神经网络解 Lorenz63 方程

MG21210021 李庆春

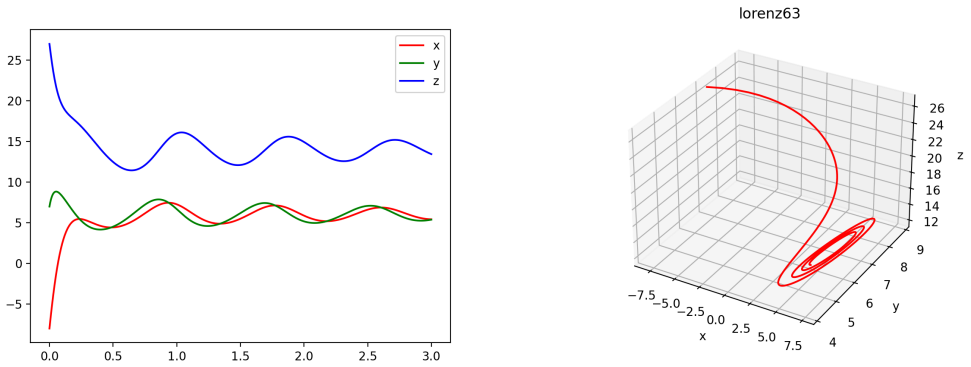
2023 年 4 月 22 日

1 Lorenz63 方程简述

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{1}$$

2 非混沌情形

$\rho = 15, \sigma = 10, \beta = \frac{8}{3}$ 初值：[-8., 7., 27.]，RK 方法的数值解如图 1：



(a) lorenz63 二维视图.

(b) lorenz63 三维视图.

图 1: lorenz63-RK

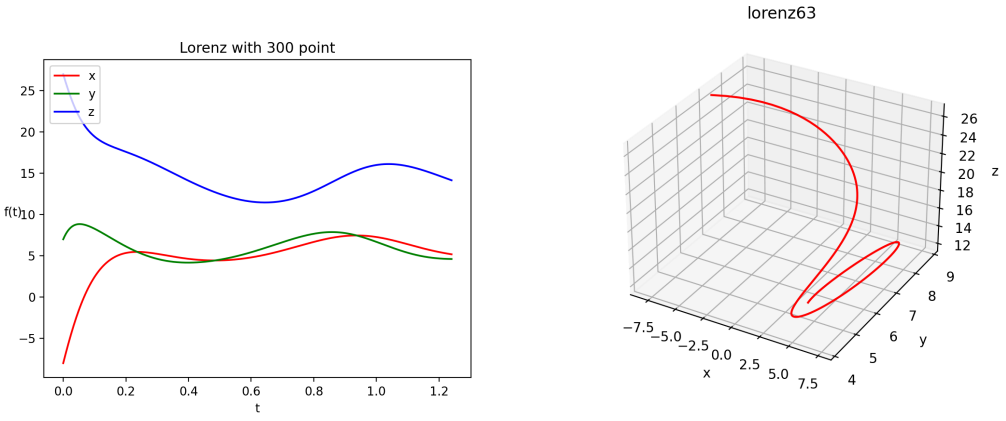
2.1 不带观测的神经网络

损失函数使用均方误差（MSE），网络结构：[1, 32, 32, 3]，初值：[-8, 7, 27]，迭代轮数：500000，对训练数据进行正规化，关键代码如下：

```
1                    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
```

```
                min_lr=1e-7, mode='min',factor=0.5,patience=50000,verbose=True)
                self.loss_func = nn.MSELoss(reduction='mean')

                layers = [1, 32, 32, 3]

                total_points = 300
                x_lb = torch.tensor(0.)
                x_ub = torch.tensor(1.24)

                rho = torch.tensor(15.0)
                sigma = torch.tensor(10.0)
                beta = torch.tensor(8.0 / 3.0)

                x_train_bc = torch.tensor([[0.]])
                y_train_bc = torch.tensor([[-8., 7., 27.]])

                epochs = 500000
```

通过调整区间长度发现，最多可训练出 [0, 1.24] 的结果，误差在 10^{-4} 量级，如图 2:



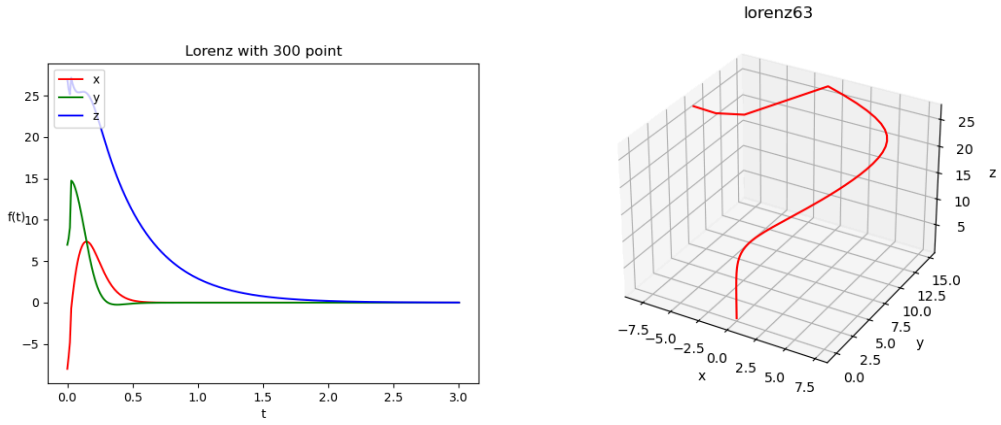
(a) lorenz63 二维视图.　　(b) lorenz63 三维视图.

图 2: lorenz63-DNN

如果区间较大，则训练不出想要的结果，神经网络会趋于零，如图 3展示的是 [0, 3] 的训练结果:

3 混沌情形

$\rho = 28, \sigma = 10, \beta = \frac{8}{3}$ ，初值 [-4., 7., 15.]，RK 方法的数值解如下图:（暂时没贴）



(a) lorenz63 二维视图.　　(b) lorenz63 三维视图.

图 3: lorenz63-DNN

3.1 整数阶 Lorenz 方程

接下来用神经网络训练，离散步长 h=0.005，区间 [0, 1.5]，观测值如下：

t	x	y	z
0.0000	-4.0000	7.0000	15.0000
0.5000	5.6789	-2.6593	33.0093
1.0000	-12.0143	-17.4640	24.3640
1.500	-1.1838	-1.6864	15.08598

3.2 分数阶 Lorenz 方程

3.2.1 Caputo 导数及数值逼近

Caputo 分数阶导数的定义：

$$ {}_aD_t^{\alpha} f(t) = \frac{1}{\Gamma(n-\alpha)} \int_a^t (t-\tau)^{n-\alpha-1} f^{(n)}(\tau) d\tau \qquad (n-1 \le \alpha < n). \tag{2} $$

当 a=0 时，简记为 D^{α}。
分数阶微积分的差商逼近格式可以写成如下统一的形式：

$$ D^{\alpha} f(t_n) \approx h^{-\alpha} \sum_{k=0}^{N} c_{n,k} f_k \tag{3} $$

L1 算法（即 $0 \le \alpha < 1$ ，上式中 N=n）：

$$ c_{n,k} = \frac{1}{\Gamma(2-\alpha)} \begin{cases} -c_{n-1}, & k = 0; \\ c_{n-k} - c_{n-k-1}, & 1 \le k \le n-1; \\ 1, & k = n; \\ 0, & else. \end{cases} \tag{4} $$

$$(D_t^\alpha f(t_n))_{L1} = \frac{1}{\Gamma(2-\alpha)h^\alpha}\sum_{k=0}^{n-1} c_k(f_{n-k} - f_{n-k-1}) \tag{5}$$

其中，$c_l = (l+1)^{1-\alpha} - l^{1-\alpha}$，fPINN 论文中另一种等价表述如下：

$$\begin{aligned}&\frac{\partial^\gamma \tilde{u}(\boldsymbol{x},t)}{\partial t^\gamma} \approx \frac{1}{\Gamma(2-\gamma)(\Delta t)^\gamma}\\ &\left\{-c_{\lceil \lambda t\rceil -1}\tilde{u}(\boldsymbol{x},0) + c_0\tilde{u}(\boldsymbol{x},t) + \sum_{k=1}^{\lceil \lambda t\rceil -1}(c_{\lceil \lambda t\rceil -k} - c_{\lceil \lambda t\rceil -k-1})\tilde{u}(\boldsymbol{x},k\Delta t)\right\},\\ &0<\gamma<1\end{aligned} \tag{6}$$

时间步长 $\Delta t = t/\lceil \lambda t\rceil \approx 1/\lambda$，$\lceil\cdot\rceil$ 是向上取整函数，表示离散化后的区间数，相当于 L1 算法公式中的 n；常量因子 λ 决定着步长，如 $\lambda = 200$，则意味着把单位 1 长度划分为 200 个小区间。由于计算机在运算时，存在舍入误差，所以 $c_0\tilde{u}(\boldsymbol{x}, n\Delta t)$ 写成 $c_0\tilde{u}(\boldsymbol{x},t)$。

(这里有疑问，t=0 的导数是不是没法计算)

所以，可先对每个训练点计算出其时间划分，然后拼成一个大向量。

如果不引入观测，训练结果的趋势是趋于 0，不符合预期；

3.2.2 分数阶常微分方程数值解法

（直接法）考虑齐次初值条件的分数阶微分方程：

$$\begin{cases}\frac{\partial^\alpha y(t)}{\partial t^\alpha} = f(t, y(t)), \quad t\in[0,T],\\ y^{(k)}(0) = 0, \quad k = 0,1,\cdots,m-1.\end{cases} \tag{7}$$

则直接应用分数阶导数的一般差商逼近公式得：

$$h^{-\alpha}\sum_{k=0}^{N} c_{n,k}y_k = f(t_n, y_n), \quad n = 0,1,\cdots,[t/h] \tag{8}$$

上式左边为 y 在 t 处的 α 阶导数，注意此时将 [0,t] 分成了 n 份，每份长度为 h，t 对应 t_n，$y(t_n)$ 对应 y_n。则上面方程组可以按下面的方式逐点计算：

$$y_N = \frac{h^\alpha}{c_{n,N}}f(t_n,y_n) - \frac{1}{c_{n,N}}\sum_{k=1}^{N-1} c_{n,k}y_k, \quad n = 1,\cdots,[T/h] \tag{9}$$

其中，N=n(对应到 G1 算法、D 算法、L1 算法、线性多步法) 或 N=n+1（对应到 G2 算法、L2 算法）。上式中，由于 $y_0 = 0$，所以求和项从 k=1 开始，对于一般的 L1 算法，可写成如下形式：

$$\begin{aligned}y_n &= \frac{h^\alpha}{c_{n,n}}f(t_n,y_n) - \frac{1}{c_{n,n}}\sum_{k=0}^{n-1} c_{n,k}y_k, \quad n = 1,\cdots,[T/h]\\ &= \Gamma(2-\alpha)h^\alpha f(t_n,y_n) - \sum_{k=1}^{n-1}(c_{n-k} - c_{n-k-1})y_k + c_{n-1}y_0\\ &= \Gamma(2-\alpha)h^\alpha f(t_n,y_n) - \sum_{k=1}^{n-1}[(n-k+1)^{1-\alpha} - 2(n-k)^{1-\alpha} + (n-k-1)^{1-\alpha}]y_k +\\ &\quad [n^{1-\alpha} - (n-1)^{1-\alpha}]y_0\end{aligned} \tag{10}$$

3.2.3 简单的例子

已知正弦、余弦函数的整数阶微分表达式分别为

$$\frac{d^k}{dt^k}[\sin at] = a^k \sin(at + \frac{k\pi}{2}), \quad \frac{d^k}{dt^k}[\cos at] = a^k \cos(at + \frac{k\pi}{2}) \tag{11}$$

由 Cauchy 积分公式可以证明，对于分数阶的微分方程来说，当 k 为分数时，上述公式仍然成立。所以考虑如下分数阶微分方程：

$$\begin{cases} \frac{d^\alpha u}{dt^\alpha} = \sin(t + \frac{\alpha\pi}{2}), & 0 < \alpha < 1 \\ u(0) = 0 \end{cases} \tag{12}$$

（这里暂时不知道解析解，所以作废）

对于 $f(t) = t^\lambda, \lambda > -1$，n 为大于 α 的最小整数，则其 α 阶 Caputo 导数为：

$$\begin{aligned} D^\alpha t^\lambda &= \frac{1}{\Gamma(n-\alpha)} \int_0^t (t-\tau)^{n-\alpha-1} \frac{d^n \tau^\lambda}{d\tau^n} d\tau \\ &= \frac{1}{\Gamma(n-\alpha)} \int_0^t (t-\tau)^{n-\alpha-1} \lambda(\lambda-1)\cdots(\lambda-n+1) t^{\lambda-n} d\tau \\ &= \frac{\Gamma(\lambda+1)}{\Gamma(n-\alpha)\Gamma(\lambda-n+1)} \int_0^t (t-\tau)^{n-\alpha-1} t^{\lambda-n} d\tau \\ &= \frac{\Gamma(\lambda+1)}{\Gamma(n-\alpha)\Gamma(\lambda-n+1)} t^{\lambda-\alpha} \int_0^1 (1-\tau)^{n-\alpha-1} t^{\lambda-n} d\tau \quad (substitution) \\ &= \frac{\Gamma(\lambda+1)}{\Gamma(n-\alpha)\Gamma(\lambda-n+1)} t^{\lambda-\alpha} B(n-\alpha, \lambda-n+1) \\ &= \frac{\Gamma(\lambda+1)}{\Gamma(n-\alpha)\Gamma(\lambda-n+1)} t^{\lambda-\alpha} \frac{\Gamma(n-\alpha)\Gamma(\lambda-n+1)}{\Gamma(\lambda-\alpha+1)} \\ &= \frac{\Gamma(\lambda+1)}{\Gamma(\lambda-\alpha+1)} t^{\lambda-\alpha} \end{aligned} \tag{13}$$

这和 R-L 导数一致，这是由于 $D^n t^\lambda$ 在区间 $[0, +\infty)$ 上连续，且 $D^k f(0) = 0, k = 0, 1, \cdots, n-1$，所以 D^n 和 $D^{-\nu}$ 可交换。

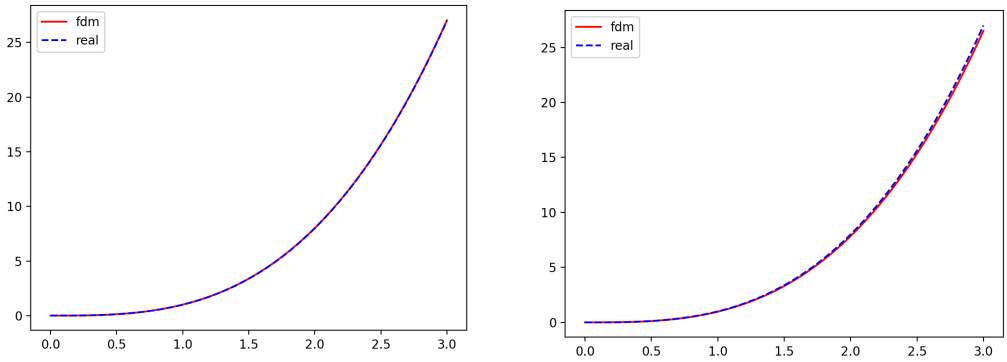
例子：考虑 $y = t^3$，则对应的分数阶微分方程为：

$$\begin{cases} D^\alpha t^3 = \frac{6}{\Gamma(4-\alpha)} t^{3-\alpha} \\ y(0) = 0 \end{cases} \tag{14}$$

写成隐式的：

$$\begin{cases} D^\alpha u = \frac{6u}{\Gamma(4-\alpha)} t^{-\alpha} \\ u(0) = 0.000001 \end{cases} \tag{15}$$

数值计算结果如图 4：(这里有疑问，时间是不是一定要从 0 开始，变成时发现如果不是 0 作为离散的初始点，则得不到结果)



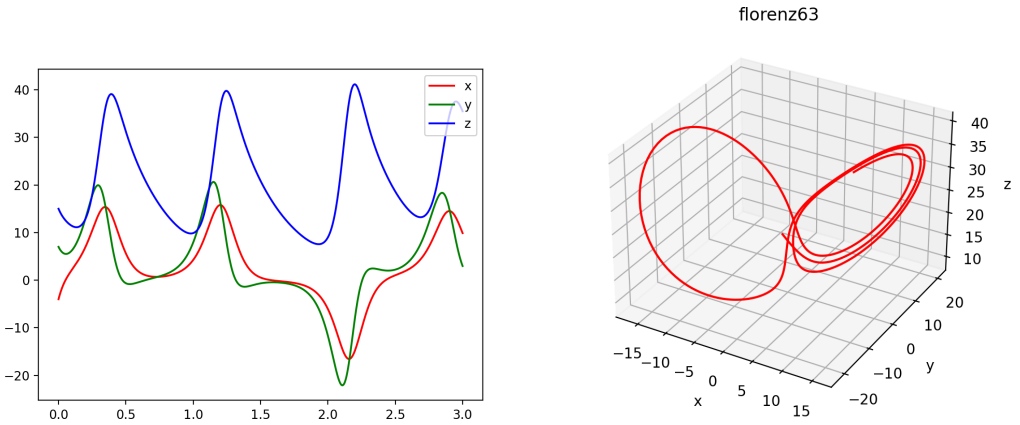
(a) 显示　　(b) 隐式

图 4: ODE-FDM

3.2.4 分数阶 Lorenz63 方程

$$\begin{cases} \frac{d^\alpha x}{dt^\alpha} = \sigma(y-x) \\ \frac{d^\alpha y}{dt^\alpha} = x(\rho-z)-y \\ \frac{d^\alpha z}{dt^\alpha} = xy-\beta z \end{cases} \tag{16}$$

取 $\alpha = 0.99$，离散步长 h=0.005，区间 [0, 3]，隐式收敛阈值 $\epsilon = 0.001$，初值 [-4., 7., 15.]，数值计算结果如图 5：



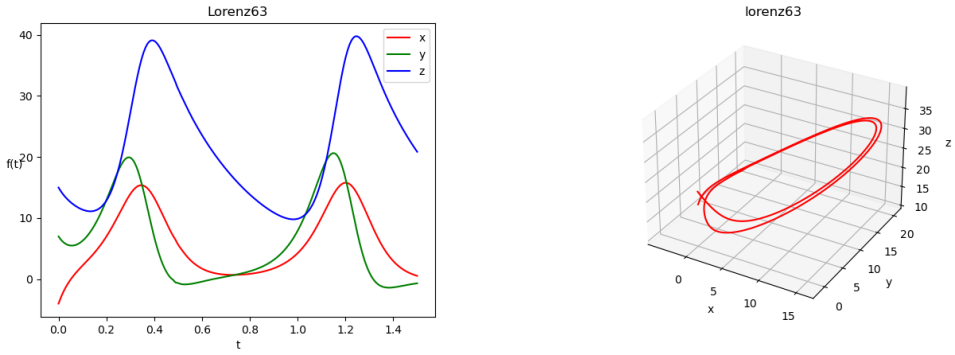
(a) 分数阶 lorenz63-2D　　(b) 分数阶 lorenz63-3D

图 5: frac-lorenz63-FDM

接下来用神经网络训练，损失函数使用 MSE，先用参数 $\alpha = 0.99$，离散步长 h=0.005，区间 [0, 1.5]，观测值如下：

t	x	y	z
0.0050	-3.4659	6.7226	14.6728
0.2500	10.4394	17.7745	17.9601
0.5000	5.6236	-0.6432	31.2168
0.7500	0.7111	0.8086	15.8068
1.0000	4.2548	7.9222	9.8927
1.2500	13.9364	7.5091	39.7435
1.5000	0.5546	-0.6693	20.8581

训练结果如图 6，符合预期：



(a) 分数阶 lorenz63-2D　　(b) 分数阶 lorenz63-3D

图 6: frac-lorenz63-FDM

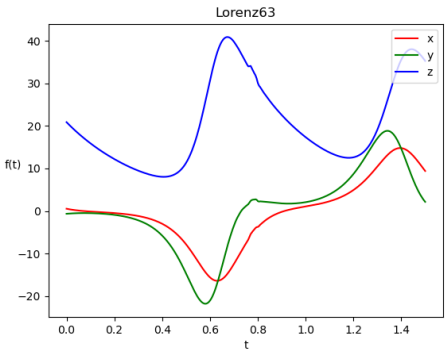
继续训练 [1.5, 3]，观测数据如下，注意此时 t=0 对应的是 t=1.5：

t	x	y	z
0.0050	0.4947	-0.6466	20.5689
0.2500	-0.7559	-1.3882	10.706
0.5000	-7.8296	-14.6883	11.2953
0.7500	-7.6905	1.1865	34.9326
1.0000	1.0532	2.0665	17.3576
1.2500	7.2898	12.7107	14.2827
1.5000	9.4016	2.1509	35.2638

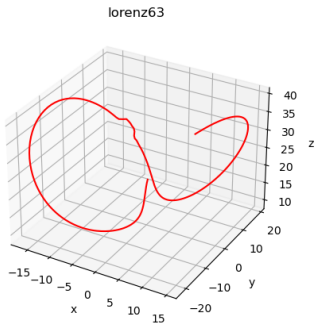
训练结果如图 7，除个别点尖锐突变，根据几次训练，突变点在 [0.4,0.8] 之间，有时是个小尖角，其他部分符合预期：所以继续对 [0.4,0.8] 增加一些观测值，如下：

t	x	y	z
0.4000	-3.0048	-5.7771	8.0258
0.6000	-15.5885	-21.0788	30.5955
0.7000	-12.4916	-3.5173	39.8045
0.8000	-3.9029	2.2364	29.9654

得到的训练结果如图 8：继续增加观测值，变为每 0.1 的长度，引入一个观测值，训练结果如

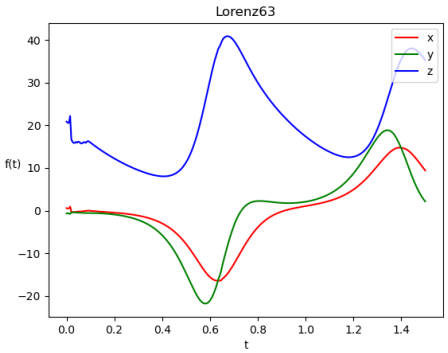


(a) 分数阶 lorenz63-2D

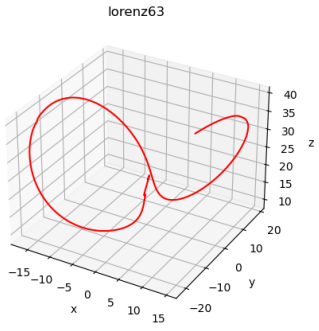


(b) 分数阶 lorenz63-3D

图 7: frac-lorenz63-FDM



(a) 分数阶 lorenz63-2D



(b) 分数阶 lorenz63-3D

图 8: frac-lorenz63-FDM

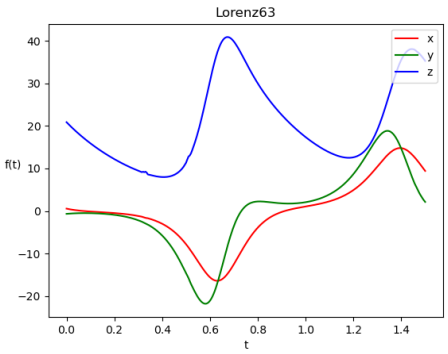
图 9:

4 Burgers 方程

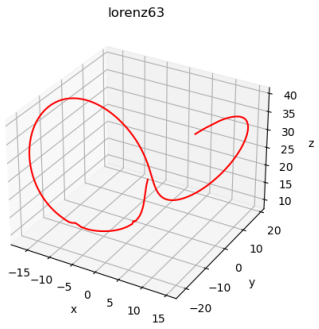
$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \\ x \in [-1, 1] \\ t \in [0, 1] \end{cases} \tag{17}$$

5 TODO

lorenz63 时间长的时候，趋于 0，什么原因?
pinn 解 PDE 的论文，多看看，多总结，作为自己毕设素材
GL 算法实现



(a) 分数阶 lorenz63-2D



(b) 分数阶 lorenz63-3D

图 9: frac-lorenz63-FDM

神经网络解 burgers

lorenz63-dnn-obs-01 [0,3] 训练每个 0.02 一个观测值，观测值由 RK 法计算，初值为: [1.508870, -1.531271, 25.46091] ，训练不收敛

lorenz63-dnn-obs-02 [0,1.5] lorenz63-dnn-obs-03 [0,3] 观测值为 iobsdisturb.txt ，导师发是数据;

lorenz63-dnn-12 [0,3] 尝试分段 mse，意图为规避训练区间加长，后续趋于 0 ，失败，仍然趋于 0。换个思路，把三个导数的平方和相加，取导数加入损失函数。失败

lorenz63-dnn-13 使用 L1Smooth + 无穷范数，失败

6 DeepONet

6.1 理论背景

一般的函数:

$$z = f_1(x) = sin(x) \in [-1, 1] \tag{18}$$

算子:

$$G(f_1(x)) = f_2(x) \tag{19}$$

比如微分算子:

$$f_2 = \frac{df_1(x)}{dx} = \frac{d}{dx} sin(x) = cos(x) \tag{20}$$

如果我们的 PDE 里面有参数（比如形状、初边值、扩散系数等），这里统一用 u(x) 表示，我们是允许它变化的。这个时候，每个 u 对应一个真解 s；参数化的 PDE 可写成如下形式:

$$\mathcal{N}(u, s) = 0 \tag{21}$$

PDE 的解算子写成（此时，我们可以把 PDE 的一般解表示为算子 G）:

$$G(u) = s \tag{22}$$

此时，u，s 均为函数，并且满足：

$$G(u)(y) = s(y) \in \mathbf{R} \tag{23}$$

算子的通用逼近定理：
$\forall \epsilon > 0$, 存在正整数 n, p, m, 常量 $c_i^k, W_{bij}^k, b_{bij}^k, W_{tk}, b_{tk}$ 使得:

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} W_{bij}^k u(x_j) + b_{bi}^k \right)}_{Branch} . \underbrace{\sigma(W_{tk}.y + b_{tk})}_{Trunk} \right| < \epsilon \tag{24}$$

神经网络：

$$NN(X) = W_n \sigma_{n-1}(W_{n-1}\sigma_{n-2}(...(W_2\sigma_1(W_1X + b_1) + b_2) + ..) + b_{n-1}) + b_n \tag{25}$$

所以说，我们可以用两个神经网络去逼近算子，其中分支网络为：

$$NN_b(u(\mathbf{x})) = b(u(\mathbf{x})) = \mathbf{c}.\sigma\left(W_b u(\mathbf{x}) + \mathbf{b}_b\right) \tag{26}$$

主干网络：

$$NN_t(\mathbf{y}) = t(\mathbf{y}) = \sigma(W_t.\mathbf{y} + \mathbf{b}_t) \tag{27}$$

DeepONet（注意：θ 是网络参数，即 weight 和 bias）

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k\left(u(x_1), u(x_2), ..., u(x_m)\right)}_{Branch} . \underbrace{t_k(\mathbf{y})}_{Trunk} \tag{28}$$

神经网络的输出应该逼近真解算子

$$G_\theta(u)(y) \approx G(u)(y) \tag{29}$$

我们把这个约束加入损失函数

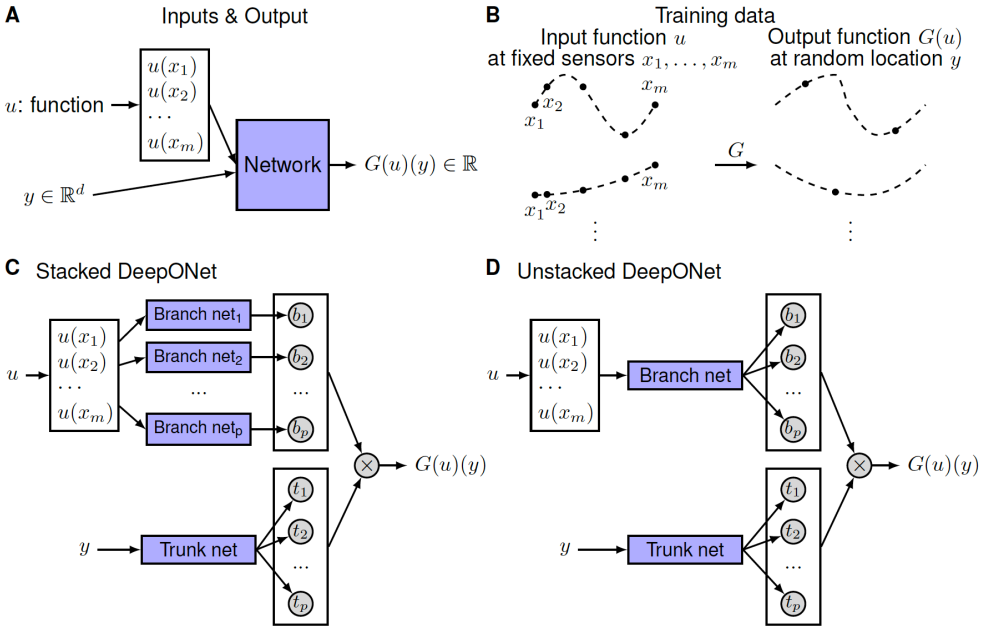
$$\mathcal{L}_{Operator}(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)}) y_j^{(i)} - G(u^{(i)}) y_j^{(i)} \right|^2 \tag{30}$$

$$\mathcal{L}_{Operator}(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| \sum_{k=1}^{q} b_k\left(u^{(i)}(x_1), u^{(i)}(x_2), ..., u^{(i)}(x_m)\right) . t_k(y_j^{(i)}) - G(u^{(i)}) y_j^{(i)} \right|^2 \tag{31}$$

Physics-informed DeepONets

$$\mathcal{L}_{Physics}(\theta) = \frac{1}{NQm} \sum_{i=1}^{N} \sum_{j=1}^{Q} \sum_{k=1}^{m} \left| \mathcal{N}(u^{(i)}(x_k), G_\theta(u^{(i)})(y_j^{(i)}) \right|^2 \tag{32}$$

$$\mathcal{L}(\theta) = \mathcal{L}_{Operator}(\theta) + \mathcal{L}_{Physics}(\theta) \tag{33}$$



(a) DeepONet 架构.

图 10: DeepONet 架构

6.2 示例：反应扩散方程

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x) \qquad (x,t) \in (0,1] \times (0,1] \tag{34}$$

其中，扩散系数 D=0.01，反应速率 k=0.01；u(x) 是源项，初边值条件为 0；（可以看出 PI-DON 不需要太多的训练数据）

对于任意给的 $u^{(i)}$，我们有：

$$u^{(i)} = \frac{\partial s^{(i)}}{\partial t} - D\frac{\partial^2 s^{(i)}}{\partial x^2} - k[s^{(i)}]^2 \tag{35}$$

理想情况下，我们的 DeepONet 拟合的算子满足：

$$G_\theta(u^{(i)})(x,t) \approx G(u^{(i)})(x,t) = s^{(i)}(x,t) \tag{36}$$

$$u^{(i)} \approx \frac{\partial G_\theta(u^{(i)})(x,t)}{\partial t} - D\frac{\partial^2 G_\theta(u^{(i)})(x,t)}{\partial x^2} - k[G_\theta(u^{(i)})(x,t)]^2 \tag{37}$$

我们记右端项

$$R_\theta^{(i)}(x,t) = \frac{\partial G_\theta(u^{(i)})(x,t)}{\partial t} - D\frac{\partial^2 G_\theta(u^{(i)})(x,t)}{\partial x^2} - k[G_\theta(u^{(i)})(x,t)]^2 \tag{38}$$

则物理模型损失：

$$\mathcal{L}_{Physics}(\theta) = \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{Q}\left|R_\theta^{(i)}(x_{r,j}^{(i)}, t_{r,j}^{(i)}) - u^{(i)}(x_{r,j}^{(i)})\right|^2 \tag{39}$$

其中，$(x_{r,j}, t_{r,j})$ 是配置点；
另一方面，用零初边值条件计算算子损失：

$$\mathcal{L}_{Operator}(\theta) = \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{P}\left|G_\theta(u^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)}) - G(u^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)}))\right|^2 \tag{40}$$

其中，$(x_{u,j}, t_{u,j})$ 是初边值条件上的点；因为这里是零初边值，所以

$$G(u^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)}) = 0 \tag{41}$$

$$\mathcal{L}_{Operator}(\theta) = \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{P}\left|G_\theta(u^{(i)})(x_{u,j}^{(i)}, t_{u,j}^{(i)}))\right|^2 \tag{42}$$

最终，损失函数为：

$$\mathcal{L}(\theta) = \mathcal{L}_{Ooperator}(\theta) + \mathcal{L}_{Physics}(\theta) \tag{43}$$

6.2.1 训练算法

1. 选择 N 个源项，也就是输入函数，

$$\{u^{(i)}(\mathbf{x})\}_{i=1}^{N} = [u^{(1)}(\mathbf{x}), u^{(2)}(\mathbf{x}), ..., u^{(N)}(\mathbf{x})] \tag{44}$$

2. 在 m 个点（即输入张量）评估 N 个输入函数

$$\begin{bmatrix} u^{(1)}(x_1) & u^{(1)}(x_2) & ... & u^{(1)}(x_m) \\ u^{(2)}(x_1) & u^{(2)}(x_2) & ... & u^{(2)}(x_m) \\ \vdots & \vdots & \ddots & \vdots \\ u^{(N)}(x_1) & u^{(N)}(x_2) & ... & u^{(N)}(x_m) \end{bmatrix} \tag{45}$$

3. 将其送入分支网络，得到

$$b_k \begin{bmatrix} u^{(1)}(x_1) & u^{(1)}(x_2) & ... & u^{(1)}(x_m) \\ u^{(2)}(x_1) & u^{(2)}(x_2) & ... & u^{(2)}(x_m) \\ \vdots & \vdots & \ddots & \vdots \\ u^{(N)}(x_1) & u^{(N)}(x_2) & ... & u^{(N)}(x_m) \end{bmatrix} \tag{46}$$

4. 从初边值选择 P 个点（即输出张量）

$$\mathbf{y}_u = y_{u1}, y_{u2}, ..., y_{uP} \tag{47}$$

5. 将其送入主干网络，得到

$$t_k(y_{u1}, y_{u2}, ..., y_{uP}) \tag{48}$$

6. 将二者输出做点积，得到近似的算子

$$G_\theta(u)(\mathbf{y}) = \sum_{k=1}^{q} \underbrace{b_k\left(u(x_1), u(x_2), ..., u(x_m)\right)}_{Branch} . \underbrace{t_k(\mathbf{y})}_{Trunk} \tag{49}$$

7. 理想情况下，$G_\theta(u)(\mathbf{y}_u) \approx G(u)(\mathbf{y}_u) = 0$ ，所以只需计算损失：

$$\mathcal{L}_{Operator}(\theta) = \frac{1}{NP}\sum_{i=1}^{N}\sum_{j=1}^{P}\left|G_\theta(u^{(i)})y_{uj}^{(i)}\right|^2 \tag{50}$$

8. 从定义域中随机选 Q 个配置点

$$\mathbf{y}_r = y_{r1}, y_{r2}, ..., y_{rQ} \tag{51}$$

9. 使用物理信息（PDE），计算配置点的损失

$$\mathcal{L}_{Physics}(\theta) = \frac{1}{NQ}\sum_{i=1}^{N}\sum_{j=1}^{Q}\left|R_\theta^{(i)}(y_{r,j}^{(i)}) - u^{(i)}(x_{r,j}^{(i)})\right|^2 \tag{52}$$

10. 计算总损失

$$\mathcal{L}(\theta) = \mathcal{L}_{operator}(\theta) + \mathcal{L}_{Physics}(\theta) \tag{53}$$

11. 更新神经网络参数使损失最小化

12. 重复上述过程直至 $G_\theta(u)(x,t) \approx G(u)(x,t)$

6.2.2 代码中涉及的知识

高斯径向基函数，是一种高斯核函数，用于计算两个时刻的高斯变量的协方差，用 RBF 得到的是一个半正定矩阵，注意：向量和自身做 RBF 的运算，得到的不一定是单位阵。而且这个核函数体现不出负相关性，如果要体现负相关性，用其他的核函数；

$$k(x, x') = \sigma^2 exp(-\frac{||x - x'||^2}{2l^2}) \tag{54}$$

Cholesky 分解

Cholesky 分解是把一个对称正定的矩阵表示成一个下三角矩阵 L 和其转置的乘积的分解。它要求矩阵的所有特征值必须大于零，故分解的下三角的对角元也是大于零的。Cholesky 分解法又称平方根法。Cholesky 分解通常用于解决线性方程组或生成随机数。

jax.numpy.linalg.cholesky 方法返回的是下三角矩阵，称为 Cholesky 因子。

对于一个正定矩阵 A，可以通过 Cholesky 分解将其转换为 $A = LL^T$ 的形式，然后利用标准正态分布的随机数生成方法，即使用均值为 0，方差为 1 的正态分布生成随机数，生成一个 n 维向量 z，其中 n 是矩阵 A 的维度。

然后，通过矩阵向量乘法，即 $y = Lz$，可以得到一个满足多元正态分布 $N(0, A)$ 的 n 维向量 y。

因此，通过 Cholesky 分解可以将生成满足多元正态分布的随机数的过程转化为简单的矩阵运算，这在许多机器学习和统计模型中都是非常有用的。

为什么使用 Cholesky 分解的时候，往往会对正定矩阵做一点扰动?

在数值计算中，对于一个给定的正定矩阵，它的 Cholesky 分解不是唯一的。这意味着，如果两次使用 Cholesky 分解来生成随机数，即使使用相同的输入矩阵，也有可能得到不同的结

果。
这是由于在使用 Cholesky 分解时，由于分解过程中需要进行平方根运算，当某些元素非常小甚至为零时，平方根运算会出现不稳定的情况，从而导致分解失败或者得到错误的结果。增加一个较小的扰动可以使得这些元素不再为零，从而避免这种不稳定性。
如果输入矩阵有某些特定的结构或数字精度问题，这种不确定性可能会增加，甚至导致错误的结果。为了解决这个问题，可以对输入矩阵进行微小的扰动，以确保得到稳定的 Cholesky 分解结果。
一种常用的扰动方法是在对角线上添加一个较小的正数 ϵ，这样可以确保输入矩阵的条件数（矩阵奇异值的比值）不会太大，从而使得 Cholesky 分解结果更加稳定。

数值解法
代码中的函数，是一个求解一般的一维扩散反应方程的程序，主要用到了高斯过程插值和有限差分的方法。以下是对代码的注释和说明：

1. 在程序开始时，定义了微分方程的各项系数和边界条件：

$$\begin{cases} u_t = (k(x)u_x)_x - (v(x)u)_x + g(u) + f(x) \\ k(x) = 0.01 \\ v(x) = 0 \\ g(u) = 0.01u^2 \\ f(x) = GausianProcessStochasticFunction \end{cases} \tag{55}$$

2. 在生成随机函数的时候，使用了高斯过程的方法。先选取一些离散点，计算协方差矩阵，然后进行 Cholesky 分解，得到一个随机向量，再通过一维分段线性插值得到一个连续的函数。

3. 利用有限差分离散微分方程，得到一个三对角矩阵，然后求解线性方程组得到下一时刻的解。

4. 时间上采用了显式的欧拉方法。

5. 整个求解过程是以矩阵形式实现的，使用了 JAX 库进行了自动微分和并行加速。

二阶导的中心差分：

$$\frac{d^2u_i}{dx^2} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} \tag{56}$$

$$\frac{d^2\mathbf{u}}{dx^2} \approx \frac{1}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 & 0 \\ 0 & 1 & -2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -2 & 1 \\ 0 & 0 & 0 & \cdots & 1 & -2 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{1}{\Delta x^2} D^2 \cdot \mathbf{u} \tag{57}$$

一阶导的中心差分：

$$\frac{du_i}{dx} \approx \frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{58}$$

$$\frac{d\mathbf{u}}{dx} \approx \frac{1}{2\Delta x} \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & -1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & -1 & 0 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{1}{2\Delta x} D^1 \cdot \mathbf{u} \tag{59}$$

代码拆解：

$$\begin{aligned} (k(x)u_x)_x &= k_x u_x + k u_{xx} \\ &\approx \frac{k_{i+1} - k_{i-1}}{2\Delta x} \cdot \frac{u_{i+1} - u_{i-1}}{2\Delta x} + k_i \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2} \\ &= \frac{1}{4\Delta x^2} [D^1 \cdot \mathbf{k} \cdot D^1 \cdot \mathbf{u} + 4 \cdot diag(\mathbf{k}) \cdot D^2 \cdot \mathbf{u}] \\ &= \frac{1}{4\Delta x^2} [diag(D1 \cdot \mathbf{k}) \cdot D1 + 4 \cdot diag(\mathbf{k}) \cdot D2] \cdot \mathbf{u} \end{aligned} \tag{60}$$

注意：这个差分格式对应着代码中的 $M_{1:-1,1:-1}$，而不是 M，因为代码中的 M 是 $N \times N$，而实际做差分，边值为 0，所以边界点不考虑，差分矩阵应该是 $(N-2) \times (N-2)$。
(这里有疑问，为什么要一开始取 $N \times N$ 的矩阵，后续都是使用其 $(N-2) \times (N-2)$ 的子矩阵，为什么不一开始就构造 $(N-2) \times (N-2)$？)

$$\begin{aligned} (v(x)u)_x &= v u_x + v_x u \\ &\approx v_i \cdot \frac{u_{i+1} - u_{i-1}}{2\Delta x} + \frac{v_{i+1} - v_{i-1}}{2\Delta x} \cdot u_i \\ &= \frac{1}{2\Delta x} (\mathbf{v} \cdot D^1 \cdot \mathbf{u} + D^1 \cdot \mathbf{v} \cdot \mathbf{u}) \end{aligned} \tag{61}$$

$$u_t \approx \frac{u_i^{n+1} - u_i^n}{\Delta t} \tag{62}$$

考虑如下方程的差分格式

$$u_t = (k(x)u_x)_x - (v(x)u)_x \tag{63}$$

$$\begin{aligned} u_i^{n+1} &= u_i^n + \Delta t \left\{ \frac{k_{i+1} - k_{i-1}}{2\Delta x} \cdot \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + k_i \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} \right. \\ &\quad \left. - v_i \cdot \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} - \frac{v_{i+1} - v_{i-1}}{2\Delta x} \cdot u_i^n \right\} \\ \mathbf{u}^{n+1} &= \mathbf{u}^n + \frac{\Delta t}{4(\Delta x)^2} \{ D^1 \cdot \mathbf{k} \cdot D^1 \cdot \mathbf{u}^n + 4 \cdot diag(\mathbf{k}) \cdot D^2 \cdot \mathbf{u}^n \\ &\quad - 2\Delta x \cdot diag(\mathbf{v}) \cdot D^1 \cdot \mathbf{u}^n - 2\Delta x \cdot D^1 \cdot \mathbf{v} \cdot \mathbf{u}^n \} \\ &= \frac{\Delta t}{4(\Delta x)^2} \left\{ \frac{4(\Delta x)^2}{\Delta t} I_n + D^1 \cdot \mathbf{k} \cdot D^1 + 4 \cdot diag(\mathbf{k}) \cdot D^2 \right. \\ &\quad \left. - 2\Delta x \cdot diag(\mathbf{v}) \cdot D^1 - 2\Delta x \cdot D^1 \cdot \mathbf{v} \right\} \cdot \mathbf{u}^n \end{aligned} \tag{64}$$

考虑如下方程的差分格式：（扩散系数 D=0.01，反应速率 k=0.01；f(x) 是源项）

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ku^2 + f(x) \qquad (x,t) \in (0,1] \times (0,1] \tag{65}$$

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = D \cdot \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2} + k(u_i^n)^2 + f_i$$

$$u_i^{n+1} = u_i^n + \Delta t \cdot \left\{ D \cdot \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{(\Delta x)^2} + k(u_i^n)^2 + f_i \right\} \tag{66}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \left\{ \frac{D}{(\Delta x)^2} \cdot D^2 \cdot \mathbf{u}^n + k(\mathbf{u}^n)^2 + F \right\}$$

这里需要注意的是，每次迭代完 $\mathbf{u}^{n+1}$ 的第 0 个元素和第 N 个元素是不对的，此时用边值条件重置下。

$$1 \tag{67}$$

$$= \frac{1}{4\Delta x^2}(2\Delta x \cdot diag(v_{1:-1}) \cdot D1_{1:-1,1:-1} + 2\Delta x \cdot diag(v_{2:} - v_{:N_x-2}) \cdot \mathbf{u} \tag{68}$$

$$1 \tag{69}$$

$$M = -diag(D1 \cdot \mathbf{k}) \cdot D1 - 4 \cdot diag(\mathbf{k}) \cdot D2 \tag{70}$$

$$m_{bond} = 8 \cdot \frac{\Delta x^2}{\Delta t} \cdot D_3 + M_{1:-1,1:-1} \tag{71}$$

$$v_{bond} = 2 \cdot h \cdot diag(v_{1:-1}) \cdot D1_{1:-1,1:-1} + 2 \cdot h \cdot diag(v_{2:} - v_{:N_x-2}) \tag{72}$$

$$mv_{bond} = m_{bond} + v_{bond} \tag{73}$$

$$c = 8 \cdot \frac{h^2}{\Delta t} \cdot D_3 - M_{1:-1,1:-1} - v_{bond} \tag{74}$$

bond 是指偏微分方程的边界条件。其中 m_{bond} 和 v_{bond} 分别计算了扩散项和反应项的边界条件，mv_{bond} 计算了边界条件的总和，c 则是经过边界条件修正后的方程系数。

$$1 \tag{75}$$

$$1 \tag{76}$$

$$1 \tag{77}$$

$$1 \tag{78}$$

$$1 \tag{79}$$

$$1 \tag{80}$$

$$1 \tag{81}$$

代买迁移的坑

要设置精度，jax 默认的事 32 位，会导致精度不够，产生 NaN

jax 对象转成 numpy 对象，否则做差分的时候一些便捷操作用不了，$dt = jax.device\_get(dt)$

CFL 条件

$$\Delta t \leq \frac{\Delta x^2}{2D} \tag{82}$$

其中 Δt 是时间步长，Δx 是空间步长，D 是扩散系数，保证该条件可以保证数值解的稳定性。

6.3 示例：反应扩散方程-初值问题

工作站训练数据：

```
# 随机生成的函数个数
stoch_func_num = 5000

# 构建网络网络结构
branch_layers = [100, 64, 64, 64, 64, 64]
trunk_layers = [2, 64, 64, 64, 64, 64]

# 初值训练点个数
ic_train_num = 100
# 边值训练点个数
each_bc_train_num = 100
# 物理信息训练点个数
physics_train_num = 200
batch_size = 10000

batch : 54519 lr : 0.0005 loss : 9.648224659031257e-05
Elapsed time:  2:37:43.443680
```

工作站存储的模型文件是：adr_don_ic_01_gzz_02.pt，相对误差 10^{-2} 量级。
本地训练的结果：网络层数少一点，损失大一点，物理信息取点少一点。在做预测的时候，相对误差也差不太多。
精度要想提升，一方面可以继续训练，我觉得更主要的一方面应该在初值函数的选取。目前代码逻辑初值函数是随机正态分布的函数值，然后把两端设置为 0，所以靠近两端梯度过大。所以看数值解和神经网络解在初值地方，有差异；

具体来说，假设我们有一个 n 维的常微分方程组：

$$\frac{d\boldsymbol{u}}{dt} = \boldsymbol{f}(\boldsymbol{u}, t; \boldsymbol{\theta}) \quad \boldsymbol{u}(t_0) = \boldsymbol{u}_0(\boldsymbol{\xi}) \tag{83}$$

其中 $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)^T$，$\boldsymbol{\theta}$ 是参数，$\boldsymbol{\xi}$ 是初始值的参数。我们可以将常微分方程组表示为微分算子 $\mathcal{N}$ 作用在 $\boldsymbol{u}$ 上的形式：

$$\mathcal{N}(\boldsymbol{u}, t; \boldsymbol{\theta}) = \frac{d\boldsymbol{u}}{dt} - \boldsymbol{f}(\boldsymbol{u}, t; \boldsymbol{\theta}) \tag{84}$$

然后，我们可以用算子神经网络来逼近微分算子 $\mathcal{N}$，形式为：

$$\mathcal{N}_\theta(\boldsymbol{u}, t) \approx \mathcal{N}(\boldsymbol{u}, t; \boldsymbol{\theta}) \tag{85}$$

同样地，我们可以用神经网络来逼近初始条件，形式为：

$$\boldsymbol{u}_\theta(t_0; \boldsymbol{\xi}) \approx \boldsymbol{u}(t_0) = \boldsymbol{u}_0(\boldsymbol{\xi}) \tag{86}$$

最后，我们可以将整个常微分方程组表示为以下形式：

$$\mathcal{N}\theta(\boldsymbol{u},t)=0\ \boldsymbol{u}(t_0)-\boldsymbol{u}\theta(t_0;\boldsymbol{\xi}) \qquad =0 \tag{87}$$

我们可以通过最小化上述形式中的损失函数来训练神经网络，从而求解参数化的常微分方程组。