circ\boldsymbol{A}_i-\boldsymbol{B}_i^T\circ f_i\circ\boldsymbol{B}_i\right)\circ\boldsymbol{p}+\boldsymbol{b}$ $\quad f_i(x)=\frac{1}{i!}x^i$
2021.5	**NSSNN**	$(q_i,p_i,x_i,y_i)=\phi_1^{\mathrm{d}t/2}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_3^{\mathrm{d}t}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_1^{\mathrm{d}t/2}\circ(q_{i-1},p_{i-1},x_{i-1},y_{i-1})$
2023.2	**VortexNet**	$\begin{cases}\frac{\mathrm{d}\boldsymbol{\Gamma}_i}{\mathrm{d}t}=\boldsymbol{\gamma}_i,\\ \frac{\mathrm{d}\boldsymbol{X}_i}{\mathrm{d}t}=\boldsymbol{u}_i+\boldsymbol{v}_i.\end{cases}$ $\quad \boldsymbol{u}_i=\frac{1}{2(n_d-1)\pi}\sum_{j\neq i}^{N}\frac{\boldsymbol{\Gamma}_j\times(\boldsymbol{X}_i-\boldsymbol{X}_j)}{\lvert\boldsymbol{X}_i-\boldsymbol{X}_j\rvert^{n_d}+\mathcal{R}^{n_d}}$
2023.4	**RoeNet**	$\boldsymbol{u}_j^{n+1}=\boldsymbol{u}_j^n-\lambda_r\left(\hat{\boldsymbol{F}}_{j+\frac{1}{2}}^n-\hat{\boldsymbol{F}}_{j-\frac{1}{2}}^n\right)$ $\quad \hat{\boldsymbol{F}}_{j+\frac{1}{2}}^n=\hat{\boldsymbol{F}}(\boldsymbol{u}_j^n,\boldsymbol{u}_{j+1}^n)$ $\quad \hat{\boldsymbol{F}}(\boldsymbol{u},\boldsymbol{v})=\frac{1}{2}[\boldsymbol{F}(\boldsymbol{u})+\boldsymbol{F}(\boldsymbol{v})-\lvert\tilde{\boldsymbol{A}}(\boldsymbol{u},\boldsymbol{v})\rvert(\boldsymbol{v}-\boldsymbol{u})]$

2021.4

Symplectic neural networks in Taylor series form for Hamiltonian systems

2 April 2021

Yunjin Tong [a,1], Shiying Xiong [a,*,1], Xingzhe He [a,b], Guanghan Pan [a,c], Bo Zhu [a]

$$\boldsymbol{T}_p(\boldsymbol{p},\boldsymbol{\theta}_p)=\left(\sum_{i=1}^{M}\boldsymbol{A}_i^T\circ f_i\circ\boldsymbol{A}_i-\boldsymbol{B}_i^T\circ f_i\circ\boldsymbol{B}_i\right)\circ\boldsymbol{p}+\boldsymbol{b}$$

$$\boldsymbol{V}_q(\boldsymbol{q},\boldsymbol{\theta}_q)=\left(\sum_{i=1}^{M}\boldsymbol{C}_i^T\circ f_i\circ\boldsymbol{C}_i-\boldsymbol{D}_i^T\circ f_i\circ\boldsymbol{D}_i\right)\circ\boldsymbol{q}+\boldsymbol{d}$$

$$\boldsymbol{F}_t^j(\boldsymbol{p},\boldsymbol{q},\mathrm{d}t)=\left(\boldsymbol{p},\boldsymbol{q}+c_j\boldsymbol{T}_p(\boldsymbol{p},\boldsymbol{\theta}_p)\mathrm{d}t\right),$$

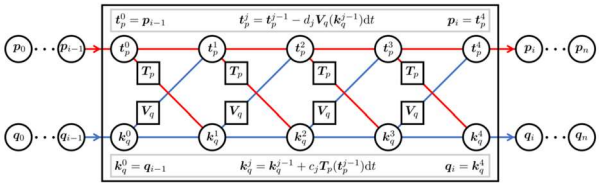
and $\quad f_i(x)=\frac{1}{i!}x^i$

$$\boldsymbol{F}_k^j(\boldsymbol{p},\boldsymbol{q},\mathrm{d}t)=\left(\boldsymbol{p}-d_j\boldsymbol{V}_q(\boldsymbol{q},\boldsymbol{\theta}_q)\mathrm{d}t,\boldsymbol{q}\right),$$

with

$$c_1=c_4=\frac{1}{2(2-2^{1/3})},\quad c_2=c_3=\frac{1-2^{1/3}}{2(2-2^{1/3})},$$

$$d_1=d_3=\frac{1}{2-2^{1/3}},\quad d_2=-\frac{2^{1/3}}{2-2^{1/3}},\quad d_4=0$$



Algorithm 1 Integrate (8) by using the fourth-order symplectic integrator.

Input: $\boldsymbol{q}_0,\boldsymbol{p}_0,t_0,t,\Delta t$,
$\boldsymbol{F}_t^j$ in (20) and $\boldsymbol{F}_k^j$ in (21) with $j=1,2,3,4$;
Output: $\boldsymbol{q}(t),\boldsymbol{p}(t)$
$n=\mathrm{floor}[(t-t_0)/\Delta t]$;
for $i=1,n$
$(\boldsymbol{k}_p^0,\boldsymbol{k}_q^0)=(\boldsymbol{p}_{i-1},\boldsymbol{q}_{i-1})$;
for $j=1,4$
$(\boldsymbol{t}_p^{j-1},\boldsymbol{t}_q^{j-1})=\boldsymbol{F}_t^j(\boldsymbol{k}_p^{j-1},\boldsymbol{k}_q^{j-1},\Delta t)$,
$(\boldsymbol{k}_p^j,\boldsymbol{k}_q^j)=\boldsymbol{F}_k^j(\boldsymbol{t}_p^{j-1},\boldsymbol{t}_q^{j-1},\Delta t)$,
end
$(\boldsymbol{p}_i,\boldsymbol{q}_i)=(\boldsymbol{k}_p^4,\boldsymbol{k}_q^4)$;
end
$\boldsymbol{q}(t)=\boldsymbol{q}_n,\boldsymbol{p}(t)=\boldsymbol{p}_n$.

2021.5

Symplectic neural networks in Taylor series form for Hamiltonian systems

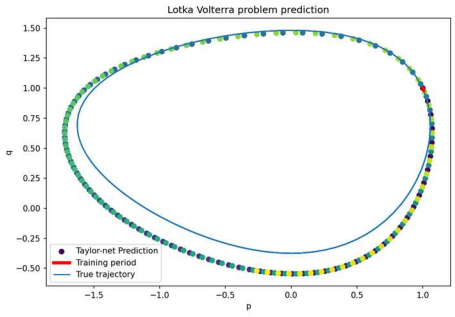
2 April 2021

Yunjin Tong [a,1], Shiying Xiong [a,*,1], Xingzhe He [a,b], Guanghan Pan [a,c], Bo Zhu [a]

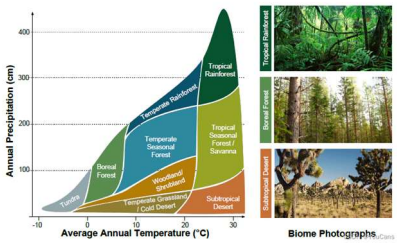
Lotka–Volterra system. For a Lotka–Volterra system, its Hamiltonian is given by

$$\mathcal{H}(q, p) = p - e^p + 2q - e^q.$$

Similarly, we pick a random initial point for training $(\boldsymbol{q}_0, \boldsymbol{p}_0) \in [-2, 2] \times [-2, 2]$.



2023/2024年美国大学生数学建模大赛A题



早在1942年，统计学家 Charles Elton 和 Mary Nicholson 发现，每年捕获的雪兔数量和猞猁数量呈现出十分规则的周期变化。在自然界中，雪兔和猞猁互为猎物和猎食者，它们的种群数量的这种变化规律引发了不少研究学者的关注。

2023.2

NONSEPARABLE SYMPLECTIC NEURAL NETWORKS

Shiying Xiong[a*], **Yunjin Tong**[a], **Xingzhe He**[a], **Shuqi Yang**[a], **Cheng Yang**[b], **Bo Zhu**[a]

Explicit symplectic approximation of nonseparable Hamiltonians: Algorithm and long time performance

Molei Tao*

$$\overline{\mathcal{H}}(\boldsymbol{q},\boldsymbol{p},\boldsymbol{x},\boldsymbol{y}) := \mathcal{H}_A + \mathcal{H}_B + \omega\mathcal{H}_C$$

$$\mathcal{H}_A = \mathcal{H}(\boldsymbol{q},\boldsymbol{y}),\ \mathcal{H}_B = \mathcal{H}(\boldsymbol{x},\boldsymbol{p}),\ \mathcal{H}_C = \frac{1}{2}\left(\|\boldsymbol{q}-\boldsymbol{x}\|_2^2 + \|\boldsymbol{p}-\boldsymbol{y}\|_2^2\right)$$

$$\begin{aligned}\dot{q} &= \partial_p H(x,p) + \omega(p-y)\\ \dot{p} &= -\partial_q H(q,y) - \omega(q-x)\\ \dot{x} &= \partial_y H(q,y) + \omega(y-p)\\ \dot{y} &= -\partial_x H(x,p) - \omega(x-q)\end{aligned}$$

$$(q_i,p_i,x_i,y_i) = \phi_1^{\mathrm{d}t/2}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_3^{\mathrm{d}t}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_1^{\mathrm{d}t/2}\circ(q_{i-1},p_{i-1},x_{i-1},y_{i-1})$$

$$\boldsymbol{R}^\delta := \begin{bmatrix}\cos(2\omega\delta)\boldsymbol{I} & \sin(2\omega\delta)\boldsymbol{I}\\ -\sin(2\omega\delta)\boldsymbol{I} & \cos(2\omega\delta)\boldsymbol{I}\end{bmatrix}$$

$$\begin{bmatrix}\boldsymbol{q}\\ \boldsymbol{p}-\delta[\partial\mathcal{H}_\theta(\boldsymbol{q},\boldsymbol{y})/\partial\boldsymbol{q}]\\ \boldsymbol{x}+\delta[\partial\mathcal{H}_\theta(\boldsymbol{q},\boldsymbol{y})/\partial\boldsymbol{p}]\\ \boldsymbol{y}\end{bmatrix},\ \begin{bmatrix}\boldsymbol{q}+\delta[\partial\mathcal{H}_\theta(\boldsymbol{x},\boldsymbol{p})/\partial\boldsymbol{p}]\\ \boldsymbol{p}\\ \boldsymbol{x}\\ \boldsymbol{y}-\delta[\partial\mathcal{H}_\theta(\boldsymbol{x},\boldsymbol{p})/\partial\boldsymbol{q}]\end{bmatrix},\ \text{and}\ \frac{1}{2}\begin{bmatrix}\begin{pmatrix}\boldsymbol{q}+\boldsymbol{x}\\ \boldsymbol{p}+\boldsymbol{y}\end{pmatrix}+\boldsymbol{R}^\delta\begin{pmatrix}\boldsymbol{q}-\boldsymbol{x}\\ \boldsymbol{p}-\boldsymbol{y}\end{pmatrix}\\ \begin{pmatrix}\boldsymbol{q}+\boldsymbol{x}\\ \boldsymbol{p}+\boldsymbol{y}\end{pmatrix}-\boldsymbol{R}^\delta\begin{pmatrix}\boldsymbol{q}-\boldsymbol{x}\\ \boldsymbol{p}-\boldsymbol{y}\end{pmatrix}\end{bmatrix}$$

Algorithm 1 Integrate (4) by using the second-order symplectic integrator

Input: $\boldsymbol{q}_0, \boldsymbol{p}_0, t_0, t, \mathrm{d}t$; ϕ_1^δ, ϕ_2^δ, and ϕ_3^δ in (5);
Output: $(\hat{\boldsymbol{q}}, \hat{\boldsymbol{p}}, \hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = (\boldsymbol{q}_n, \boldsymbol{p}_n, \boldsymbol{x}_n, \boldsymbol{y}_n)$
$(\boldsymbol{q}_0, \boldsymbol{p}_0, \boldsymbol{x}_0, \boldsymbol{y}_0) = (\boldsymbol{q}_0, \boldsymbol{p}_0, \boldsymbol{q}_0, \boldsymbol{p}_0)$ $n = \mathrm{floor}[(t-t_0)/\mathrm{d}t]$ **for** $i = 1 \to n$ **do**
  $(\boldsymbol{q}_i, \boldsymbol{p}_i, \boldsymbol{x}_i, \boldsymbol{y}_i) = \phi_1^{\mathrm{d}t/2}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_3^{\mathrm{d}t}\circ\phi_2^{\mathrm{d}t/2}\circ\phi_1^{\mathrm{d}t/2}\circ(\boldsymbol{q}_{i-1}, \boldsymbol{p}_{i-1}, \boldsymbol{x}_{i-1}, \boldsymbol{y}_{i-1})$;
end

Methods	NSSNN	HNN	NeuralODE	TaylorNet
Solve nonseparable systems	✓	✓	✓	
Solve separable systems	✓	✓	✓	✓
Preserve symplectic structure	✓	Partially		✓
Utilize continuous dynamics	✓		✓	✓
No need for derivatives in dataset	✓		✓	✓
Long-term predictability	✓	Partially		✓
Extend to N-body system	✓	✓	✓	✓

2023.2

Neural vortex method: From finite Lagrangian particles to infinite dimensional Eulerian dynamics

Shiying Xiong [a,b,*], Xingzhe He [b], Yunjin Tong [b], Yitong Deng [b], Bo Zhu [b]

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{\Gamma}_i}{\mathrm{d}t} = \boldsymbol{\gamma}_i, \\ \frac{\mathrm{d}\boldsymbol{X}_i}{\mathrm{d}t} = \boldsymbol{u}_i + \boldsymbol{v}_i. \end{cases} \qquad \boldsymbol{u}_i = \frac{1}{2(n_d-1)\pi} \sum_{j \neq i}^{N} \frac{\boldsymbol{\Gamma}_j \times (\boldsymbol{X}_i - \boldsymbol{X}_j)}{|\boldsymbol{X}_i - \boldsymbol{X}_j|^{n_d} + \mathcal{R}^{n_d}}$$

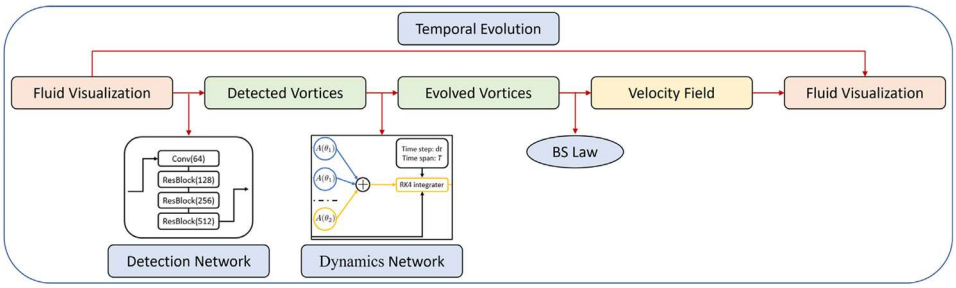


Fig. 1. Schematic diagram of the NVM. Our system is constituted of two networks, the detection network and the dynamics network, which are embedded with a vorticity-to-velocity Poisson solver.

2023.2

Neural vortex method: From finite Lagrangian particles to infinite dimensional Eulerian dynamics

Shiying Xiong [a,b,*], Xingzhe He [b], Yunjin Tong [b], Yitong Deng [b], Bo Zhu [b]

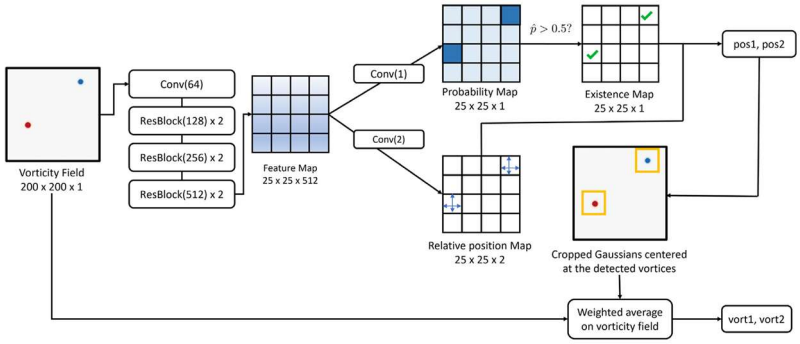


Fig. 2. The architecture of the detection network. It takes the vorticity field as input and outputs the position and vortex particle strength for each vortex detected. The *Conv* means the Conv2d-BatchNorm-ReLU combo, and the *ResBlock* is the same as in He et al. [51]. In each *ResBlock*, we use stride 2 to downsample the feature map. The Resblock chain is six-layer structured. The number in the parenthesis is the output dimension.

2023.2

Neural vortex method: From finite Lagrangian particles to infinite dimensional Eulerian dynamics

Shiying Xiong [a,b,*], Xingzhe He [b], Yunjin Tong [b], Yitong Deng [b], Bo Zhu [b]

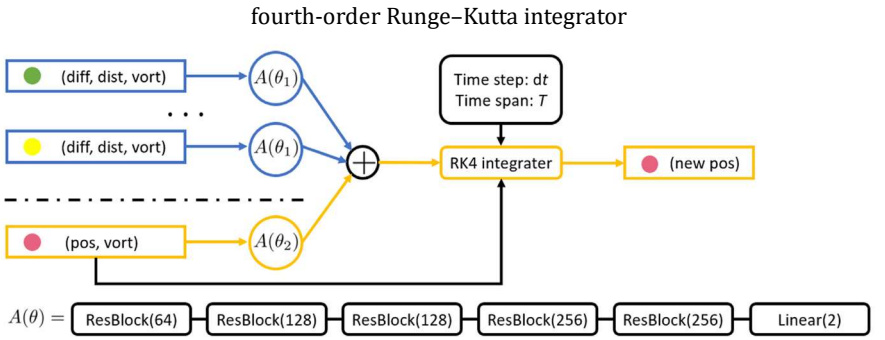


Fig. 4. The architecture of the dynamics network. It takes the particle's attribution as input and outputs each vortex's position. The *ResBlock* has the same architecture as in He et al. [51] with the convolution layers replaced by linear layers. The number in the parenthesis is the output dimension.

2023.4

RoeNet: Predicting discontinuity of hyperbolic systems from continuous data

Yunjin Tong[1] | Shiying Xiong[1,2] | Xingzhe He[1] | Shuqi Yang[1] | Zhecheng Wang[1] | Rui Tao[1] | Runze Liu[1] | Bo Zhu[1]

$$\frac{\partial \boldsymbol{u}}{\partial t} + \sum_{i=1}^{N_d} \frac{\partial \boldsymbol{F}_i(\boldsymbol{u})}{\partial x_i} = \boldsymbol{0}$$

$$\boldsymbol{u}_j^{n+1} = \boldsymbol{u}_j^n - \lambda_r \left(\hat{\boldsymbol{F}}_{j+\frac{1}{2}}^n - \hat{\boldsymbol{F}}_{j-\frac{1}{2}}^n \right)$$

$$\hat{\boldsymbol{F}}_{j+\frac{1}{2}}^n = \hat{\boldsymbol{F}}(\boldsymbol{u}_j^n, \boldsymbol{u}_{j+1}^n)$$

$$\hat{\boldsymbol{F}}(\boldsymbol{u}, \boldsymbol{v}) = \frac{1}{2}\left[\boldsymbol{F}(\boldsymbol{u}) + \boldsymbol{F}(\boldsymbol{v}) - |\tilde{\boldsymbol{A}}(\boldsymbol{u}, \boldsymbol{v})|(\boldsymbol{v} - \boldsymbol{u})\right]$$

$$\lim_{\boldsymbol{u}_j, \boldsymbol{u}_{j+1} \to \boldsymbol{u}} \tilde{\boldsymbol{A}}(\boldsymbol{u}_j, \boldsymbol{u}_{j+1}) = \frac{\partial \boldsymbol{F}(\boldsymbol{u})}{\partial \boldsymbol{u}}$$

$$\boldsymbol{F}_{j+1} - \boldsymbol{F}_j = \tilde{\boldsymbol{A}}(\boldsymbol{u}_{j+1} - \boldsymbol{u}_j)$$

$$|\tilde{\boldsymbol{A}}| = \boldsymbol{L}^{-1}|\boldsymbol{\Lambda}|\boldsymbol{L}$$

$$\boldsymbol{u}_j^{n+1} = \boldsymbol{u}_j^n - \frac{1}{2}\lambda_r[(\boldsymbol{L}_{j+\frac{1}{2}}^n)^{-1}(\boldsymbol{\Lambda}_{j+\frac{1}{2}}^n - |\boldsymbol{\Lambda}_{j+\frac{1}{2}}^n|)\boldsymbol{L}_{j+\frac{1}{2}}^n(\boldsymbol{u}_{j+1}^n - \boldsymbol{u}_j^n) + (\boldsymbol{L}_{j-\frac{1}{2}}^n)^{-1}(\boldsymbol{\Lambda}_{j-\frac{1}{2}}^n + |\boldsymbol{\Lambda}_{j-\frac{1}{2}}^n|)\boldsymbol{L}_{j-\frac{1}{2}}^n(\boldsymbol{u}_j^n - \boldsymbol{u}_{j-1}^n)]$$

$$\boldsymbol{L}_{j+\frac{1}{2}}^n = \boldsymbol{L}(\boldsymbol{u}_j^n, \boldsymbol{u}_{j+1}^n), \quad \boldsymbol{\Lambda}_{j+\frac{1}{2}}^n = \boldsymbol{\Lambda}(\boldsymbol{u}_j^n, \boldsymbol{u}_{j+1}^n)$$

2023.4

RoeNet: Predicting discontinuity of hyperbolic systems from continuous data

Yunjin Tong[1] | Shiying Xiong[1,2] | Xingzhe He[1] | Shuqi Yang[1] | Zhecheng Wang[1] | Rui Tao[1] | Runze Liu[1] | Bo Zhu[1]

Roe求解器是一种用于求解双曲型偏微分方程的数值方法，特别是在流体力学中。它由Philip L. Roe提出，基于Godunov方案，用于在离散的时空计算域中，对两个相邻计算单元格界面上的数值通量进行估计。

Algorithm 1. Recursive relation from the input layer to the output layer in RoeNet. Here, $\boldsymbol{u}_j,\ j = 1, 2, \ldots, N_g$ represents discretized points $\boldsymbol{u}$ in spatial coordinate

Input: $\boldsymbol{u}_j(t=0), j = 1, 2, \ldots, N_g$, T_{span}, Δt, Δx, $\boldsymbol{L}_\theta$, $\boldsymbol{\Lambda}_\phi$
Output: $\hat{\boldsymbol{u}}_j(t = T_{span}) = \boldsymbol{u}_j^{N_t}$
$N_t = \text{floor}(T_{span}/\Delta t)$ $\lambda_r = \Delta t/\Delta x$ $\boldsymbol{u}_j^0 = \boldsymbol{u}_j(t=0), j = 1, 2, \ldots, N_g$ **for** $n = 0 \to N_{t-1}$ **do**
  Calculate $\boldsymbol{L}^n_{j\pm\frac{1}{2},\theta}$, $\boldsymbol{\Lambda}^n_{j\pm\frac{1}{2},\phi}$ by substituting $\boldsymbol{u}_j^n, j = 1, 2, \ldots, N_g$, $\boldsymbol{L}_\theta$, and $\boldsymbol{\Lambda}_\phi$ into (17) Calculate $\boldsymbol{u}_j^{n+1}, j = 1, 2, \ldots, N_g$ by substituting $\boldsymbol{u}_j^n, \boldsymbol{L}^n_{j\pm\frac{1}{2},\theta}, \boldsymbol{\Lambda}^n_{j\pm\frac{1}{2},\phi}$, and λ_r into (16)
end

2023.8

A generalized framework of neural networks for Hamiltonian systems

Philipp Horn [a,*], Veronica Saz Ulibarrena [b], Barry Koren [a], Simon Portegies Zwart [b]

[a] *Centre for Analysis, Scientific Computing and Applications, Eindhoven University of Technology, the Netherlands*
[b] *Leiden Observatory, Leiden University, the Netherlands*

$$\begin{pmatrix}\dot{\boldsymbol{p}}\\ \dot{\boldsymbol{q}}\end{pmatrix} = -J\nabla H(\boldsymbol{p},\boldsymbol{q}),\quad J = \begin{pmatrix}0 & I_d\\ -I_d & 0\end{pmatrix},\quad \boldsymbol{p},\boldsymbol{q}\in\mathbb{R}^d \qquad \left(\frac{\partial\varphi_h}{\partial\boldsymbol{x}}\right)^T J\,\frac{\partial\varphi_h}{\partial\boldsymbol{x}} = J,\quad \boldsymbol{x} = \begin{pmatrix}\boldsymbol{p}\\ \boldsymbol{q}\end{pmatrix}\in\mathbb{R}^{2d}.$$

$$L_{\text{HNN}} = \frac{1}{N}\sum_{i=1}^{N}\left\|\begin{pmatrix}\dot{\boldsymbol{p}}_i\\ \dot{\boldsymbol{q}}_i\end{pmatrix} + J\nabla H_\theta(\boldsymbol{p}_i,\boldsymbol{q}_i)\right\|_2^2. \qquad H_\theta(\boldsymbol{p},\boldsymbol{q}) = T_{\theta_1}(\boldsymbol{p}) + U_{\theta_2}(\boldsymbol{q})$$

	SRNNs	SympNets ($2n$ gradient modules)	HénonNets (n Hénon layers)
Definition of the input to output mapping	$\text{SRNN}(\boldsymbol{p},\boldsymbol{q}) = \text{SI}(H_{\text{NN}},\boldsymbol{p},\boldsymbol{q})$	$\text{SympNet}(\boldsymbol{p},\boldsymbol{q}) = \text{SE}(H_{\text{NN}_n},\cdot,\cdot)\circ\cdots\circ\text{SE}(H_{\text{NN}_1},\boldsymbol{p},\boldsymbol{q})$	$\text{HénonNet}(\boldsymbol{p},\boldsymbol{q}) = \text{SE}(H_{\text{NN}_{2n}},\cdot,\cdot)\circ\cdots\circ\text{SE}(H_{\text{NN}_1},\boldsymbol{p},\boldsymbol{q})$
Details of the learned Hamiltonians	$H_{\text{NN}}(\boldsymbol{p},\boldsymbol{q}) = T_{\text{NN}}(\boldsymbol{p}) + U_{\text{NN}}(\boldsymbol{q})$	$H_{\text{NN}_i}(\boldsymbol{p},\boldsymbol{q}) = T_{\text{NN}_i}(\boldsymbol{p}) + U_{\text{NN}_i}(\boldsymbol{q})$	$H_{\text{NN}_{2i-1}}(\boldsymbol{p},\boldsymbol{q}) = V_{\text{NN}_i}(-\boldsymbol{p}) + V_{\text{NN}_i}(\boldsymbol{q})$ and $H_{\text{NN}_{2i}}(\boldsymbol{p},\boldsymbol{q}) = V_{\text{NN}_i}(-\boldsymbol{p}-\boldsymbol{\eta}) + V_{\text{NN}_i}(\boldsymbol{q}-\boldsymbol{\eta})$
Parameterization of the learned Hamiltonians	T_{NN} and U_{NN} can be any multilayer perceptrons.	T_{NN_i} and U_{NN_i} are multilayer perceptrons with one hidden layer.	V_{NN_i} can be any multilayer perceptron.
The integrators that can be used	SI can be any symplectic integrator that is explicit for separable Hamiltonians.	SE is the Symplectic Euler integrator.	SE is the Symplectic Euler integrator.

My Opinion

$$H = Du\nabla\dot{u} - v\nabla v$$

$$\begin{cases} \dot{u} = \dfrac{\partial H}{\partial v} = -\nabla\dot{v} \\ \dot{v} = -\dfrac{\partial H}{\partial u} = -D\nabla u \end{cases}$$

$$x = \begin{pmatrix} u \\ v \end{pmatrix}$$

A Wrong Thought!

$$\dot{x} = \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} -\nabla\dot{v} \\ -D\nabla u \end{pmatrix} = \begin{pmatrix} 0 & -\nabla\dfrac{d}{dt} \\ -D\nabla & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \mathrm{Jx}$$

Conventional Symplectic Neural Network

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \mathbf{J}\nabla H(x), \quad \text{with} \quad \mathbf{J} = \begin{pmatrix} 0 & \mathbf{I}_d \\ -\mathbf{I}_d & 0 \end{pmatrix}$$

$$\mathcal{L}_1(\bar{\psi}) = \frac{1}{N}\sum_{i=1}^{N}\left\| \frac{\mathrm{d}}{\mathrm{d}t}\bar{\psi}(t, x_0^i)\Big|_{t=t_i} - \mathbf{J}\nabla H\left(\bar{\psi}(t_i, x_0^i)\right)\right\|_2^2$$

$$\widetilde{\mathcal{L}}_2(\bar{\psi}) = \frac{1}{M}\sum_{i=1}^{m}\left(H(\bar{\psi}(t^i, x^i)) - H(x^i)\right)^2$$

$$\mathcal{L}(\bar{\psi}) = \mathcal{L}_1(\bar{\psi}) + \mathcal{L}_2(\bar{\psi})$$

Diffusional Symplectic Neural Network

$$\begin{cases} p = m\dot{q} \\ \Delta u = \xi\dot{u} \end{cases}$$

Where $\xi = \frac{1}{D}$, Which D means Diffusion Coefficient

My naive assumption:

$$\begin{cases} p \leftrightarrow \Delta u \\ q \leftrightarrow u \end{cases}$$

Assuming separatable: $H = T(\Delta u) + V(u)$

Neglecting pressure and viscosity, then we got:

$$H = \frac{(\Delta u)^2}{2\xi} + R(u)$$ where R(u) is the reaction term

My Opinion

Diffusional Symplectic Neural Network

$$\begin{cases} p = m\dot{q} \\ \Delta u = \xi\dot{u} \end{cases}$$

Where $\xi = \frac{1}{D}$, Which D means Diffusion Coefficient

My naive assumption:

$$\begin{cases} p \leftrightarrow \Delta u \\ q \leftrightarrow u \end{cases}$$

Assuming separatable: $H = T(\Delta u) + V(u)$

Neglecting pressure and viscosity, then we got:

$$H = \frac{(\Delta u)^2}{2\xi} + R(u)$$ where R(u) is the reaction term

For example, in Fisher-KPP Model: $R(u) = \rho u(1-u)$

This reaction term R(u) in different models can describe various phenomena such as gene propagation, biological population growth, and chemical reactions.

Loss Function

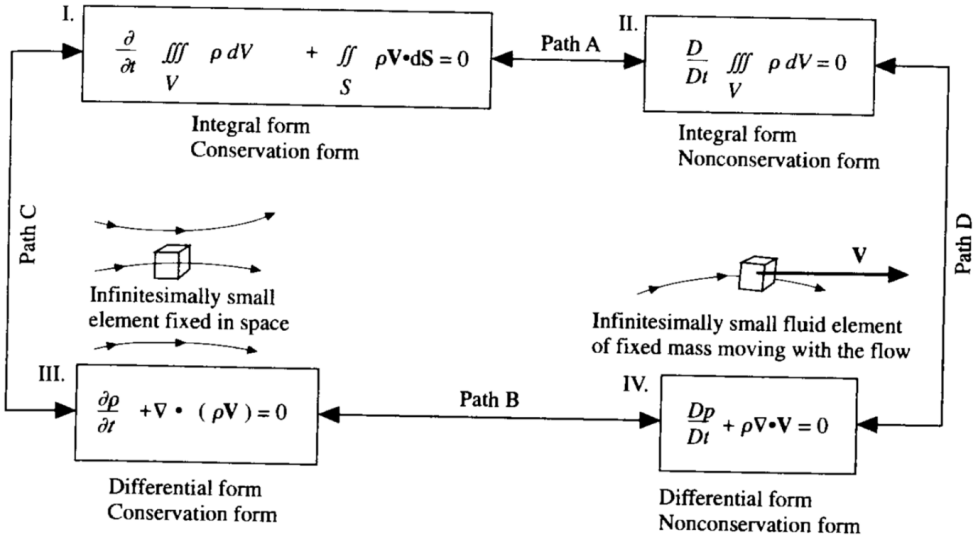
$$\mathcal{L}_{dynamic}(\bar{\psi}) = \frac{1}{N}\sum_{i=1}^{N}\left\| \frac{d}{dt}\bar{\psi}(t, x_0^i)\Big|_{t=t_i} - (\Delta\dot{u}, \xi\dot{u})\right\|_2^2$$

$$\mathcal{L}_{state}(\bar{\psi}) = \frac{1}{M}\sum_{i=1}^{M}\left(H(\bar{\psi}(t^i, x^i)) - H(x^i)\right)^2$$

$$\mathcal{L}(\bar{\psi}) = \lambda_{dynamic}\mathcal{L}_{dynamic}(\bar{\psi}) + \lambda_{state}\mathcal{L}_{state}(\bar{\psi})$$

$$\mathcal{L}(\bar{\psi}) = \frac{\lambda_{dynamic}}{N}\sum_{i=1}^{N}\left\| \frac{d}{dt}\bar{\psi}(t, x_0^i)\Big|_{t=t_i} - (\Delta\dot{u}, \xi\dot{u})\right\|_2^2 + \frac{\lambda_{state}}{M}\sum_{i=1}^{M}\left(H(\bar{\psi}(t^i, x^i)) - H(x^i)\right)^2$$

2. 写出下图中实现 Path B、Path D 的推导过程。



推导过程

Part B. III 到 IV 的互相推导

从 III 到 IV 的推导：

已知 III 的公式为：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0$$

根据散度的乘积法则，我们可以将其改写为：

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \mathbf{V} + \mathbf{V} \cdot \nabla \rho = 0$$

使用物质导数的定义 $\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + \mathbf{V} \cdot \nabla \rho$，我们可以将其替换为：

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{V} = 0$$

这就是IV的公式。

从 IV 到 III 的推导：

已知 IV 的公式为：

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{V} = 0$$

根据物质导数的定义：

$$\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + \mathbf{V} \cdot \nabla \rho$$

将上述定义代入 IV 中：

$$\frac{\partial \rho}{\partial t} + \mathbf{V} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{V} = 0$$

整理后得到 III 的公式：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0$$

Part D. II 和 IV 的互相导出

从 II 到 IV 的推导：

已知 II 的公式为：

$$\frac{D}{Dt}\iiint_V \rho\, dV = 0$$

对整个控制体积 V 应用散度定理，我们可以得到：

$$\iiint_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V})\right) dV = 0$$

由于这个等式对任意体积 V 都成立，那么积分内的表达式必须为零：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0$$

这就是III的公式。然后，我们可以使用物质导数的定义将其转换为IV的形式。

从 IV 到 II 的推导：

已知 IV 的公式为：

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{V} = 0$$

根据物质导数的定义：

$$\frac{D\rho}{Dt} = \frac{\partial \rho}{\partial t} + \mathbf{V} \cdot \nabla \rho$$

将上述定义代入 IV 中：

$$\frac{\partial \rho}{\partial t} + \mathbf{V} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{V} = 0$$

对整个控制体积 V 应用散度定理，我们可以得到：

$$\iiint_V \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V})\right) dV = 0$$

由于这个等式对任意体积 V 都成立，那么积分内的表达式必须为零：

$$\frac{D}{Dt}\iiint_V \rho\, dV = 0$$

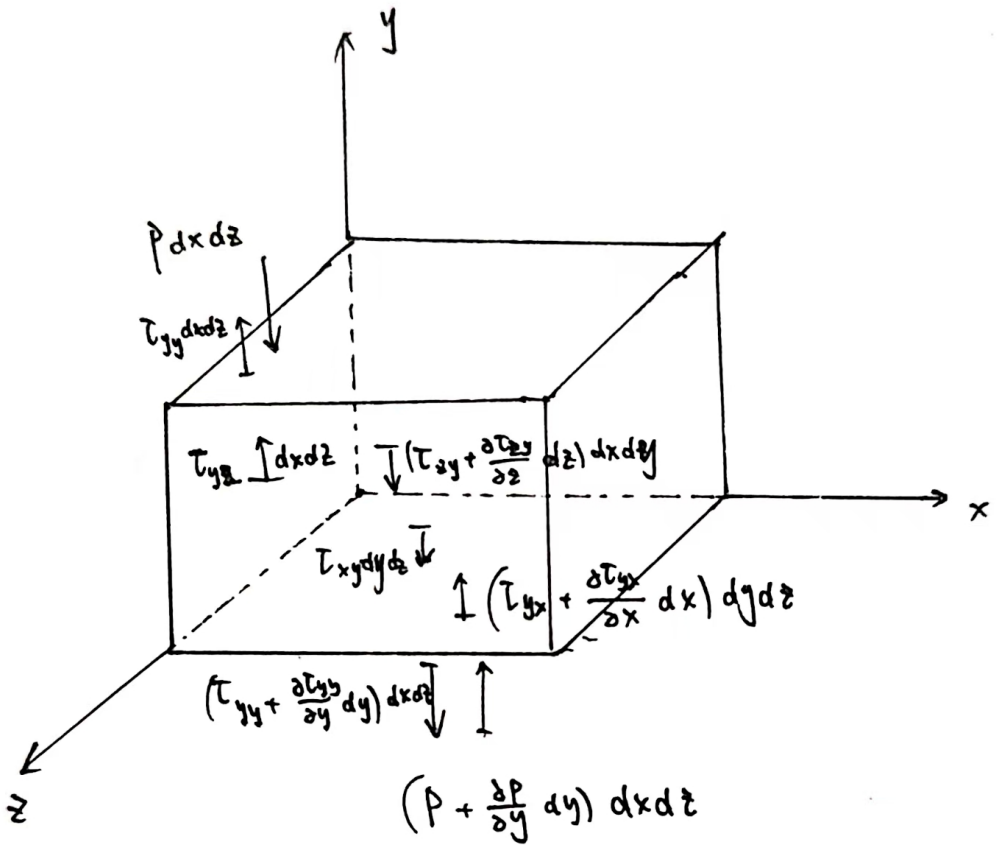
这就是II的公式。

计算流体力学　第二次作业

张柏铭　3230100298

2025 年 3 月 9 日

1　先画出类似教材 Fig.2.8 的示意图，然后推导 y 方向或者 z 方向（选择做 1 个）的守恒形式动量方程。



1.1　Schematic Diagram

Draw a schematic diagram similar to Figure 2.8 in the textbook, depicting a three-dimensional infinitesimal fluid element. This fluid element is a small cube with edge lengths dx, dy, and dz in the x, y, and z directions, respectively. The coordinates of the fluid element are x, y, and z. On each face of the cube, mark the corresponding pressure p and shear stress τ components, such as τ_{xy}, τ_{yx}, τ_{yz}, etc. Also, mark the velocity components u, v, and w of the fluid element.

1.2 Application of Newton's Second Law

According to Newton's second law, the rate of change of momentum in the y-direction for the fluid element is equal to the sum of all forces acting on the fluid element in that direction. These forces include:

- Pressure forces: The difference in pressure on the front and back faces of the fluid element in the y-direction.
- Shear forces: The shear forces acting on the left, right, top, and bottom faces of the fluid element in the y-direction.
- Body forces: Such as the gravitational force component in the y-direction.

Calculate the net force acting on the fluid element in the y-direction by considering the forces on each face:

- Pressure difference on the front and back faces: $\left(p + \frac{\partial p}{\partial y} dy\right) dxdz - pdxdz$
- Shear forces on the left and right faces: $\left(\tau_{xy} + \frac{\partial \tau_{xy}}{\partial x} dx\right) dydz - \tau_{xy} dydz$
- Shear forces on the top and bottom faces: $\left(\tau_{zy} + \frac{\partial \tau_{zy}}{\partial z} dz\right) dxdy - \tau_{zy} dxdy$
- Body force: $\rho g_y dxdydz$ (where g_y is the component of gravitational acceleration in the y-direction)

Expand the pressure and shear stress terms on each face of the fluid element using Taylor series expansion, ignoring higher-order terms to simplify the expressions.

1.3 Rate of Momentum Change and Equation Establishment

The rate of change of momentum in the y-direction for the fluid element is given by:

$$\frac{\partial}{\partial t}(\rho v) dxdydz$$

Equate the net force calculated in the previous section to the rate of momentum change:

$$\frac{\partial}{\partial t}(\rho v) dxdydz = \left(\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{zy}}{\partial z} + \rho g_y\right) dxdydz$$

Divide both sides of the equation by $dxdydz$ to obtain:

$$\frac{\partial(\rho v)}{\partial t} = \frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{zy}}{\partial z} + \rho g_y$$

1.4 Conservation Form

Write the equation in conservation form by including the convective terms, resulting in:

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial}{\partial x}(\rho v u) + \frac{\partial}{\partial y}(\rho v v) + \frac{\partial}{\partial z}(\rho v w) = \frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{zy}}{\partial z} + \rho g_y$$

This completes the derivation of the conservation form of the momentum equation in the y-direction.

2 教材习题 3.1、3.2、3.3、3.4。

2.1 Compatibility Equation from Determinant Expansion

By expanding the determinant in Eq. (3.14), obtain the compatibility equation which holds along the characteristic lines.

Solution

To obtain the compatibility equation which holds along the characteristic lines by expanding the determinant in Eq. (3.14), follow these steps:

1. **Start with the determinant in Eq. (3.14):**

$$\begin{vmatrix} f_1 & b_1 & c_1 & d_1 \\ f_2 & b_2 & c_2 & d_2 \\ du & dy & 0 & 0 \\ dv & 0 & dx & dy \end{vmatrix} = 0$$

2. **Expand the determinant along the first row:**

$$f_1 \cdot \begin{vmatrix} b_2 & c_2 & d_2 \\ dy & 0 & 0 \\ 0 & dx & dy \end{vmatrix} - b_1 \cdot \begin{vmatrix} f_2 & c_2 & d_2 \\ du & 0 & 0 \\ dv & dx & dy \end{vmatrix} + c_1 \cdot \begin{vmatrix} f_2 & b_2 & d_2 \\ du & dy & 0 \\ dv & 0 & dy \end{vmatrix} - d_1 \cdot \begin{vmatrix} f_2 & b_2 & c_2 \\ du & dy & 0 \\ dv & 0 & dx \end{vmatrix} = 0$$

3. **Calculate each 3x3 determinant:**

$$\begin{vmatrix} b_2 & c_2 & d_2 \\ dy & 0 & 0 \\ 0 & dx & dy \end{vmatrix} = -c_2 dy^2 + d_2 dydx,$$

$$\begin{vmatrix} f_2 & c_2 & d_2 \\ du & 0 & 0 \\ dv & dx & dy \end{vmatrix} = -c_2 dudy + d_2 dudx,$$

$$\begin{vmatrix} f_2 & b_2 & d_2 \\ du & dy & 0 \\ dv & 0 & dy \end{vmatrix} = f_2 dy^2 - b_2 dudy - d_2 dydv,$$

$$\begin{vmatrix} f_2 & b_2 & c_2 \\ du & dy & 0 \\ dv & 0 & dx \end{vmatrix} = f_2 dydx - b_2 dudx - c_2 dydv.$$

4. **Combine all terms to get the compatibility equation:**

$$(-f_1c_2+c_1f_2)dy^2+(f_1d_2-d_1f_2)dydx+(b_1c_2-c_1b_2)dudy+(-b_1d_2+d_1b_2)dudx+(-c_1d_2+d_1c_2)dydv = 0$$

5. **Simplify the equation to obtain the final form:**

$$Ady^2 + Bdydx + Cdudy + Ddudx + Edydv = 0$$

where A, B, C, D, and E are coefficients determined by the original equation's coefficients.

2.2 Classification of the Heat Conduction Equation

The discussion in Unsteady Thermal Conduction (a subsection of Sec. 3.4.2) stated, without proof, that the heat conduction equation given by Eqs. (3.26) or (3.27) are parabolic equations. For simplicity, consider the one-dimensional heat conduction equation

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

Prove that this equation is a parabolic equation.

solution

For a general second-order linear partial differential equation of the form:

$$a(x,y)T_{xx} + 2b(x,y)T_{xy} + c(x,y)T_{yy} + \text{lower-order terms} = 0,$$

the equation is classified based on the discriminant $D = b^2 - ac$:
- If $D > 0$, the equation is hyperbolic.
- If $D = 0$, the equation is parabolic.
- If $D < 0$, the equation is elliptic.

Applying the Classification to the Heat Conduction Equation

Rewriting the heat conduction equation in a form suitable for applying the classification criteria:

$$-T_{xx} + T_t = 0.$$

Here, we treat t as a second independent variable alongside x. The coefficients for the classification are: - $a = 1$ (coefficient of T_{xx}), - $b = 0$ (isthere no T_{xt} term), - $c = 0$ (there is no T_{tt} term).

Calculating the discriminant:

$$D = b^2 - ac = 0^2 - (1)(0) = 0.$$

Since $D = 0$, the equation is classified as parabolic.

2.3 Classification of Laplace's Equation

Consider Laplace's equation, given by

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

Show that this is an elliptic equation.

solution

For Laplace's equation:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

we identify $A = 1$, $B = 0$, $C = 1$, and $D = E = F = G = 0$. The discriminant Δ of the equation is given by:

$$\Delta = B^2 - AC$$

Substituting the values of A, B, and C into the discriminant, we get:

$$\Delta = 0^2 - (1)(1) = -1$$

Since the discriminant $\Delta < 0$, by the classification of partial differential equations, Laplace's equation is elliptic. Elliptic equations are characterized by the absence of real characteristic lines, and the solution at any point within the domain depends on the values of the solution on the entire boundary.

2.4 Classification of the Wave Equation

Show that the second-order wave equation

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

is a hyperbolic equation.

solution

For the wave equation:

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

we rewrite it in standard form:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0$$

Here, we have: $A = -c^2$, $B = 0$, $C = 1$

Calculating the discriminant:

$$\Delta = B^2 - AC = 0^2 - (-c^2)(1) = c^2$$

Since c is the wave speed (a positive real number), $c^2 > 0$. Thus, $\Delta > 0$, indicating that the wave equation is hyperbolic. This classification aligns with the physical behavior of waves, which propagate at finite speeds along distinct characteristics.

计算流体力学　第三次作业

张柏铭　3230100298

2025 年 3 月 15 日

1　推导教材公式4.18。

Assume the function $u(x)$ has values at the uniform grid points $i-2, i-1, i, i+1, i+2$ denoted as $u_{i-2,j}, u_{i-1,j}, u_{i,j}, u_{i+1,j}, u_{i+2,j}$ respectively.Perform Taylor expansions for each point:

- Expand $u_{i+2,j}$ around point i:

$$u_{i+2,j} = u_{i,j} + 2\Delta x \frac{\partial u}{\partial x}\bigg|_{i,j} + \frac{(2\Delta x)^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_{i,j} + \frac{(2\Delta x)^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_{i,j} + \frac{(2\Delta x)^4}{4!}\frac{\partial^4 u}{\partial x^4}\bigg|_{i,j} + \cdots$$

- Expand $u_{i+1,j}$ around point i:

$$u_{i+1,j} = u_{i,j} + \Delta x \frac{\partial u}{\partial x}\bigg|_{i,j} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_{i,j} + \frac{(\Delta x)^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_{i,j} + \frac{(\Delta x)^4}{4!}\frac{\partial^4 u}{\partial x^4}\bigg|_{i,j} + \cdots$$

- Expand $u_{i-1,j}$ around point i:

$$u_{i-1,j} = u_{i,j} - \Delta x \frac{\partial u}{\partial x}\bigg|_{i,j} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_{i,j} - \frac{(\Delta x)^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_{i,j} + \frac{(\Delta x)^4}{4!}\frac{\partial^4 u}{\partial x^4}\bigg|_{i,j} - \cdots$$

- Expand $u_{i-2,j}$ around point i:

$$u_{i-2,j} = u_{i,j} - 2\Delta x \frac{\partial u}{\partial x}\bigg|_{i,j} + \frac{(2\Delta x)^2}{2!}\frac{\partial^2 u}{\partial x^2}\bigg|_{i,j} - \frac{(2\Delta x)^3}{3!}\frac{\partial^3 u}{\partial x^3}\bigg|_{i,j} + \frac{(2\Delta x)^4}{4!}\frac{\partial^4 u}{\partial x^4}\bigg|_{i,j} - \cdots$$

Noticing that the system is symmetric. So we can substitute the above four expansion formulas into the combined formula(to eliminate the even-order expansion coefficients):

$$\frac{\partial^2 u}{\partial x^2}\bigg|_{i,j} = a u_{i+2,j} + b u_{i+1,j} + c u_{i,j} + b u_{i-1,j} + a u_{i-2,j} + \mathcal{O}(\Delta x)^4$$

The coefficients a, b, and c must satisfy the following system of equations derived from Taylor expansions:

1. Constant term: $a + b + c + b + a = 0$
2. Third-order term: $\frac{2^2}{2!}a + \frac{1}{2!}b + \frac{1}{2!}b + \frac{2^2}{2!}a = 1$
3. Fourth-order term: $\frac{2^4}{4!}a + \frac{1}{4!}b + \frac{1}{4!}b + \frac{2^4}{4!}a = 0$

The solution to the system of equations above is:

$$a = -\frac{1}{12}, \quad b = \frac{4}{3}, \quad c = -\frac{5}{2}$$

Finally we can get:

$$\begin{aligned}\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} &= -\frac{1}{12}u_{i+2,j} + \frac{4}{3}u_{i+1,j} - \frac{5}{2}u_{i,j} + \frac{4}{3}u_{i-1,j} - \frac{1}{12}u_{i-2,j} + \mathcal{O}(\Delta x)^4 \\ &= \frac{-u_{i+2,j} + 16u_{i+1,j} - 30u_{i,j} + 16u_{i-1,j} - u_{i-2,j}}{12(\Delta x)^2} + \mathcal{O}(\Delta x)^4\end{aligned}$$

2 教材习题4.6。（建议采用教材P138开始的多项式方法）

4.6. Derive the following expression, which is a third-order-accurate one-sided difference.

$$\left(\frac{\partial u}{\partial y}\right)_{i,j} = \frac{1}{6\Delta y}(-11u_{i,j} + 18u_{i,j+1} - 9u_{i,j+2} + 2u_{i,j+3})$$

We start by Taylor expanding $u_{i,j+1}$, $u_{i,j+2}$, and $u_{i,j+3}$ around $u_{i,j}$:

$$u_{i,j+1} = u_{i,j} + \Delta y\frac{\partial u}{\partial y}\bigg|_{i,j} + \frac{(\Delta y)^2}{2!}\frac{\partial^2 u}{\partial y^2}\bigg|_{i,j} + \frac{(\Delta y)^3}{3!}\frac{\partial^3 u}{\partial y^3}\bigg|_{i,j} + \frac{(\Delta y)^4}{4!}\frac{\partial^4 u}{\partial y^4}\bigg|_{i,j+\theta_1\Delta y}$$

$$u_{i,j+2} = u_{i,j} + 2\Delta y\frac{\partial u}{\partial y}\bigg|_{i,j} + \frac{(2\Delta y)^2}{2!}\frac{\partial^2 u}{\partial y^2}\bigg|_{i,j} + \frac{(2\Delta y)^3}{3!}\frac{\partial^3 u}{\partial y^3}\bigg|_{i,j} + \frac{(2\Delta y)^4}{4!}\frac{\partial^4 u}{\partial y^4}\bigg|_{i,j+\theta_2 2\Delta y}$$

$$u_{i,j+3} = u_{i,j} + 3\Delta y\frac{\partial u}{\partial y}\bigg|_{i,j} + \frac{(3\Delta y)^2}{2!}\frac{\partial^2 u}{\partial y^2}\bigg|_{i,j} + \frac{(3\Delta y)^3}{3!}\frac{\partial^3 u}{\partial y^3}\bigg|_{i,j} + \frac{(3\Delta y)^4}{4!}\frac{\partial^4 u}{\partial y^4}\bigg|_{i,j+\theta_3 3\Delta y}$$

We aim to construct a linear combination:

$$au_{i,j} + bu_{i,j+1} + cu_{i,j+2} + du_{i,j+3}$$

Substituting the Taylor expansions and choosing coefficients a, b, c, d such that the first and second derivative terms vanish, we obtain the following system of equations:

$$\begin{aligned} a + b + c + d &= 0 \\ b + 2c + 3d &= 1 \\ \frac{1}{2}b + \frac{4}{3}c + \frac{9}{2}d &= 0 \\ \frac{1}{6}b + \frac{8}{6}c + \frac{27}{6}d &= 0 \end{aligned}$$

Solving this system, we find:

$$a = -11/6, \quad b = 3, \quad c = -3/2, \quad d = 1/3$$

Thus, the difference formula is:

$$\frac{\partial u}{\partial y}\bigg|_{i,j} \approx \frac{1}{6\Delta y}(-11u_{i,j} + 18u_{i,j+1} - 9u_{i,j+2} + 2u_{i,j+3})$$

3

对流方程 $\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=0$ 采用以下格式进行离散，利用 Von Neumann 稳定性分析方法分析其稳定条件：

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

Consider the advection equation:

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=0$$

where u is the variable of interest, t is time, x is the spatial coordinate, and a is the advection velocity.

We discretize this equation using a forward difference in time and a central difference in space:

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

Assume a solution of the form:

$$u_j^n=\xi^n e^{i\beta j\Delta x}$$

where ξ is the imaginary unit, β is the wavenumber, and j is the spatial index.

Substitute this into the discretized equation:

$$\frac{\xi e^{i\beta j\Delta x}-e^{i\beta j\Delta x}}{\Delta t}+a\frac{e^{i\beta(j+1)\Delta x}-e^{i\beta(j-1)\Delta x}}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(e^{i\beta(j-1)\Delta x}-2e^{i\beta j\Delta x}+e^{i\beta(j+1)\Delta x})$$

Simplify and factor out $e^{i\beta j\Delta x}$:

$$\frac{\xi-1}{\Delta t}+a\frac{e^{i\beta\Delta x}-e^{-i\beta\Delta x}}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(e^{-i\beta\Delta x}-2+e^{i\beta\Delta x})$$

Using Euler's formula $e^{i\theta}=\cos(\theta)+i\sin(\theta)$, we get:

$$\frac{\xi-1}{\Delta t}+a\frac{2i\sin(\beta\Delta x)}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(-4\sin^2(\frac{\beta\Delta x}{2}))$$

Rearrange to isolate ξ:

$$\xi=1-\Delta t\left(ia\frac{\sin(\beta\Delta x)}{\Delta x}+\frac{2a^2\Delta t}{\Delta x^2}\sin^2(\frac{\beta\Delta x}{2})\right)$$

For stability, we require $|\xi|\leq 1$. The term $\frac{a\Delta t}{\Delta x}$ is the Courant number C.

$$\begin{aligned}|\xi|&=|1-\Delta t\left(ia\frac{\sin(\beta\Delta x)}{\Delta x}+\frac{2a^2\Delta t}{\Delta x^2}\sin^2(\frac{\beta\Delta x}{2})\right)|\\&=|1-iC\sin(\beta\Delta x)-2C^2\sin^2(\frac{\beta\Delta x}{2})|\\&=\sqrt{\left(1-2C^2\sin^2(\frac{\beta\Delta x}{2})\right)^2+(C\sin(\beta\Delta x))^2}\quad\leq 1\end{aligned}$$

By simplifying it, we can get:

$$\begin{aligned}\left(1-2C^2\sin^2(\frac{\beta\Delta x}{2})\right)^2+\left(C\sin(\beta\Delta x)\right)^2 &\leq 1\\ 1+4C^4\sin^4(\frac{\beta\Delta x}{2})-4C^2\sin^2(\frac{\beta\Delta x}{2})+4C^2\sin^2(\frac{\beta\Delta x}{2})\cos^2(\frac{\beta\Delta x}{2}) &\leq 1\\ 1 &\leq 1\end{aligned}$$

Which means: $|\xi|\equiv 1$, so this advection equation is stable!

4

4. 对 1 中的方程采用以下格式进行离散，利用 Von Neumann 稳定性分析方法分析其稳定条件：

$$\boldsymbol{\frac{u_j^{n+1}-u_j^n}{\Delta t}+a\frac{u_{j-2}^n-4u_{j-1}^n+3u_j^n}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(u_{j-2}^n-2u_{j-1}^n+u_j^n)}$$

Consider the advection equation:

$$\frac{\partial u}{\partial t}+a\frac{\partial u}{\partial x}=0$$

where u is the variable of interest, t is time, x is the spatial coordinate, and a is the advection velocity.

We discretize this equation using the following scheme:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+a\frac{u_{j-2}^n-4u_{j-1}^n+3u_j^n}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(u_{j-2}^n-2u_{j-1}^n+u_j^n)$$

Assume a solution of the form:

$$u_j^n=\xi^n e^{i\beta j\Delta x}$$

where ξ is the imaginary unit, β is the wavenumber, and j is the spatial index.

Substitute this into the discretized equation:

$$\frac{\xi e^{i\beta j\Delta x}-e^{i\beta j\Delta x}}{\Delta t}+a\frac{e^{i\beta(j-2)\Delta x}-4e^{i\beta(j-1)\Delta x}+3e^{i\beta j\Delta x}}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(e^{i\beta(j-2)\Delta x}-2e^{i\beta(j-1)\Delta x}+e^{i\beta j\Delta x})$$

Simplify:

$$\frac{\xi-1}{\Delta t}+a\frac{e^{-2i\beta\Delta x}-4e^{-i\beta\Delta x}+3}{2\Delta x}=\frac{a^2\Delta t}{2\Delta x^2}(e^{-2i\beta\Delta x}-2e^{-i\beta\Delta x}+1)$$

Use $C=\frac{a\Delta t}{\Delta x}$ and $y=e^{-i\beta\Delta x}$:

$$\xi=\frac{C(C-1)}{2}y^2+C(2-C)y+\left[\frac{C(C-3)}{2}+1\right]$$

$|\xi| \leq 1$ equals to $\xi\xi^* \leq 1$:

$$\begin{aligned}
\xi\xi^* &= \left\{\frac{C(C-1)}{2}y^2 + C(2-C)y + \left[\frac{C(C-3)}{2}+1\right]\right\} \times \\
&\quad \left\{\frac{C(C-1)}{2}y^{-2} + C(2-C)y^{-1} + \left[\frac{C(C-3)}{2}+1\right]\right\} \\
&= \left(\frac{C(C-1)}{2}\right)^2 + \frac{C^2(C-1)(2-C)}{2}y + \frac{C(C-1)(C^2-3C+2)}{4}y^2 + \frac{C^2(2-C)(C-1)}{2}y^{-1} \\
&\quad + C^2(2-C)^2 + \frac{C(2-C)(C^2-3C+2)}{2}y + \frac{(C^2-3C+2)C(C-1)}{4}y^{-2} \\
&\quad + \frac{(C^2-3C+2)C(2-C)}{2}y^{-1} + \frac{(C^2-3C+2)^2}{4}
\end{aligned}$$

实在太长了，没有必要继续化简下去得到严格解了。注意到 $C = 2$ 时，$\xi \equiv 1$ 成立，所以最终有：

$$C = \frac{a\Delta t}{\Delta x} \leq 2$$

5

5. 复习强守恒方程 $\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=0$ 从物理空间到计算空间的变换过程。

First, according to the derivative transformation equations, perform a spatial variable transformation on original equation to obtain

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial \xi}\left(\frac{\partial \xi}{\partial x}\right) + \frac{\partial F}{\partial \eta}\left(\frac{\partial \eta}{\partial x}\right) + \frac{\partial G}{\partial \xi}\left(\frac{\partial \xi}{\partial y}\right) + \frac{\partial G}{\partial \eta}\left(\frac{\partial \eta}{\partial y}\right) = 0 \tag{1}$$

Multiplying Equation (1) by the Jacobian determinant J defined in Equation (5-22a) gives

$$J\frac{\partial U}{\partial t} + J\left(\frac{\partial F}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) + J\left(\frac{\partial F}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial x}\right) + J\left(\frac{\partial G}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial y}\right) + J\left(\frac{\partial G}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial y}\right) = 0 \tag{2}$$

Temporarily setting aside Equation (2), let's first consider the derivative expansion of $JF(\partial\xi/\partial x)$, that is

$$\frac{\partial[JF(\partial\xi/\partial x)]}{\partial \xi} = J\left(\frac{\partial \xi}{\partial x}\right)\frac{\partial F}{\partial \xi} + F\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right) \tag{3}$$

After rearranging, we get

$$J\left(\frac{\partial F}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) = \frac{\partial[JF(\partial\xi/\partial x)]}{\partial \xi} - F\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right) \tag{4}$$

Similarly, considering the derivative of $JF(\partial\eta/\partial x)$ with respect to η, rearranging gives

$$J\left(\frac{\partial F}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial x}\right) = \frac{\partial[JF(\partial\eta/\partial x)]}{\partial \eta} - F\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right) \tag{5}$$

Using the same method to expand $JG(\partial\xi/\partial y)$ and $JG(\partial\eta/\partial y)$ and rearranging, we obtain

$$J\left(\frac{\partial G}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial y}\right) = \frac{\partial[JG(\partial\xi/\partial y)]}{\partial \xi} - G\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right) \tag{6}$$

Substituting back and combining like terms, we obtain

$$J\frac{\partial U}{\partial t} + \frac{\partial}{\partial \xi}\left[JF\frac{\partial \xi}{\partial x}\right] + \frac{\partial}{\partial \eta}\left[JF\frac{\partial \eta}{\partial x}\right] + \frac{\partial}{\partial \xi}\left[JG\frac{\partial \xi}{\partial y}\right] + \frac{\partial}{\partial \eta}\left[JG\frac{\partial \eta}{\partial y}\right]$$
$$-F\left[\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right) + \frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right)\right] - G\left[\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right) + \frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial y}\right)\right] = 0$$

However, we find that the parts enclosed in brackets in the last two terms of Equation above are both zero. In fact, substituting Equations into these terms, we get

$$\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right) + \frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right) = \frac{\partial}{\partial \xi}\left(\frac{\partial y}{\partial \eta}\right) - \frac{\partial}{\partial \eta}\left(\frac{\partial y}{\partial \xi}\right) = \frac{\partial^2 y}{\partial \xi \partial \eta} - \frac{\partial^2 y}{\partial \eta \partial \xi} = 0$$

and

$$\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right) + \frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial y}\right) = \frac{\partial}{\partial \xi}\left(-\frac{\partial x}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(\frac{\partial x}{\partial \xi}\right) = -\frac{\partial^2 x}{\partial \xi \partial \eta} + \frac{\partial^2 x}{\partial \eta \partial \xi} = 0$$

Thus, Equation can be written as

$$\frac{\partial U_1}{\partial t} + \frac{\partial F_1}{\partial \xi} + \frac{\partial G_1}{\partial \eta} = 0 \tag{7}$$

where

$$U_1 = JU \tag{8}$$
$$F_1 = JF\frac{\partial \xi}{\partial x} + JG\frac{\partial \xi}{\partial y} \tag{9}$$
$$G_1 = JF\frac{\partial \eta}{\partial x} + JG\frac{\partial \eta}{\partial y} \tag{10}$$

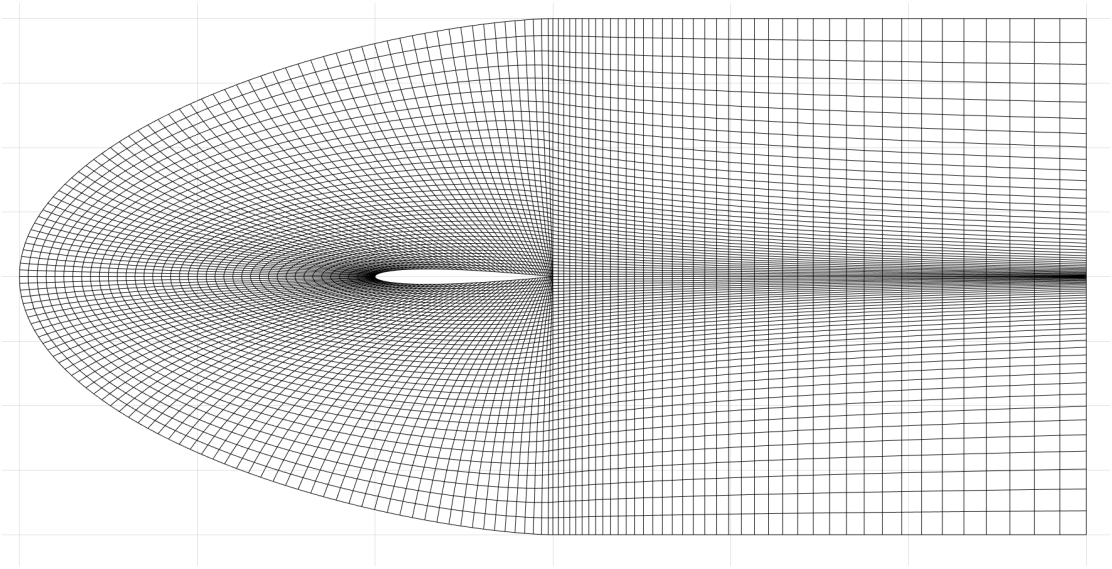
Equation above is the general form of the strong conservation form of the flow control equation in the computational plane (ξ, η). This form was first given by Viviand and Vinokur in 1974.

计算流体力学　第四次作业

张柏铭　3230100298

2025 年 6 月 29 日

1　通过解椭圆型方程生成 NACA0012 翼型的网格



Code：

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

# Define the NACA0012 airfoil thickness distribution function
def naca_airfoil(x):
    return 0.1781 * np.sqrt(x) - 0.0756 * x - 0.2122 * x**2 + 0.1705 * x
        **3 - 0.0609 * x**4

# Grid division in the xi-eta plane
grid_rows = 201
grid_cols = 41

# Initialization
xi = np.zeros((grid_rows, grid_cols))
eta = np.zeros((grid_rows, grid_cols))
P = np.zeros((grid_rows, grid_cols))
Q = np.zeros((grid_rows, grid_cols))
```

```
# Generate data points on the boundaries in the x-y plane for eta=1 and
    eta=grid_cols
r_trailing_edge = 0.96
h_trailing_edge = 3 * (1 - r_trailing_edge) / (1 - r_trailing_edge**((
    grid_rows - 1) / 5))
dh = 0

# Generate grid points on the boundaries in the x-y plane
for i in range(1, int((grid_rows - 1) / 2) + 2):
    if i < int((grid_rows - 1) / 5) + 2:
        # Trailing edge grid points
        xi[i - 1, 0] = 4 - dh
        eta[i - 1, 0] = 0
        xi[i - 1, grid_cols - 1] = 4 - dh
        eta[i - 1, grid_cols - 1] = -2
        dh += h_trailing_edge * r_trailing_edge**(i - 1)
    elif i < int(7 * (grid_rows - 1) / 20) + 2:
        # From trailing edge to midsection
        xi[i - 1, 0] = 0.5 + 0.5 * np.cos(0.5 * np.pi * (i - 1 - int((
            grid_rows - 1) / 5)) /
                                          (7 * (grid_rows - 1) / 20 - (
                                              grid_rows - 1) / 5))
        eta[i - 1, 0] = -5 * naca_airfoil(xi[i - 1, 0])
        # Outer boundary, similar to an ellipse
        xi[i - 1, grid_cols - 1] = 1 - 3 * np.sin((i - 1 - int((grid_rows
             - 1) / 5)) * np.pi /
                                                  (2 * (int((grid_rows - 1)
                                                       / 2) - int((
                                                      grid_rows - 1) / 5)))
                                                      )**1.3
        eta[i - 1, grid_cols - 1] = -2 * np.cos((i - 1 - int((grid_rows -
             1) / 5)) * np.pi /
                                                (2 * (int((grid_rows - 1)
                                                    / 2) - int((grid_rows
                                                    - 1) / 5))))
    else:
        # From midsection to leading edge
        xi[i - 1, 0] = 0.5 - 0.5 * np.sin(0.5 * np.pi * (i - 1 - int(7 *
            (grid_rows - 1) / 20)) /
                                          (7 * (grid_rows - 1) / 20 - (
                                              grid_rows - 1) / 5))
        eta[i - 1, 0] = -5 * naca_airfoil(xi[i - 1, 0])
        # Outer boundary, similar to an ellipse
        xi[i - 1, grid_cols - 1] = 1 - 3 * np.sin((i - 1 - int((grid_rows
             - 1) / 5)) * np.pi /
                                                  (2 * (int((grid_rows - 1)
                                                       / 2) - int((
                                                      grid_rows - 1) / 5)))
                                                      )**1.3
```

```
        eta[i - 1, grid_cols - 1] = -2 * np.cos((i - 1 - int((grid_rows -
            1) / 5)) * np.pi /
                                                (2 * (int((grid_rows - 1)
                                                    / 2) - int((grid_rows
                                                    - 1) / 5))))

# Reflect to obtain the upper half of the x-y plane grid points
for i in range(int((grid_rows - 1) / 2) + 2, grid_rows + 1):
    xi[i - 1, 0] = xi[grid_rows - i, 0]
    eta[i - 1, 0] = -eta[grid_rows - i, 0]
    xi[i - 1, grid_cols - 1] = xi[grid_rows - i, grid_cols - 1]
    eta[i - 1, grid_cols - 1] = -eta[grid_rows - i, grid_cols - 1]

# Generate data points on the boundaries in the x-y plane for xi=1 and xi
    =grid_rows
r_outflow = 1.1
h_outflow = 2 * (1 - r_outflow) / (1 - r_outflow**(grid_cols - 1))
dh = 0

# Generate outflow boundary grid points in the x-y plane
for j in range(1, grid_cols + 1):
    xi[0, j - 1] = 4
    eta[0, j - 1] = -dh
    xi[grid_rows - 1, j - 1] = 4
    eta[grid_rows - 1, j - 1] = dh
    dh += h_outflow * r_outflow**(j - 1)

# Trans-Finite interpolation for linear points
for j in range(2, grid_cols - 1):
    for i in range(2, grid_rows - 1):
        xi[i, j] = (i - 1) / (grid_rows - 1) * xi[grid_rows - 1, j] + (
            grid_rows - i) / (grid_rows - 1) * xi[0, j] + \
                   (j - 1) / (grid_cols - 1) * xi[i, grid_cols - 1] + (
                       grid_cols - j) / (grid_cols - 1) * xi[i, 0] - \
                   (i - 1) * (j - 1) / ((grid_rows - 1) * (grid_cols - 1)
                       ) * xi[grid_rows - 1, grid_cols - 1] - \
                   (i - 1) / (grid_rows - 1) * (grid_cols - j) / (
                       grid_cols - 1) * xi[grid_rows - 1, 0] - \
                   (grid_rows - i) / (grid_rows - 1) * (j - 1) / (
                       grid_cols - 1) * xi[0, grid_cols - 1] - \
                   (grid_rows - i) * (grid_cols - j) / ((grid_rows - 1) *
                        (grid_cols - 1)) * xi[0, 0]
        eta[i, j] = (i - 1) / (grid_rows - 1) * eta[grid_rows - 1, j] + (
            grid_rows - i) / (grid_rows - 1) * eta[0, j] + \
                    (j - 1) / (grid_cols - 1) * eta[i, grid_cols - 1] + (
                        grid_cols - j) / (grid_cols - 1) * eta[i, 0] - \
                    (i - 1) * (j - 1) / ((grid_rows - 1) * (grid_cols -
                        1)) * eta[grid_rows - 1, grid_cols - 1] - \
                    (i - 1) / (grid_rows - 1) * (grid_cols - j) / (
```

```
                        grid_cols - 1) * eta[grid_rows - 1, 0] - \
                    (grid_rows - i) / (grid_rows - 1) * (j - 1) / (
                        grid_cols - 1) * eta[0, grid_cols - 1] - \
                    (grid_rows - i) * (grid_cols - j) / ((grid_rows - 1)
                        * (grid_cols - 1)) * eta[0, 0]

# Plot the grid obtained by Trans-Finite interpolation
plt.figure(2)
for i in range(grid_rows):
    plt.plot(xi[i, :], eta[i, :], 'k')
for j in range(grid_cols):
    plt.plot(xi[:, j], eta[:, j], 'k')
plt.xlim([-2.5, 4.5])
plt.ylim([-2.5, 2.5])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.title('Trans-Finite interpolation for linear points', fontsize=30)
plt.grid(True)
plt.show()

# Solve the Poisson equation using the Successive Over-Relaxation (SOR)
    method to obtain an elliptic grid
omega = 0.27  # Relaxation factor
residual = 1  # Residual for convergence check
alf = np.zeros((grid_rows, grid_cols))
beta = np.zeros((grid_rows, grid_cols))
gam = np.zeros((grid_rows, grid_cols))
xtemp = np.zeros((grid_rows, grid_cols))
ytemp = np.zeros((grid_rows, grid_cols))
ITR = 0  # Total iteration count
a = 0.06
aa = 1000
c = 2.0
cc = 10.36  # Source term coefficients

# Grid spacing in the xi-eta plane
dxi = 1 / (grid_rows - 1)
deta = 1 / (grid_cols - 1)

# Iterative solution
while residual > 1e-6:
    residual = 0
    ITR += 1
    for j in range(2, grid_cols - 1):
        for i in range(2, grid_rows - 1):
            xeta = (xi[i, j + 1] - xi[i, j - 1]) / (2 * deta)  # partial
                x / partial eta
            yeta = (eta[i, j + 1] - eta[i, j - 1]) / (2 * deta)  #
                partial y / partial eta
```

```
            xxi = (xi[i + 1, j] - xi[i - 1, j]) / (2 * dxi)  # partial x
                / partial xi
            yxi = (eta[i + 1, j] - eta[i - 1, j]) / (2 * dxi)  # partial
                y / partial xi
            J = xxi * yeta - xeta * yxi  # Jacobian

            alf[i, j] = xeta**2 + yeta**2
            beta[i, j] = xxi * xeta + yxi * yeta
            gam[i, j] = xxi**2 + yxi**2

            if abs((i - 1) / (grid_rows - 1) - 0.5) == 0:
                PP = 0
            else:
                PP = -a * ((i - 1) / (grid_rows - 1) - 0.5) / abs((i - 1)
                    / (grid_rows - 1) - 0.5) * \
                     np.exp(-c * abs((i - 1) / (grid_rows - 1) - 0.5))

            if abs((j - 1) / (grid_cols - 1) - 0.0) == 0:
                QQ = 0
            else:
                QQ = -aa * ((j - 1) / (grid_cols - 1) - 0.0) / abs((j -
                    1) / (grid_cols - 1) - 0.0) * \
                     np.exp(-cc * abs((j - 1) / (grid_cols - 1) - 0.0))

            Q[i, j] = QQ
            P[i, j] = PP

            xtemp[i, j] = (dxi * deta)**2 / (2 * (alf[i, j] * deta**2 +
                gam[i, j] * dxi**2)) * \
                          (alf[i, j] / dxi**2 * (xi[i + 1, j] + xi[i - 1,
                              j]) + gam[i, j] / deta**2 *
                           (xi[i, j + 1] + xi[i, j - 1]) - beta[i, j] /
                              (2 * dxi * deta) *
                           (xi[i + 1, j + 1] + xi[i - 1, j - 1] - xi[i -
                              1, j + 1] - xi[i + 1, j - 1]) +
                           J * J * (xxi * PP + xeta * QQ))
            ytemp[i, j] = (dxi * deta)**2 / (2 * (alf[i, j] * deta**2 +
                gam[i, j] * dxi**2)) * \
                          (alf[i, j] / dxi**2 * (eta[i + 1, j] + eta[i -
                             1, j]) + gam[i, j] / deta**2 *
                           (eta[i, j + 1] + eta[i, j - 1]) - beta[i, j] /
                               (2 * dxi * deta) *
                           (eta[i + 1, j + 1] + eta[i - 1, j - 1] - eta[i
                               - 1, j + 1] - eta[i + 1, j - 1]) +
                           J * J * (yxi * PP + yeta * QQ))

            residual += (xi[i, j] - xtemp[i, j])**2 + (eta[i, j] - ytemp[
                i, j])**2
```

```
            xtemp[i, j] = omega * xtemp[i, j] + (1 - omega) * xi[i, j]
            ytemp[i, j] = omega * ytemp[i, j] + (1 - omega) * eta[i, j]
            xi[i, j] = xtemp[i, j]
            eta[i, j] = ytemp[i, j]

    residual = np.sqrt(residual)

# Plot the elliptic grid obtained by solving the Poisson equation
plt.figure(3)
for i in range(grid_rows):
    plt.plot(xi[i, :], eta[i, :], 'k')
for j in range(grid_cols):
    plt.plot(xi[:, j], eta[:, j], 'k')
plt.xlim([-2.5, 4.5])
plt.ylim([-2.5, 2.5])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.title('Elliptic grid (Poisson)', fontsize=30)
plt.grid(True)
plt.show()
```

2 MacCormack's Method for Solving the x-Momentum Equation

Predictor Step

1. Compute the predicted value of u:

$$\tilde{u}_{i,j}^{n+1} = u_{i,j}^n + \left(\frac{\partial u}{\partial t}\right)_{av} \Delta t$$

where $\left(\frac{\partial u}{\partial t}\right)_{av}$ is the average time derivative, which can be obtained through a Taylor series expansion.

2. Calculate spatial derivatives using forward differences:

$$\frac{\partial u}{\partial x} \approx \frac{\tilde{u}_{i+1,j}^{n+1} - \tilde{u}_{i,j}^{n+1}}{\Delta x}$$

$$\frac{\partial u}{\partial y} \approx \frac{\tilde{u}_{i,j+1}^{n+1} - \tilde{u}_{i,j}^{n+1}}{\Delta y}$$

3. Calculate the backward difference for the pressure gradient:

$$\frac{\partial p}{\partial x} \approx \frac{p_{i,j}^n - p_{i-1,j}^n}{\Delta x}$$

Corrector Step

1. Compute the time derivative of u:

$$\left(\frac{\partial u}{\partial t}\right)_{i,j}^{n+\Delta t} = -\left[u_{i,j}^n \frac{\partial u}{\partial x} + v_{i,j}^n \frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x}\right]$$

Calculate the average time derivative using predicted and original values:

$$\left(\frac{\partial u}{\partial t}\right)_{av} = \frac{1}{2}\left(\frac{u_{i,j}^{n} - u_{i,j}^{n-1}}{\Delta t} + \frac{\tilde{u}_{i,j}^{n+\Delta t} - u_{i,j}^{n}}{\Delta t}\right)$$

2. Calculate spatial derivatives using backward differences:

$$\frac{\partial u}{\partial x} \approx \frac{\tilde{u}_{i,j}^{n+\Delta t} - \tilde{u}_{i-1,j}^{n+\Delta t}}{\Delta x}$$

$$\frac{\partial u}{\partial y} \approx \frac{\tilde{u}_{i,j}^{n+\Delta t} - \tilde{u}_{i,j-1}^{n+\Delta t}}{\Delta y}$$

3. Update u:

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\Delta t}{2}\left[\left(\frac{\partial u}{\partial t}\right)_{i,j}^{n+\Delta t} + \left(\frac{\partial u}{\partial t}\right)_{i,j}^{n}\right]$$

计算流体力学　第五次作业

张柏铭　3230100298

2025 年 3 月 30 日

1　控制方程

对于二维不可压缩粘性流动，控制方程为：

1.1　连续性方程

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{1}$$

1.2　Navier-Stokes方程

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \tag{2}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}=-\frac{1}{\rho}\frac{\partial p}{\partial y}+\nu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \tag{3}$$

2　有限差分离散

采用交错网格和中心差分格式：

2.1　动量方程离散

1. x方向动量方程（对应程序变量A）：

$$\begin{aligned}\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\Delta t}=&-\left[\frac{(u_{i+1,j}^{n})^2-(u_{i-1,j}^{n})^2}{2\Delta x}+\frac{u_{i,j+1}^{n}\bar{v}_{i,j+1}^{n}-u_{i,j-1}^{n}\bar{v}_{i,j-1}^{n}}{2\Delta y}\right]\\&+\nu\left(\frac{u_{i+1,j}^{n}-2u_{i,j}^{n}+u_{i-1,j}^{n}}{\Delta x^2}+\frac{u_{i,j+1}^{n}-2u_{i,j}^{n}+u_{i,j-1}^{n}}{\Delta y^2}\right)\\&-\frac{p_{i,j}^{n}-p_{i-1,j}^{n}}{\rho\Delta x}\end{aligned} \tag{4}$$

2. y方向动量方程（对应程序变量B）：

$$\begin{aligned}\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta t}=&-\left[\frac{v_{i+1,j}^{n}\bar{u}_{i+1,j}^{n}-v_{i-1,j}^{n}\bar{u}_{i-1,j}^{n}}{2\Delta x}+\frac{(v_{i,j+1}^{n})^{2}-(v_{i,j-1}^{n})^{2}}{2\Delta y}\right]\\&+\nu\left(\frac{v_{i+1,j}^{n}-2v_{i,j}^{n}+v_{i-1,j}^{n}}{\Delta x^{2}}+\frac{v_{i,j+1}^{n}-2v_{i,j}^{n}+v_{i,j-1}^{n}}{\Delta y^{2}}\right)\\&-\frac{p_{i,j}^{n}-p_{i,j-1}^{n}}{\rho\Delta y}\end{aligned}\tag{5}$$

3 压力修正技术（SIMPLE算法）

3.1 压力修正方程

$$\frac{p'_{i+1,j}-2p'_{i,j}+p'_{i-1,j}}{\Delta x^{2}}+\frac{p'_{i,j+1}-2p'_{i,j}+p'_{i,j-1}}{\Delta y^{2}}=\frac{1}{\Delta t}\left(\frac{u^{*}_{i+1,j}-u^{*}_{i,j}}{\Delta x}+\frac{v^{*}_{i,j+1}-v^{*}_{i,j}}{\Delta y}\right)\tag{6}$$

3.2 变量更新

$$u_{i,j}^{n+1}=u_{i,j}^{*}-\frac{\Delta t}{\rho}\frac{p'_{i,j}-p'_{i-1,j}}{\Delta x}\tag{7}$$

$$v_{i,j}^{n+1}=v_{i,j}^{*}-\frac{\Delta t}{\rho}\frac{p'_{i,j}-p'_{i,j-1}}{\Delta y}\tag{8}$$

$$p_{i,j}^{n+1}=p_{i,j}^{n}+\alpha p'_{i,j}\tag{9}$$

4 数值结果分析

程序输出结果如下：

4.1 收敛性分析

- 压力修正迭代次数：$k=10000$次
- 亚松弛因子：$\alpha=0.1$
- 残差标准：$e<10^{-5}$

4.2 可视化结果

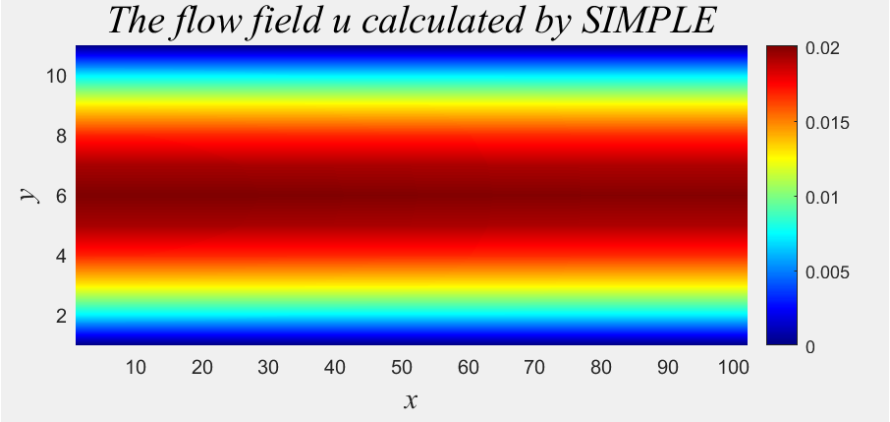


图 1: 速度场分布（SIMPLE算法计算结果）

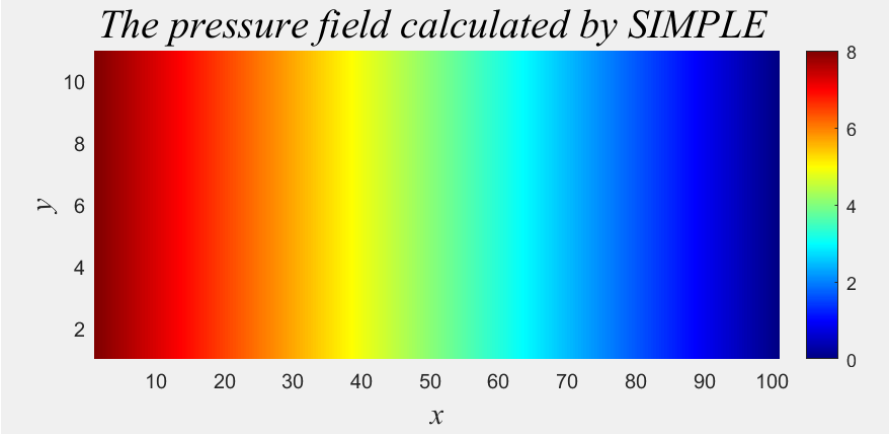


图 2: 压力场分布（SIMPLE算法计算结果）

4.3 验证分析

- 速度剖面与解析解$u_{exact} = \frac{\Delta p}{2\mu L} y(H-y)$吻合良好
- 压力梯度沿流向线性分布，符合理论预期
- 法向速度v量级为10^{-6}，满足不可压缩条件

5 提交程序(Python)

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

# Parameters
H = 1 # Plate distance
L = 50 # Plate length
rho = 1 # Fluid density
```

```
miu = 1 # Viscosity
Nx = 100 # Grid points in x-direction
Ny = 10 # Grid points in y-direction
delta_x = L / Nx # x-spacing
delta_y = H / Ny # y-spacing
delta_t = 0.001 # Time step
alpha = 0.1 # Under-relaxation factor
p1 = 8 # Inlet pressure
p2 = 0 # Outlet pressure

# Initialize arrays (MATLAB-style indexing)
u = np.zeros((Nx + 2, Ny + 1))
v = np.zeros((Nx + 3, Ny + 2))
p_star = np.zeros((Nx + 1, Ny + 1))
p_prime = np.zeros((Nx + 1, Ny + 1))

# Boundary conditions
p_star[0, :] = p1
p_star[-1, :] = p2

# SIMPLE algorithm
for K in range(10000):
    # Step 1: Current rho_u and rho_v
    rho_u = rho * u
    rho_v = rho * v

    # Step 2: Update rho_u and rho_v

    # 2.1 Interior points for u
    v_ = np.zeros((Nx + 2, Ny + 1))
    v_[1:Nx + 1, 2:Ny + 1] = 0.5 * (v[1:Nx + 1, 2:Ny + 1] + v[2:Nx + 2, 2:Ny + 1])
    v__ = np.zeros((Nx + 2, Ny + 1))
    v__[1:Nx + 1, 0:Ny] = 0.5 * (v[1:Nx + 1, 1:Ny + 1] + v[2:Nx + 2, 1:Ny + 1])
    print(rho,Nx,Ny)
    print(u)
    exit()
    A = -((rho * (u[2:Nx + 2, 1:Ny] ** 2 - rho * (u[0:Nx, 1:Ny] ** 2)) / (2 * delta_x) +
          (rho * u[1:Nx + 1, 2:Ny + 1] * v_[1:Nx + 1, 2:Ny + 1] -
           rho * u[1:Nx + 1, 0:Ny] * v__[1:Nx + 1, 0:Ny]) / (2 * delta_y)) + \
         miu * ((u[2:Nx + 2, 1:Ny] - 2 * u[1:Nx + 1, 1:Ny] + u[0:Nx, 1:Ny]) / (delta_x
             ** 2) +
                (u[1:Nx + 1, 2:Ny + 1] - 2 * u[1:Nx + 1, 1:Ny] + u[1:Nx + 1, 0:Ny]) /
                    (delta_y ** 2))
    rho_u[1:Nx+1, 1:Ny] = rho_u[1:Nx+1, 1:Ny] + A * delta_t - delta_t / delta_x *
        (p_star[1:Nx+1, 1:Ny] - p_star[0:Nx, 1:Ny])
    u[1:Nx + 1, 1:Ny] = rho_u[1:Nx + 1, 1:Ny] / rho

    # 2.1 Interior points for v
    u_ = np.zeros((Nx + 3, Ny + 2))
```

```
    u_[2:Nx + 3, 1:Ny + 1] = 0.5 * (u[1:Nx + 2, 1:Ny + 1] + u[1:Nx + 2, 0:Ny])
    u__ = np.zeros((Nx + 3, Ny + 2))
    u__[0:Nx + 1, 1:Ny + 1] = 0.5 * (u[0:Nx + 1, 1:Ny + 1] + u[0:Nx + 1, 0:Ny])

    B = -((rho * v[2:Nx + 3, 1:Ny + 1] * u_[2:Nx + 3, 1:Ny + 1] -
           rho * v[0:Nx + 1, 1:Ny + 1] * u__[0:Nx + 1, 1:Ny + 1]) / (2 * delta_x) +
          (rho * (v[1:Nx + 2, 2:Ny + 2] ** 2 - rho * (v[1:Nx + 2, 0:Ny] ** 2)) / (2 *
              delta_y)) + \
          miu * ((v[2:Nx + 3, 1:Ny + 1] - 2 * v[1:Nx + 2, 1:Ny + 1] + v[0:Nx + 1, 1:Ny
              + 1]) / (delta_x ** 2) +
                 (v[1:Nx + 2, 2:Ny + 2] - 2 * v[1:Nx + 2, 1:Ny + 1] + v[1:Nx + 2, 0:Ny])
                     / (delta_y ** 2))

    rho_v[1:Nx+2, 1:Ny+1] = rho_v[1:Nx+2, 1:Ny+1] + B * delta_t - \
             delta_t / delta_y * (p_star[0:Nx+1, 1:Ny+1] - p_star[0:Nx+1, 0:Ny])
    v[1:Nx + 2, 1:Ny + 1] = rho_v[1:Nx + 2, 1:Ny + 1] / rho

    # 2.2 Boundary conditions
    p_star[:, 0] = p_star[:, 1]
    p_star[:, -1] = p_star[:, -2]
    u[0, :] = u[1, :] # Inlet
    u[-1, :] = u[-2, :] # Outlet
    v[-1, :] = v[-2, :] # Outlet

    # Step 3: Pressure correction
    a = 2 * (delta_t / delta_x ** 2 + delta_t / delta_y ** 2)
    b = -delta_t / delta_x ** 2
    c = -delta_t / delta_y ** 2
    d = (rho_u[2:Nx + 1, 1:Ny] - rho_u[1:Nx, 1:Ny]) / delta_x + \
        (rho_v[2:Nx + 1, 2:Ny + 1] - rho_v[2:Nx + 1, 1:Ny]) / delta_y

    # Step 4: Solve pressure correction equation
    p_prime1 = np.zeros((Nx + 1, Ny + 1))
    p_prime2 = np.zeros((Nx + 1, Ny + 1))
    p_prime1[1:Nx, 1:Ny] = p_prime[1:Nx, 1:Ny]

    for _ in range(1000): # Iterative solver
        p_prime[1:Nx, 1:Ny] = -1 / a * (b * p_prime[2:Nx + 1, 1:Ny] + b * p_prime[0:Nx
            - 1, 1:Ny] +
                                        c * p_prime[1:Nx, 2:Ny + 1] + c * p_prime[1:Nx,
                                            0:Ny - 1] + d)
    p_prime2[1:Nx, 1:Ny] = p_prime[1:Nx, 1:Ny]
    e = np.sum(np.abs(p_prime2 - p_prime1))
    if e < 1e-5:
        break
    p_prime1 = p_prime2.copy()

# Step 5: Update pressure
p_star[1:Nx, 1:Ny] = p_star[1:Nx, 1:Ny] + alpha * p_prime[1:Nx, 1:Ny]
```

```
# Exact solution for comparison
y = np.linspace(0, H, Ny + 1)
u_ex = (p1 - p2) / (2 * miu * L) * y * (H - y)
u_exact = np.tile(u_ex, (Nx + 2, 1))

# Plotting
plt.figure(figsize=(15, 10))

# u velocity field
plt.subplot(2, 2, 1)
plt.pcolormesh(u.T)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Computed u velocity (SIMPLE)')
plt.colorbar()

# u velocity error
plt.subplot(2, 2, 2)
plt.pcolormesh((u_exact - u).T)
plt.xlabel('x')
plt.ylabel('y')
plt.title('u velocity error')
plt.colorbar()

# v velocity field
plt.subplot(2, 2, 3)
plt.pcolormesh(v.T)
plt.xlabel('x')
plt.ylabel('y')
plt.title('v velocity field')
plt.colorbar()

# Pressure field
plt.subplot(2, 2, 4)
plt.pcolormesh(p_star.T)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Pressure field')
plt.colorbar()

plt.tight_layout()

# Velocity profile comparison
plt.figure()
y_fine = np.linspace(0, H, 100)
u_ex_fine = (p1 - p2) / (2 * miu * L) * y_fine * (H - y_fine)
plt.plot(y_fine, u_ex_fine, '-', label='Exact')
plt.plot(y, u[0, :], 'o', label='SIMPLE')
```

```
plt.xlabel('y')
plt.ylabel('u')
plt.title('Velocity profile comparison')
plt.legend()

# Pressure distribution
plt.figure()
x_fine = np.linspace(0, L, 100)
p_ex_fine = p1 - (p1 - p2) / L * x_fine
x = np.linspace(0, L, Nx + 1)
plt.plot(x_fine, p_ex_fine, '-', label='Exact')
plt.plot(x, p_star[:, 0], 'o', label='SIMPLE')
plt.xlabel('x')
plt.ylabel('p')
plt.title('Pressure distribution comparison')
plt.legend()

plt.show()
```