基于神经网络的数据同化研究

MG21210021 李庆春

2024 年 3 月 4 日

1 绪论

文献 [2] 提出了一种新型的微分方程数值解法，RFM 的全称是 The Random Feature Method，意为随机特征方法。目的是解决经典数值方法和机器学习方法的痛点，即经典的微分方程数值方法通常具有稳定的收敛阶，但依赖于网格离散的特性让它们难以处理复杂几何上的问题；而机器学习方法在这类复杂问题上具有优势，但求解误差不可控，且相比于经典方法需要很长的求解时间。作为一种新型的微分方程数值方法，融合了经典方法和机器学习方法的优势，是一种兼具稳定收敛性和简便性的新型微分方程数值方法。

科学计算中最古老且研究最广泛的主题之一是用于解决偏微分方程（PDEs）的算法。已经提出并广泛研究了有限差分 [13]、有限元 [25]、谱方法 [20] 以及许多其他方法，并取得了巨大成功。与此同时，基于这些方法的各种科学软件已经被开发出来，并广泛应用于学术界和工业界。它们已经成为几乎所有工程应用中的标准资源。

近年来，随着神经网络模型在各种人工智能（AI）任务中取得了巨大成功，将这些模型用于解决 PDEs 的想法变得非常流行 [5, 7, 9, 18, 22, 24]。虽然早在 90 年代，就已经提出了在 PDE 求解器中使用神经网络作为测试或试验函数的想法 [12]，但最近的提议通常具有一些非常重要的新变化。最显著的成功是解决高维度的 PDEs 和控制问题 [5, 8, 9, 22]，这是传统算法无法处理的问题类别。事实上，基于深度学习的算法现在已经使得在数百维甚至更高维度中解决大类 PDEs 变得相当常见 [6]，而这在几年前是不可能的。在另一个方向上，神经网络还可以用于参数化解决方案。

尽管付出了大量努力并取得了巨大成功，但解决 PDEs 的情况甚至对于一些传统工程问题来说仍然不完全令人满意。以下是我们仍然遇到的一些困难的不完全列表：

复杂几何问题。典型问题是多孔介质中的斯托克斯流 [1]。原则上，有限元方法（FEM）非常适用于具有复杂几何形状的问题。但在实际中，找到一个合适的网格通常是一项非常复杂的任务，无论从人力投入还是实际计算成本的角度来看。基于机器学习的算法虽然容易编码，但在实际情况中尚未证明在与传统算法的竞争中具备可靠性。

动力学方程。尽管其维度远低于上述高维问题，但动力学方程，如玻尔兹曼方程，传统上被视为高维问题，传统方法在处理这些问题时确实遇到了困难。基于稀疏网格的思想应该有所帮助 [2, 21]，但目前解决动力学方程的最流行方法仍然是直接模拟蒙特卡洛算法（DSMC）[23]。DSMC 的一个问题是其产生的解包含太多噪音。

多尺度问题。示例包括涉及化学动力学的问题，通常涵盖了大范围的时间尺度；完全发展的湍流流动包含大范围的空间和时间尺度；以及复合材料的建模；参见 [4]。

本文的目标是提出一种解决通用 PDEs 的方法，该方法既具有传统方法的优点，又兼具基于机器学习的算法的优势。这一新类算法可以实现谱精度。同时，它们也是无网格的，因此即使在具有复杂几何形状的情况下，也易于使用。我们的出发点是基于一系列相当简单而众所周知的思想的组合：我们使用随机特征函数来表示近似解，用最小二乘法处理 PDE 以及边界条件，并采用重新缩放程序来平衡损失函数中来自 PDE 和边界条件的贡献。在实际实现中，我们汲取了机器学习文献中的一些灵感。

经典数值方法：线性系统往往行列数相等（条件个数 = 自由度个数）通常具有稳定的收敛阶（在 log-log 误差图中误差线性下降）依赖网格，难以处理复杂的计算区域无法求解高维问题（存在维数灾难）无法求解反问题（不可微分框架）

机器学习方法：线性系统行列数可以不相等（条件个数　自由度个数）不依赖网格，在处理复杂区域时有显著优势能够求解极其高维的微分方程，甚至可以用于解算子的参数化可微分框架，能够在同一框架下求解反问题需要长时训练，方程求解时间长非凸优化导致方程数值解的精度有限，无法系统性优化边界罚项中罚参数的调整困难

随机特征方法（RFM）：线性系统行列数可以不相等（条件个数　自由度个数）具有谱精度（在 semi-log 误差图中误差线性下降）不依赖网格，在处理复杂区域时有显著优势线性最小二乘优化框架，求解精度高、效率高可微分框架，能够在同一框架下求解反问题边界罚项中罚参数的调整容易

书籍 [7] 中对数据同化给出定义：

2 预备知识

2.1 Lorenz63 方程及龙格-库塔法

2.1.1 Lorenz63 方程基本概念

Lorenz63 方程是由物理学家 Edward Lorenz 于 1963 年提出的一组非线性常微分方程，用于描述大气对流中的混沌现象。这组方程形成了混沌理论的基础之一。方程的表达式如下：

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

其中，x, y, z 是状态变量，t 是时间，而 σ, ρ, β 是控制参数。洛伦兹方程的三个具体方程描述了一个三维动力系统，通常用来模拟大气对流中的混沌行为。以下是这三个方程的具体含义：

- 第一个方程描述了 x 的变化率。在这里 ρ 是控制参数，它代表了对流层中的水平温度梯度和垂直温度梯度之比的敏感性。这一项表示流体中的扩散效应，即 x 的变化受到 y 和 x 本身之间的差异的影响。

- 第二个方程描述了 y 的变化率。在这里 ρ 是控制参数，它代表了流体对温度差异的响应。第一项 $x(\rho - z)$ 描述了温度差异对 y 的影响，而第二项 $-y$ 描述了 y 本身的耗散效应。

- 第三个方程描述了 z 的变化率。在这里 β 是控制参数，它与流体中的垂直温度梯度有关。第一项 xy 表示 x 和 y 之间的相互作用，而第二项 $-\beta z$ 描述了 z 本身的耗散效应。

Lorenz63 系统显示出在特定参数值下的混沌行为，这意味着系统对初始条件极其敏感，微小的变化可能导致系统轨迹的巨大差异。这使得洛伦兹方程在混沌理论和动力系统研究中具有重要意义。

2.1.2 龙格-库塔法基本概念

龙格-库塔法（Runge-Kutta method）是一种数值积分方法，用于求解常微分方程（ODEs）的初值问题。这个方法基于一系列的数学步骤，通过逐步逼近解的值，允许我们在离散的时间步长上计算 ODEs 的数值解。它在求解这对于一些无法通过解析方法得到解的方程非常有用，例如洛伦兹方程。

一般来说，最常见的 Runge-Kutta 方法是四阶（RK4）。这是因为它在数值稳定性、精度和计算效率之间取得了一种良好的平衡。RK4 相对于更低阶的 Runge-Kutta 方法，如 RK2，具有更高的精度，同时相对于更高阶的方法，如 RK6 或 RK8，它的计算开销更小。

令初值问题的表述如下：

$$\begin{cases} \frac{dy}{dt} = f(t, y) \\ y(t_0) = y_0 \end{cases} \tag{2}$$

则对于该问题的 RK4 由如下方程给出：

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 3k_3 + k_4) \tag{3}$$

其中

$$\begin{cases} k_1 = f(t_n, y_n) \\ k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 = f(t_n + h, y_n + hk_3) \end{cases} \tag{4}$$

2.2 一维浅水方程及有限体积法

2.2.1 一维浅水方程基本概念

浅水方程（shallow water equation, SWE）常用来描述一个水平广度远远大于深度的水体（例如湖泊，溪流，海岸）。浅水方程，顾名思义，它成立的一个重要假设就是水很“浅”，这个假设具体体现在它默认对于任意一个固定的位置，在不同深度的上的所有水都有着同样的运动速度。换句话说，水的流速在竖直方向上是不变的。水“浅”假设的另外一个简化效果就是我们只需要考虑水体在水平方向的移动，而忽略水在竖直方向的运动。一维浅水波的方程如下：

$$\begin{cases} \frac{\partial}{\partial t}h + \frac{\partial}{\partial x}hv = 0 \\ \frac{\partial}{\partial t}hv + \frac{\partial}{\partial x}(hv^2 + \frac{gh^2}{2}) = 0 \end{cases} \tag{5}$$

其中，$h(x,t)$ 为水在时间 t 位置 x 的高度；$v(x,t)$ 为在时间 t 位置 x 上水体的流速，默认右为正方向；g 为重力加速度；

一维浅水方程的积分形式如下：

$$\begin{cases} \frac{d}{dt}\int_{x_0}^{x_1} h dx = h(x_0,t)v(x_0,t) - h(x_1,t)v(x_1,t) = \int_{x_0}^{x_1} -\frac{\partial}{\partial x}(hv)dx \\ \frac{d}{dt}\int_{x_0}^{x_1} hv dx = h(x_0,t)v^2(x_0,t) + \frac{gh^2(x_0,t)}{2} - h(x_1,t)v^2(x_1,t) - \frac{gh^2(x_1,t)}{2} \end{cases} \tag{6}$$

第一个方程是质量守恒方程：$\int_{x_0}^{x_1} h dx$ 是水体在区间 $[x_0,x_1]$ 上的体积，则 $\int_{x_0}^{x_1} \rho h dx$ 为该区间上水体的质量。所以 $\frac{d}{dt}\int_{x_0}^{x_1} \rho h dx$ 表示该区域水体质量随着时间的变化，由质量守恒定律可知，显然等于 x_0 处水体质量的变化减去 x_1 处水体质量的变化，即 $\rho h(x_0,t)v(x_0,t) - \rho h(x_1,t)v(x_1,t)$。等式两边约去水体密度 ρ 即得第一个方程。

第二个方程是动量守恒方程：由于 $\int_{x_0}^{x_1} \rho h dx$ 是水体在区间 $[x_0,x_1]$ 上的质量，则 $\int_{x_0}^{x_1} \rho hv dx$ 是水体在区间 $[x_0,x_1]$ 上的动量，$\frac{d}{dt}\int_{x_0}^{x_1} \rho hv dx$ 表示区间 $[x_0,x_1]$ 上水体总动量随着时间的变化，这一变化有两个来源。

1. 流入流出区间的水本身带有的动量：用 x_0 处流入的动量减去 x_1 处流出的动量，即 $\rho h(x_0,t)v^2(x_0,t) - \rho h(x_1,t)v^2(x_1,t)$。

2. 边界处的水压对区间内水体施加的力；根据公式 $\frac{dI}{dt} = F$，在三维空间，我们将压强对面积进行积分便可得到压力，现在在二维空间，我们只需要将压强对于深度进行积分便可得到压力。物理学中的压强表达式为 $\rho g(h-y)$，则 x_0 处左边的水体对区间内水体的压力为：

$$\begin{aligned} \int_0^{h(x_0,t)} \rho g(h(x_0,t)-y)dy &= \int_0^{h(x_0,t)} \rho g h(x_0,t)dy - \int_0^{h(x_0,t)} \rho g y dy \\ &= \rho g h^2(x_0,t) - \frac{\rho g h^2(x_0,t)}{2} \\ &= \frac{\rho g h^2(x_0,t)}{2} \end{aligned} \tag{7}$$

同理，x_1 处右边的水体对区间内水体的压力为 $\frac{\rho g h^2(x_1,t)}{2}$，所以外部压力导致的动量变化为：$\frac{\rho g h^2(x_0,t)}{2} - \frac{\rho g h^2(x_1,t)}{2}$。

所以，

$$\frac{d}{dt}\int_{x_0}^{x_1} \rho hv dx = \rho h(x_0,t)v^2(x_0,t) + \frac{\rho g h^2(x_0,t)}{2} - \rho h(x_1,t)v^2(x_1,t) - \frac{\rho g h^2(x_1,t)}{2} \tag{8}$$

等式两边同时约去水体密度 ρ，即得第二个积分方程。

2.2.2 有限体积法解一维浅水方程

有限体积法（finite volume method，FVM）是计算流体力学中常用的一种数值算法，有限体积法基于的是积分形式的守恒方程而不是微分方程，该积分形式的守恒方程描述的是计算网格定义的每个控制体。有限体积法着重从物理观点来构造离散方程，每一个离散方程都是有限大小体积上某种物理量守恒的表示式，推导过程物理概念清晰，离散方程系数具有一定的物理意义，并可保证离散方程具有守恒特性。

使用有限体积法求解一维浅水方程的步骤如下：

1. 将求解区间划分为有限多个子区间 $[x_0, x_1], [x_1, x_2], [x_2, x_3], \cdots, [x_{n-1}, x_n]$，这里的每一个区间被称为控制体积（control volume）。每一个区间的长度不一定要相等，但是在本文，为了方便起见，我们令每一个控制体积都有相同的长度 Δx。

2. 对于每一个小区间 $[x_i, x_{i+1}]$，计算守恒量。对于一维浅水方程，需要存储质量的平均值和动量的平均值：

$$\begin{cases} H_i = \frac{1}{\Delta x} \int_{x_i}^{x_{i+1}} h dx \\ M_i = \frac{1}{\Delta x} \int_{x_i}^{x_{i+1}} hv dx \end{cases} \tag{9}$$

3. 使用 H_i, M_i 近似 $h(x,t)$ 和 $v(x,t)$

   - 对于任意的 $x \in (x_i, x_{i+1})$，将 $h(x,t)$ 和 $v(x,t)$ 近似为：

$$\begin{cases} h(x,t) \approx H_i(t) \\ h(x,t)v(x,t) \approx M_i(t) \\ v(x,t) \approx \frac{M_i(t)}{H_i(t)} \end{cases} \tag{10}$$

   - 在邻接点 $\{x_i\}_1^{n-1}$ 上，我们可以通过对边界点左右两个区间的守恒量取平均值，对于 FVM 以及其他同类数值解函数存在不连续性的数值方法，这样给定两个相邻的控制体积上的函数解，求两个控制体积交界处的函数值的策略被称为数值通量（numerical flux）。这样简单取两个值的平均的通量被称为中央通量（central flux），不过它只是众多通量选择中的一种。在方程比较简单并且解函数比较平滑的时候，中央通量还是比较可靠的。但是对于 SWE，中央通量是数值不稳定的（numerically unstable）。为了解决这一问题，这里我们采用 Lax–Friedrichs 通量。Lax-Friedrich 通量很容易计算：分别取左边和右边两区间上的值计算通量，取二值的平均，再加上一修正参数，即系统中的波速乘以该守恒量在两个区间上的差再除以二。这一修正参数被称为数值消散（numerical dissipation）。

$$h(x_i,t)v(x_i,t) \approx \frac{M_{i-1} + M_i}{2} + c \cdot \frac{H_{i-1} - H_i}{2} \tag{11}$$

     其中，c 为波速，可以通过计算通量向量对守恒向量的雅可比矩阵的特征值得到。在一些复杂的问题中，雅可比矩阵的计算可能较为繁琐，或者数值解法的收敛性可能受到影响。这时，工程师或研究人员可能会通过"trial and error" 方法来选择一个合适的常数，使得模拟或计算的结果符合实际情况。具体来说，在计算波速时，可能会通过设定不同的波速值，运行模拟或数值计算，然后观察模拟结果，选择使结果与实际情况相符的波速。这样的过程可能需要多次迭代，即反复尝试不同的值，直到找到一个令人满意的解。

   - 对于边界点 x_0 和 x_n，当我们使用反射墙边界条件时，我们可以人为地设置两个假想体积（ghost cell）。比如，我们可以认为在区间 $[x_0, x_1]$ 的左边还有一个区间：$[x_{-1}, x_0]$。并且 $H_{-1} = H_0, M_{-1} = -M_0$。也就是说：我们认为在 $[x_{-1}, x_0]$ 上的水和 $[x_0, x_1]$ 上的水有相同的高度但是相反的运动方向。形象一点来说的话，那就是我们可以认为 x_0 上有一面镜子，只不过这个"镜子"反射的不是光，而是动量。区间 $[x_{n-1}, x_n]$ 也采用同样的策略去处理。

4. 将上述结果带入浅水方程的积分形式 6进行替换：

$$\begin{cases} \frac{d}{dt}H_i\Delta x = h(x_i,t)v(x_i,t) - h(x_{i+1},t)v(x_{i+1},t) \\ \frac{d}{dt}M_i\Delta x = h(x_i,t)v^2(x_i,t) - h(x_{i+1},t)v^2(x_{i+1},t) + \frac{gh^2(x_i,t)}{2} - \frac{gh^2(x_{i+1},t)}{2} \end{cases} \tag{12}$$

其中

$$\begin{cases} h(x_i,t)v(x_i,t) = \frac{M_{i-1}+M_i}{2} + c\cdot\frac{H_{i-1}-H_i}{2} \\ h(x_i,t)v^2(x_i,t) + \frac{gh^2(x_i,t)}{2} = \frac{\frac{M_{i-1}^2}{H_{i-1}}+\frac{M_i^2}{H_i}}{2} + \frac{gH_{i-1}^2+gH_i^2}{4} + c\cdot\frac{M_{i-1}-M_i}{2} \end{cases} \tag{13}$$

其中 c 为波速，$H_{-1} = H_0, H_{n+1} = H_n, M_{-1} = -M_0, M_{n+1} = -M_n$，所以由 FVM 得到的最终格式为：

$$\begin{cases} \frac{d}{dt}H_i(t) = \frac{1}{2\Delta x}[M_{i-1} - M_{i+1} + c\cdot(H_{i-1} - 2H_i + H_{i+1})] \\ \frac{d}{dt}M_i(t) = \frac{1}{2\Delta x}[(\frac{M_{i-1}^2}{H_{i-1}} - \frac{M_{i+1}^2}{H_{i+1}}) + \frac{g}{2}(H_{i-1}^2 - H_{i+1}^2) + c(M_{i-1} - 2M_i + M_{i+1})] \end{cases} \tag{14}$$

5. 使用四阶龙格库塔法（RK4）进行数值求解；

   由 FVM 得到的格式 14对应的是一个非线性方程组，因为 $i \in \{1,2,3,\cdots,n\}$，所以方程的个数为 $2n$。由于四阶龙格－库塔法（RK4）计算简单，精度较高，所以采用该方法求解。

2.3 RFM 解偏微分方程

随机特征方法（random feature metod, RFM）是 [1] 提出的一种用于解偏微分方程组的方法，该方法是传统算法和基于机器学习的算法之间的自然桥梁，它吸取了二者的优点，将微分方程求解问题转换为线性最小二乘问题。RFM 基于以下几个思想的组合：

1. 使用随机特征函数表示近似解；
2. 用配点法处理偏微分方程的约束；
3. 用罚函数处理边界条件，这使得我们可以在相同的基础上处理边界条件和物理方程的约束；
4. 多尺度表示；
5. 损失函数中各项权重的重新缩放，以平衡各项对总损失的影响；

考虑如下静态边值问题：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Omega \\ \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}) & \boldsymbol{x} \in \partial\Omega \end{cases} \tag{15}$$

其中，f 和 g 是已知函数，$\boldsymbol{x} = (x_1, x_2, \cdots, x_{d_x})^T \in \Omega \subset \mathbb{R}^{d_x}$，$\partial\Omega$ 是 Ω 的边界，记 $d_x \in \mathbb{N}^+$ 为 $\boldsymbol{x}$ 的维度，$d_u \in \mathbb{N}^+$ 为输出的维度，$\mathcal{L}$ 是线性微分算子，$\mathcal{B}$ 是边界算子。

2.3.1 随机特征函数

随机特征函数（random feature function, RFF）就是特征向量随机生成的函数。参考神经网络的随机特征模型，选取随机特征函数作为基函数，构造一个内部参数固定的三层神经网络。如图，输入层和隐藏层之间的权重参数和偏置参数固定，隐藏层和输出层之间的权重参数待训练。

对于机器学习方法来说，特征向量就是网络的权重和偏置，特征函数的随机生成就是网络权重和偏置的随机初始化步骤。因此从机器学习的框架下看，RFM 就是利用 M 个定义在 Ω 上的网络基函数 $\{\phi_m\}$ 的线性组合来表示数值解：

$$u_M(\boldsymbol{x}) = \sum_{m=1}^{M} u_m \phi_m(\boldsymbol{x}) \tag{16}$$

$$\phi_m(\boldsymbol{x}) = \sigma(\boldsymbol{k}_m \cdot \boldsymbol{x} + b_m) \tag{17}$$

其中 $\boldsymbol{k}_m, b_m$ 就是随机生成后固定的内层参数，而 σ 是非线性激活函数（一般取双曲正切函数 $tanh$ 即可），为方程求解提供非线性的部分；u_m 是外层待训练的参数。

此时，对于 $\forall \boldsymbol{x} \in \Omega$，则有：

$$\begin{aligned}
\mathcal{L}u(\boldsymbol{x}) &\approx \mathcal{L}u_M(\boldsymbol{x}) \\
&= \mathcal{L}\sum_{m=1}^{M} u_m \phi_m(\boldsymbol{x}) \\
&= \sum_{m=1}^{M} u_m \mathcal{L}\phi_m(\boldsymbol{x}) \\
&= \begin{bmatrix} \mathcal{L}\phi_1(\boldsymbol{x}) & \mathcal{L}\phi_2(\boldsymbol{x}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix}
\end{aligned}$$

所以，我们在 Ω 上取一组配点 $\boldsymbol{x}_P^{(1)}, \boldsymbol{x}_P^{(2)}, \cdots, \boldsymbol{x}_P^{(N_P)}$，则有

$$\begin{bmatrix} \mathcal{L}\phi_1(\boldsymbol{x}_P^{(1)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(1)}) \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(2)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(N_P)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} \approx \begin{bmatrix} f(\boldsymbol{x}_P^{(1)}) \\ f(\boldsymbol{x}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \tag{18}$$

同理，我们在 $\partial\Omega$ 上取一组配点 $\boldsymbol{x}_B^{(1)}, \boldsymbol{x}_B^{(2)}, \cdots, \boldsymbol{x}_B^{(N_B)}$，则有

$$\begin{bmatrix} \phi_1(\boldsymbol{x}_B^{(1)}) & \phi_2(\boldsymbol{x}_B^{(1)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(1)}) \\ \phi_1(\boldsymbol{x}_B^{(2)}) & \phi_2(\boldsymbol{x}_B^{(2)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_B^{(N_B)}) & \phi_2(\boldsymbol{x}_B^{(N_B)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} \approx \begin{bmatrix} g(\boldsymbol{x}_B^{(1)}) \\ g(\boldsymbol{x}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \tag{19}$$

记:

$$P = \begin{bmatrix} \mathcal{L}\phi_1(\boldsymbol{x}_P^{(1)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(1)}) \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(2)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(N_P)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix}$$

$$B = \begin{bmatrix} \phi_1(\boldsymbol{x}_B^{(1)}) & \phi_2(\boldsymbol{x}_B^{(1)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(1)}) \\ \phi_1(\boldsymbol{x}_B^{(2)}) & \phi_2(\boldsymbol{x}_B^{(2)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_B^{(N_B)}) & \phi_2(\boldsymbol{x}_B^{(N_B)}) & \cdots & \phi_M(\boldsymbol{x}^{(N_B)}) \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} \quad , f = \begin{bmatrix} f(\boldsymbol{x}_P^{(1)}) \\ f(\boldsymbol{x}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{x}_P^{(N_P)}) \\ g(\boldsymbol{x}_B^{(1)}) \\ g(\boldsymbol{x}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix}$$

将方程 18和 19 整合成一个线性方程组:

$$Au = f \quad , A = \begin{bmatrix} P \\ B \end{bmatrix} \tag{20}$$

这样一来就把偏微分方程的求解问题转换成了形如 $Au = f$ 的线性方程组求解问题，其中 u 是待求解的参数。这种问题可以采用线性最小二乘方法，当前比较流行的科学计算库都有相应的 API 可以直接使用，如 Numpy、Scipy 等。

2.3.2 局部随机特征模型和单位分解

上述随机特征函数是全局定义的，一般而言是无法较好地拟合偏微分方程的，因为微分方程的解常常有小尺度的局部变化。这一点从神经网络的角度去理解也是很显然的，用一个三层的神经网络去拟合一个较为复杂的偏微分方程，并且还是在内层参数固定，可训练参数只有一层的情况下，几乎是无法较好地拟合的。为了解决这一问题，可以将定义域划分为多个局部子区域，让每个局部子区域的范围足够小，并在每个局部子区域构造随机特征函数，使得每个局部随机特征函数能够很好地拟合对应范围内的偏微分方程。最后再用单位分解（partition of unity, PoU）技术将它们组合。

具体来说，分为以下几个步骤:

1. 将定义域 Ω 划分为多个子区域 $\{\Omega_n\}_{n=1}^{M_p}$，在没有先验知识的情况下，均匀划分即可。

2. 计算所有子区域的的中心 $\{\hat{\boldsymbol{x}}_n\}_{n=1}^{M_p}$ 和区域半径 $\{\boldsymbol{r}_n\}_{n=1}^{M_p}$。

$$\begin{aligned}\hat{\boldsymbol{x}}_n &= (\hat{x}_{n1}, \hat{x}_{n2}, \cdots, \hat{x}_{nd_x})^T \\ &= (\frac{x_{n1\_max} + x_{n1\_min}}{2}, \frac{x_{n2\_max} + x_{n2\_min}}{2}, \cdots, \frac{x_{nd_x\_max} + x_{nd_x\_min}}{2})^T\end{aligned} \tag{21}$$

$$\begin{aligned}\boldsymbol{r}_n &= (r_{n1}, r_{n2}, \cdots, r_{nd_x})^T \\ &= (\frac{x_{n1\_max} - x_{n1\_min}}{2}, \frac{x_{n2\_max} - x_{n2\_min}}{2}, \cdots, \frac{x_{nd_x\_max} - x_{nd_x\_min}}{2})^T\end{aligned} \tag{22}$$

其中，x_{nj_max} 和 x_{nj_min} 是子区域 Ω_n 在第 j 维的区间上限和区间下限。

3. 在每个 Ω_n 上构造线性变换

$$l_n(\boldsymbol{x}) = \frac{1}{\boldsymbol{r}_n}(\boldsymbol{x} - \hat{\boldsymbol{x}}_n), \quad n = 1, 2, \cdots, M_p, \quad \boldsymbol{x} \in \Omega \tag{23}$$

这个线性变换类似于神经网络的输入标准化，将 $[\hat{x}_{n1} - r_{n1}, \hat{x}_{n1} + r_{n1}] \times [\hat{x}_{n2} - r_{n2}, \hat{x}_{n2} + r_{n2}] \times \cdots \times [\hat{x}_{nd_x} - r_{nd_x}, \hat{x}_{nd_x} + r_{nd_x}]$ 的小局部映射到统一的区间 $[-1, 1]^{d_x}$，以便实现局部特征的拟合。

4. 对每个 Ω_n，通过一个两层的神经网络构造 $J_n \in \mathbb{N}^+$ 个随机特征函数：

$$\phi_{nj}(\boldsymbol{x}) = \sigma(\boldsymbol{k}_{nj}\boldsymbol{x} + b_{nj}), \quad j = 1, \cdots, J_n. \tag{24}$$

其中，非线性激活函数 σ 一般取为双曲正切函数 $tanh$ 或三角函数 sin, cos；$\boldsymbol{k}_{nj}$ 和 b_{nj} 独立同分布，都是均匀采样：$\boldsymbol{k}_{nj} \sim \mathbb{U}([-R_{nj}, R_{nj}]^{d_x})$ 和 $b_{nj} \sim \mathbb{U}([-R_{nj}, R_{nj}])$，一旦随机初始化完成，则固定不变。

5. 以每个 Ω_n 为支集，构造一个单位分解函数 ψ_n，即 $supp(\psi_n) = \Omega_n$。单位分解函数一般有两种取法，以下以 $d_x = 1$，即一维输入的情形举例，函数图像见图：

$$\psi_n^a(x) = \mathbb{I}_{[-1,1]}(x) \tag{25a}$$

$$\psi_n^b(x) = \begin{cases} \frac{1+\sin(2\pi x)}{2} & -\frac{5}{4} \le x < -\frac{3}{4}, \\ 1 & -\frac{3}{4} \le x < \frac{3}{4}, \\ \frac{1-\sin(2\pi x)}{2} & \frac{3}{4} \le x < \frac{5}{4}, \\ 0 & otherwise. \end{cases} \tag{25b}$$

对于 $d_x \ge 2$，即多维输入的情形。先对每个维度构造一维的单位分解函数，然后对这些一维单位分解函数进行张量积运算：$\psi_n(\boldsymbol{x}) = \prod_{i=1}^{d_x} \psi_n(x_i)$。

则最终数值解就是将局部随机特征函数通过单位分解函数组合起来：

$$u_M(\boldsymbol{x}) = \sum_{n=1}^{M_p} \psi_n(l_n(\boldsymbol{x})) \sum_{j=1}^{J_n} u_{nj}\phi_{nj}(l_n(\boldsymbol{x})) \tag{26}$$

需要注意的是，在单位分解函数的连续性不满足方程解所需的连续性时，需要额外加入一组连续性条件。例如：在使用 ψ^a 求解二阶方程时，需要在不同单位分解计算区域的接口处加入零阶和一阶连续性条件。而 ψ^b 是连续可微的，不需要添加额外条件。两种单位分解函数并无优劣之分，后文如无特殊说明，皆采用 ψ^b 作为默认的单位分解函数。

仍以方程 15为例，对其使用局部随机特征模型和单位分解技术进行离散化：此时，对于 $\forall \boldsymbol{x} \in \Omega$，则有：

$$
\begin{aligned}
\mathcal{L}u(\boldsymbol{x}) &\approx \mathcal{L}u_M(\boldsymbol{x}) \\
&= \mathcal{L}\sum_{n=1}^{M_p}\psi_n(l_n(\boldsymbol{x}))\sum_{j=1}^{J_n}u_{nj}\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \mathcal{L}\sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\mathcal{L}\psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\mathcal{L}\Phi_{nj}(\boldsymbol{x}) \quad \left[\Phi_{nj}(\boldsymbol{x}) \triangleq \psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x}))\right] \\
&= \begin{bmatrix}\mathcal{L}\Phi_{11}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x})\end{bmatrix}
\begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix}
\end{aligned} \tag{27}
$$

所以，我们在 Ω 上取一组配点 $\boldsymbol{x}_P^{(1)}, \boldsymbol{x}_P^{(2)}, \cdots, \boldsymbol{x}_P^{(N_P)}$，则有

$$
\begin{bmatrix}
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(1)}) \\
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(2)}) \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(N_P)})
\end{bmatrix}
\begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix}
\approx
\begin{bmatrix} f(\boldsymbol{x}_P^{(1)}) \\ f(\boldsymbol{x}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \tag{28}
$$

同理，我们在 $\partial\Omega$ 上取一组配点 $\boldsymbol{x}_B^{(1)}, \boldsymbol{x}_B^{(2)}, \cdots, \boldsymbol{x}_B^{(N_B)}$，则有

$$
\begin{bmatrix}
\Phi_{11}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(1)}) \\
\Phi_{11}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(2)}) \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\Phi_{11}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(N_B)})
\end{bmatrix}
\begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix}
\approx
\begin{bmatrix} g(\boldsymbol{x}_B^{(1)}) \\ g(\boldsymbol{x}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \tag{29}
$$

记:

$$P=\begin{bmatrix}\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(1)})\\ \mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(2)})\\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ \mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(N_P)})\end{bmatrix}$$

$$B=\begin{bmatrix}\Phi_{11}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(1)})\\ \Phi_{11}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(2)})\\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ \Phi_{11}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(N_B)})\end{bmatrix}$$

$$u=\begin{bmatrix}u_{11}\\ \vdots\\ u_{1J_1}\\ \vdots\\ u_{M_p1}\\ \vdots\\ u_{M_pJ_{M_p}}\end{bmatrix},f=\begin{bmatrix}f(\boldsymbol{x}_P^{(1)})\\ f(\boldsymbol{x}_P^{(2)})\\ \vdots\\ f(\boldsymbol{x}_P^{(N_P)})\\ g(\boldsymbol{x}_B^{(1)})\\ g(\boldsymbol{x}_B^{(2)})\\ \vdots\\ g(\boldsymbol{x}_B^{(N_B)})\end{bmatrix}$$

则偏微分方程的求解问题再次转换为线性方程组求解问题，自由度为 $M=\sum_{n=1}^{M_p}J_n$:

$$Au=f\quad,A=\begin{bmatrix}P\\ B\end{bmatrix}\tag{30}$$

2.3.3 损失函数及罚参数自动缩放

在 RFM 中，损失函数同样在配点上进行计算，和 PINN 的损失函数构造完全一致，使用强形式在配点上构造损失函数

$$Loss=\sum_{\boldsymbol{x}_i\in\Omega}\lambda_i\|\mathcal{L}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{x}_i)-\boldsymbol{f}(\boldsymbol{x}_i)\|_2^2+\sum_{\boldsymbol{x}_j\in\partial\Omega}\lambda_j\|\mathcal{B}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{x}_j)-\boldsymbol{g}(\boldsymbol{x}_j)\|_2^2.\tag{31}$$

公式中，罚参数 λ_i 的确定是关键，往往决定着模型的收敛速度和收敛效果。为了平衡 PDE 损失项和边界条件损失项，一种简单而高效的策略就是通过缩放罚参数使得损失函数中每一项具有相同的数量级。

在传统的 PINN 的训练过程中，罚参数的调整较为困难，往往依赖经验。但 RFM 的情况则完全不同，虽然损失函数形式上看完全类似，但 RFM 与 PINN 构造的优化问题则完全不同，RFM 求解的是内层参数固定、仅优化最外层线性参数的线性优化问题。从经典数值算法的角度来看，RFM 构造的是线性最小二乘问题，该问题可以表为线性系统 $Au=f$ 的形式。

对于方程 $Au = f$，最小二乘解的目的就是求解向量 u 使得误差 $e = ||Au - f||^2$ 达到最小，要使每一项损失 $e_i = ||A_i u - f_i||^2$ 具有相同的数量级，只需直接基于矩阵 A 的信息，对矩阵的每一行元素 A_i 缩放到同一个数量级。

$$\begin{cases} \lambda_i = \frac{c_i}{max(A_i, -A_i)} \\ A_i = \lambda_i \times A_i \\ f_i = \lambda_i \times f_i \end{cases} \tag{32}$$

其中，$max(A_i, -A_i)$ 表示矩阵第 i 行的最大值，c_i 表示要缩放到的数量级，在没有先验知识的情况下设置为统一常数即可，如 100。

2.3.4 时空随机特征方法

考虑如下时空偏微分方程：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x}, t) = f(\boldsymbol{x}, t) \quad \boldsymbol{x}, t \in \Omega \times [0, T] \\ \mathcal{B}u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t) \quad \boldsymbol{x}, t \in \partial\Omega \times [0, T] \\ \mathcal{I}u(\boldsymbol{x}, t) = h(\boldsymbol{x}) \quad \boldsymbol{x}, t \in \Omega \end{cases} \tag{33}$$

其中，f，g 和 h 是已知函数，$\boldsymbol{x} = (x_1, x_2, \cdots, x_{d_x})^T \in \Omega \subset \mathbb{R}^{d_x}$，$t \in [0, T]$，$\partial\Omega$ 是 Ω 的边界，记 $d_x \in \mathbb{N}^+$ 为 $\boldsymbol{x}$ 的维度，$d_u \in \mathbb{N}^+$ 为输出的维度，$\mathcal{L}$ 是线性微分算子，$\mathcal{B}$ 是边界算子，$\mathcal{I}$ 是初值算子。

时空随机特征方法（Space-Time Random Feature Method, ST-RFM）是一种用于求解时间相关的偏微分方程的方法，其背后思想类似于静态方程的 RFM 方法，下面我们简要介绍算法逻辑：

1. 将空间域 Ω 划分为 N_x 个子域 $\{\Omega_n\}_{n=1}^{N_x}$，每个子域 Ω_n 的中心点记为 $\hat{\boldsymbol{x}}_n$。

2. 将时间域 $[0, T]$ 划分为 N_t 个子域，即 $[0, T] = [t_0, t_1) \cup [t_1, t_2) \cup \cdots \cup [t_{N_t - 1}, t_{N_t}]$，每个子域的中心点 $\hat{t}_n = \frac{t_{n-1} + t_n}{2}$。

3. 构造时空单位分解函数，通过对空间单位分解函数和时间单位分解函数进行乘积运算：

$$\psi_{n_x, n_t}(\boldsymbol{x}, t) = \psi_{n_x}(l_{n_x}(\boldsymbol{x})) \cdot \psi_{n_t}(l_{n_t}(t)) \tag{34}$$

   其中，n_x 和 n_t 分别是空间子域和时间子域的索引。$l_{n_x}(\boldsymbol{x})$ 和 $l_{n_t}(t)$ 是线性变换，其作用是分别把 $\boldsymbol{x} \in \Omega_{n_x}$ 映射到 $[-1, 1]^{d_x}$，把 $t \in [t_{n_t - 1}, t_{n_t}]$ 映射到 $[-1, 1]$。

4. 将空间随机特征函数推广到时空随机特征函数，这里有两种思路：

   时空拼接 (concatenation of spatial and temporal, STC)，这种思路是空间随机特征函数的自然推广，相当于把空间变量 $\boldsymbol{x}$ 和时间变量 t 拼接到一起，成为一个 $d_x + 1$ 维的输入变量。

$$\phi_{n_x, n_t, j}(\boldsymbol{x}, t) = \sigma(\boldsymbol{k}_{n_x, n_t, j} l_{n_x, n_t}(\boldsymbol{x}, t) + b_{n_x, n_t, j}) \tag{35}$$

   其中，$\boldsymbol{k}_{n_x, n_t, j}$ 是时空输入的权重矩阵，$b_{n_x, n_t, j}$ 是偏置，$l_{n_x, n_t}(\boldsymbol{x}, t)$ 是线性变换，其作用是把 $(\boldsymbol{x}, t) \in \Omega_{n_x} \times [t_{n_t - 1}, t_{n_t}]$ 映射到 $[-1, 1]^{d_x + 1}$。

变量分离 (separation of variables, SoV)，为了模拟偏微分方程数值解法的一类经典方法：变量分离技术，把空间输入和时间输入对应到不同的随机特征函数，然后对结果做乘积：

$$\phi_{n_x,n_t,j}(\boldsymbol{x},t)=\sigma(\boldsymbol{k}^x_{n_x,n_t,j}l_{n_x}(\boldsymbol{x})+b^x_{n_x,n_t,j})\cdot\sigma(k^t_{n_x,n_t,j}l_{n_t}(t)+b^t_{n_x,n_t,j}) \tag{36}$$

以上两种随机特征函数的结构如图，这两种构造策略均被证明能够有效地收敛到真解，并且在理论上和数值算例上并无谁优谁劣，后文如无特殊说明，皆采用 STC 作为默认策略。

5. 将局部随机特征函数通过单位分解函数组合起来，得到最终的数值解：

$$u_M(\boldsymbol{x},t)=\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\psi_{n_x,n_t}(\boldsymbol{x},t)\sum_{j=1}^{J_n}u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \tag{37}$$

ST-RFM 算法的其他部分诸如矩阵的构造、损失函数的罚参数缩放等，都与 RFM 高度相似，这里不做过多赘述。

2.4 PINN 解偏微分方程

2.4.1 通用逼近定理

通用逼近定理（Universal Approximation Theorem）有多个版本，最早的版本可以追溯到 1989 年，由 George Cybenko 在他的论文 [6] 中提出。

定理 1. *(Universal Approximation Theorem).* 对于任意连续函数 $f:\mathbb{R}^n\to\mathbb{R}^m$ 和任意小的正数 $\epsilon>0$，存在一个具有单个隐层的前馈神经网络（*MLP*），该网络使用某种特定的非线性激活函数（如 *sigmoidal* 函数），使得该网络能够以任意精度逼近 f。更具体地说，对于给定的输入 x，存在一个神经网络输出 $NN(x)$，满足 $|f(x)-NN(x)|<\epsilon$。

随后的研究产生了许多关于通用逼近定理的扩展，其中一些关注的方面包括神经网络的深度、宽度、激活函数的选择等。一些扩展表达了更一般化的定理，包括多层神经网络、任意宽度的隐藏层等情况。这些扩展使得我们能够更深入地理解神经网络的表示能力。

总体而言，通用逼近定理提供了理论支持，证明了神经网络在理论上具备以任意精度逼近连续函数的能力。然而，实际应用中，选择适当的网络结构、激活函数以及合理的训练方法等仍然是至关重要的。

2.4.2 物理信息神经网络 PINN

文献 [9, 11] 介绍了物理信息神经网络（Physics-Informed Neural Networks，PINN），文献 [9] 介绍了一种 PINN 的求解框架。PINN 是一种结合神经网络和物理方程的方法，用于求解偏微分方程（PDE）或其他物理问题。PINN 的目标是通过神经网络学习系统的非线性行为，同时利用已知的物理方程来指导神经网络的训练，从而提高模型的准确性。

以下是 PINN 的基本思想和特点：

- 神经网络模型：PINN 使用神经网络来建模未知的物理量，该神经网络通常称为“解网络”或“主网络”。这个网络可以是深度前馈神经网络（Feedforward Neural Network）或者是循环神经网络（Recurrent Neural Network），具体结构取决于问题的性质。
- 物理方程约束：PINN 的关键创新在于将已知的物理方程嵌入到神经网络中。这通常通过在损失函数中引入物理方程项来实现。这样，模型在学习数据的同时，也被强制满足物理方程，从而提高模型的物理可解性。
- 初边值条件：与传统的数值方法不同，PINN 通常不需要显式地给定初边值条件。初边值条件被隐式地包含在训练数据中，模型通过学习过程中的损失函数来自动适应这些条件。
- 不需要大量数据：由于使用物理方程作为先验知识，PINN 在处理小样本问题时表现良好，这使得它适用于许多实际场景，尤其是在实验数据有限的情况下。
- 适用范围：PINN 广泛应用于求解偏微分方程、反问题、流体动力学、结构力学等领域。由于其对物理方程的利用，PINN 在少量数据或者数据噪声较大的情况下表现良好。

总体而言，物理信息神经网络是一种将深度学习与已知物理方程相结合的方法，旨在提高模型的解释性和可靠性，特别适用于需要从有限数据中学习的物理问题。

2.5 深度算子网络解参数化的偏微分方程

由 2.4节，我们知道对于任意一个给定的偏微分方程，可以使用一个神经网络去逼近。这当然是一件很棒的事情，但是如果该偏微分方程的某些参数（如初值函数、边值函数、源项等）发生变化，那么我们将不得不重新训练神经网络，重新拟合满足新的条件参数的偏微分方程，而神经网络的训练是一件比较耗时的事情。

PINN 的作用相当于一个函数，显然不适合取拟合参数化的偏微分方程，我们需要的是一个算子网络，能够将偏微分方程的参数作为网络输入，输出该参数对应的真解。

2.5.1 算子通用逼近定理

文献 [3, 4, 5, 10, 12] 陈述了这样一个事实，对于任意一个非线性的连续泛函或者算子，存在一个具有单个隐层的神经网络去逼近它。文献 [5] 介绍了算子通用逼近定理。

定理 2. *(Universal Approximation Theorem for Operator).* 假设 σ 是连续非多项式函数，X 是巴拿赫空间，$K_1 \subset X, K_2 \subset \mathbb{R}^d$ 分别为 X 和 $\mathbb{R}^d$ 上的紧集，V 是 $C(K_1)$ 上的紧集，G 是将 V 映射到 $C(K_2)$ 的非线性连续算子。则对于 $\forall \epsilon > 0$，存在正整数 n, p, m 以及常数 $c_i^k, W_{bij}^k, b_{bij}^k, b_{tk} \in \mathbb{R}, W_{tk} \in \mathbb{R}^d, x_j \in K_1, i = 1, 2, \cdots, n, k = 1, 2, \cdots, p, j = 1, 2, \cdots, m$，使得对于 $\forall u \in V, y \in K_2$，成立：

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} W_{bij}^k u(x_j) + b_{bi}^k \right)}_{Branch} \cdot \underbrace{\sigma(W_{tk} \cdot y + b_{tk})}_{Trunk} \right| < \epsilon \tag{38}$$

算子通用逼近定理提供了理论支持，证明了神经网络在理论上具备以任意精度逼近连续泛函和算子的能力。实际应用中，选择适当的网络结构、激活函数以及合理的训练方法等仍然是至关重要的。

2.5.2 深度算子网络

深度算子网络（Deep Operator Network, DeepONet）是基于算子通用逼近定理衍生出来的一种神经网络架构，旨在为求解非线性的连续泛函或者算子提供更通用的方案。

我们知道深度神经网络的一般形式为：

$$NN(X) = W_n\sigma_{n-1}(W_{n-1}\sigma_{n-2}(...(W_2\sigma_1(W_1X + b_1) + b_2) + ..) + b_{n-1}) + b_n \tag{39}$$

再观察定理 2中的公式可以发现

$$\sum_{k=1}^{p}\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} W_{bij}^k u(x_j) + b_{bi}^k\right) \cdot \sigma(W_{tk} \cdot y + b_{tk}) \tag{40}$$

这部分和神经网络的表达式很类似，但有一些区别。我们需要分开来看：

- $\sigma(W_{tk} \cdot y + b_{tk})$

  这是一个两层神经网络，输入 y 为偏微分方程的配点，输出一个 p 维向量，我们称这个网络为主干网络。

- $\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} W_{bij}^k u(x_j) + b_{bi}^k\right)$

  这是一个三层神经网络，网络的输入是偏微分的参数 $u(x)$，网络的输出是一个一维标量。因为 $u(x)$ 是一个函数，所以要取其离散格式 $u(x_j), j = 1, 2, \cdots, m$ 作为输入。我们称这个网络为分支网络。

所以，算子网络是由 p 个分支网络和 1 个主干网络构成，最后通过一个内积操作将二者的输出结果融合，即把所有分支网络的输出合并为一个 p 维向量，与主干网络的输出向量做内积。图 1 展示了算子网络的架构。

观察算子网络的分支网络部分，我们可以发现它是由多个独立的分支网络堆叠而成，每个分支网络的输入是一样的，输出的维度也都是一维的，所以每个分支网络都是定义在相同定义域上的多元函数。把它们向量化就得到向量值函数，如此一来，自然可以用单独的一个深度神经网络来拟合。图 2 展示了非堆叠式的算子网络架构。

2.5.3 物理信息算子网络

文献 [8] 介绍了算子网络解参数化方程的方法论，并以源项和外力为参数展示了数值算例。偏微分方程的参数不仅有源项，还包括初值函数、边值函数等一切可以影响方程解的元素，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：

$$\mathcal{N}(u, s) = 0 \tag{41}$$

其中 $\mathcal{N}$ 表示非线性微分算子。

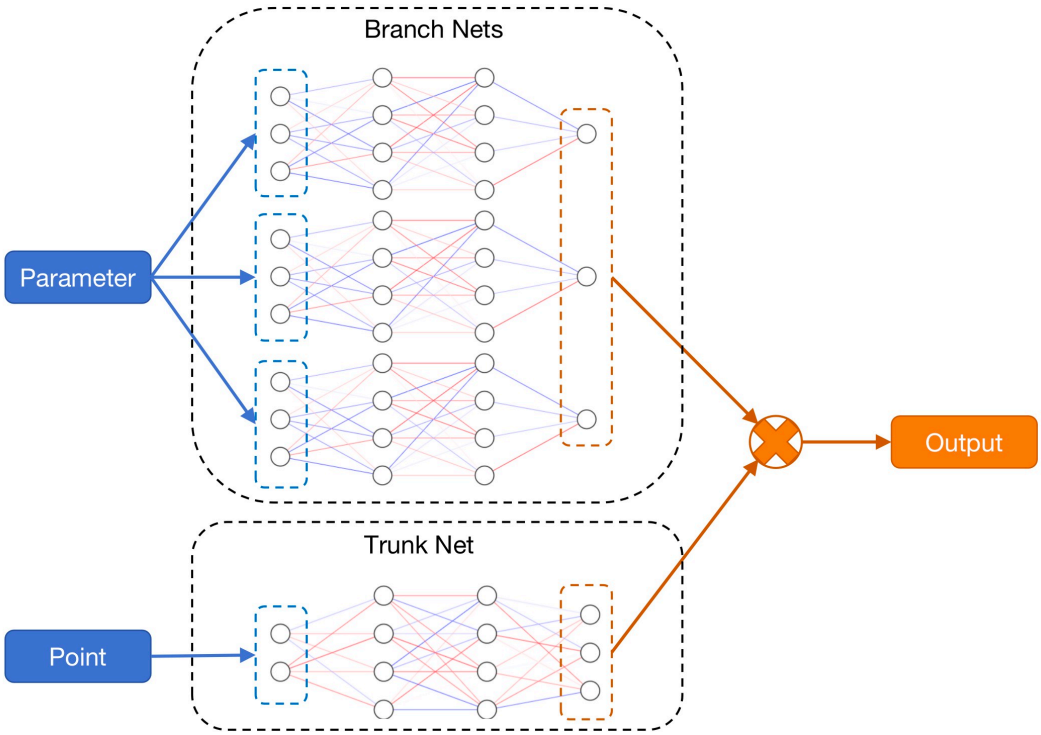


图 1: 堆叠式算子网络架构

此时，我们可以把 PDE 的一般解表示为算子 G：

$$G(u) = s \tag{42}$$

此时，u，s 均为函数，并且满足：

$$G(u)(y) = s(y) \in \mathbb{R}, \qquad \forall y \in \Omega \times [0, T] \tag{43}$$

其中，Ω 为偏微分方程的空间域，$[0, T]$ 为时间域。

结合深度算子网络和物理信息神经网络的思想，我们可以使用一个算子网络表示参数化的偏微分方程，并在损失函数中加入关于物理方程的项，得到物理信息算子网络（Physics-informed DeepONet）。分支网络的输入为偏微分方程的参数 $u(x)$，主干网络的输入为偏微分方程定义域上的配点。通过网络训练，得到网络解算子：

$$G_\theta(u)(y) \approx G(u)(y) \tag{44}$$

其中 θ 是网络的参数，包括权重（weight）和偏置（bias）。

物理信息算子网络的损失函数和 PINN 的损失函数类似，也由两部分组成：

- 数据项损失

  对于初值条件和边值条件等带标签的训练数据，这些数据配点都有对应的函数值，它们的损失为：

$$\mathcal{L}_{Data}(\theta) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \left\| G_\theta(u^{(i)}) y_j - G(u^{(i)}) y_j \right\|^2 \tag{45}$$

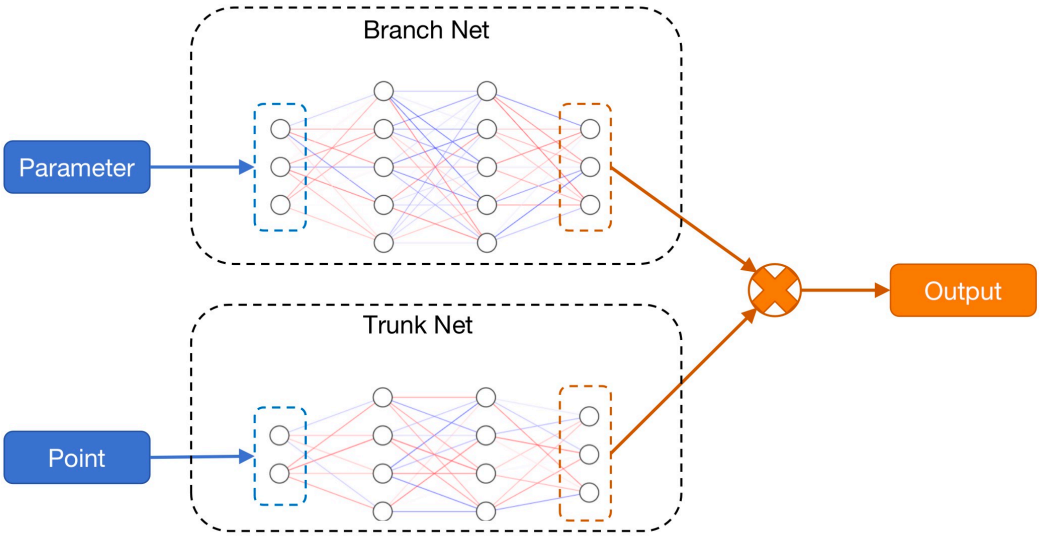


图 2: 非堆叠式算子网络架构

其中，N 表示偏微分方程参数的训练集元素个数，比如偏微分方程的参数是初值函数，那么我们需要准备一组不同的初值函数进行训练；M 表示数据配点训练集元素的个数。这里需要注意的是 $u^{(i)}$ 表示第 i 个训练参数，它是一个函数，但是神经网络无法直接输入一个函数，所以这里输入的是 $u^{(i)}$ 的离散化格式，即一个 m 维的向量

$$u^{(i)} = [u^{(i)}(x_1), u^{(i)}(x_2), u^{(i)}(x_3), \cdots, u^{(i)}(x_m)] \tag{46}$$

其中，$x_1, x_2, x_3, \cdots, x_m$ 是函数 $u(x)$ 定义域上的离散点，一般取等分点即可。

- 物理项损失

  类似于 PINN，物理信息算子网络通过最小化潜在的控制规律（即非线性微分算子）的残差来输出符合物理约束的函数。

$$\mathcal{L}_{Physics}(\theta) = \frac{1}{NK} \sum_{i=1}^{N} \sum_{j=1}^{K} \left\| \mathcal{N}(u^{(i)}, G_\theta(u^{(i)})(y_j)) \right\|^2 \tag{47}$$

其中，N 表示偏微分方程参数的训练集元素个数；K 表示物理配点训练集元素的个数。

所以，最终的损失函数为：

$$\mathcal{L}(\theta) = \lambda_1 \mathcal{L}_{Data}(\theta) + \lambda_2 \mathcal{L}_{Physics}(\theta) \tag{48}$$

其中，λ_1 和 λ_1 是两种损失各占的权重。

2.5.4 训练算法

1. 构造分支网络的训练数据集

(a) 使用高斯径向基函数随机生成 N 个函数，以此作为偏微分方程的参数。这里的参数，可能是初值函数、边值函数或者源项等。

$$\{u^{(i)}(\mathbf{x})\}_{i=1}^{N} = \{u^{(1)}(\mathbf{x}), u^{(2)}(\mathbf{x}), u^{(2)}(\mathbf{x}), \cdots, u^{(N)}(\mathbf{x})\} \tag{49}$$

(b) 对每个参数离散化

对定义域等分，得到离散点 $x_1, x_2, x_3, \cdots, x_m$，每个参数取这些点上的函数值构成向量，作为分支网络的实际输入。记为：

$$B = \begin{bmatrix} u_1^{(1)} & u_2^{(1)} & \cdots & u_m^{(1)} \\ u_1^{(2)} & u_2^{(2)} & \cdots & u_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ u_1^{(N)} & u_2^{(N)} & \cdots & u_m^{(N)} \end{bmatrix} \tag{50}$$

2. 构造主干网络的训练数据集

(a) 数据配点采样

对于初值条件和边值条件等带标签的训练数据，我们一般对定义域进行等间距采样，获取 M 个数据点作为主干网络的输入，并把数据配点对应的函数值作为网络训练的目标值。

$$T_D = \{y_{D1}, y_{D2}, y_{D3}, \cdots, y_{DM}\} \tag{51}$$

(b) 物理配点采样

对偏微分方程的整个定义域，使用拉丁超立方采样，选取 K 个物理配点。由于物理配点是不带目标值的，所以理论上可以采样任意多的点，采取的点越多，训练拟合的效果会更精细，但是这也会增加训练的复杂度，所以这里需要权衡。

$$T_P = \{y_{P1}, y_{P2}, y_{P3}, \cdots, y_{PK}\} \tag{52}$$

3. 整合训练数据集

由于数据配点和物理配点计算损失的方式不一样，所以要分开处理。

将分支网络训练数据集和主干网络的数据配点集做笛卡尔积，得到最终的数据训练集：

$$S_D = B \times T_D \tag{53}$$

将分支网络训练数据集和主干网络的物理配点集做笛卡尔积，得到最终的物理训练集：

$$S_P = B \times T_P \tag{54}$$

4. 构造算子网络

(a) 构造分支网络

分支网络分为堆叠式和非堆叠式的，二者效力等同。但非堆叠式的架构简单，易于编码和训练，所以一般情况下分支网络采用非堆叠式的网络架构，即一个全连接深度神经网络。这个网络的输入维度是偏微分方程参数的离散点数。输出维度是任意的，只需和主干网络的输入维度保持一致即可。

(b) 构造主干网络
主干网络也是一个全连接深度神经网络，输入维度是偏微分方程的时空域维度之和，即空间维度加一。输出维度是任意的，只需和分支网络的输入维度保持一致即可。

(c) 构造聚合网络
由于分支网络和主干网络的输出维度相同，所以可以构造一个单层网络，以主干网络和分支网络的输出为输入，对两个向量进行内积操作，得到一维标量的最终输出。

5. 训练算子网络

(a) 前向传播
将数据训练集和物理训练集分别喂入网络进行前向运算，并根据公式 80和公式 47计算数据项损失和物理项损失；再根据公式 48计算总的损失。

(b) 反向传播
将网络的损失反向传播，计算梯度。

(c) 更新网络参数
根据梯度及学习率，优化网络参数，一般采用 Adam 优化器即可。

上述几个步骤是神经网络训练的一般范式，实际训练过程中，根据具体情况，设置合适的超参数，如：输出维度、网络的宽度和深度、学习率退火策略等。不断重复这几个步骤，直至损失达到目标阈值则停止。

3 理论基础 or 本文工作?

3.1 基于 ST-RFM 的数据同化

3.1.1 ST-RFM 的改进

在 RFM 章节，公式 27解释了微分算子离散化的过程，该过程展示了线性微分算子如何一步步转为矩阵和向量的线性运算，但该过程的结果还包含着单位分解函数，这显然并不完美。虽然文献 [1] 提供了两种单位分解函数，并通过数值算例验证了其效能，但是单位分解函数的构造方式还可能有很多，无法确认哪一种才是最合适的。这里的合适有两层含义：一是能够促进数值解的收敛，在相同的参数下，尽可能达到更高的收敛阶；二是能够适合计算机的高效计算，结构不能太复杂，比如不能有太多的分支条件，否则不利于并行计算等。本文的一项工作就是对该过程做简化，去除单位分解函数，得到 ST-RFM 数值解的简化格式：

$$u_M(\boldsymbol{x},t)=\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n}u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \tag{55}$$

由 $\phi_{n_x,n_t,j}(\boldsymbol{x},t)$ 的定义可知，其是连续可微函数。所以 $u_M(\boldsymbol{x},t)$ 也是连续可微的，故而不需要处理各个子区域的边界问题。这里把单位分解函数去除，虽然形式上看似变得简单了，但是在表现能力上却变得丰富了。因为经过这个改动，相当于把光滑性的构造也交给神经网络了，让神经网络自己去训练出最合适的单位分解函数。此外，由于没有了单位分解函数的

代码分支逻辑，使得 RFM 方法的运算速度更快。在此基础上，我们也能得到微分算子离散化的简单形式：

$$\begin{aligned}\mathcal{L}u(\boldsymbol{x},t) &\approx \mathcal{L}u_M(\boldsymbol{x},t)\\ &= \mathcal{L}\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n} u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t)\\ &= \sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n} \mathcal{L}u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t)\\ &= \sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n} u_{n_x,n_t,j}\mathcal{L}\phi_{n_x,n_t,j}(\boldsymbol{x},t)\end{aligned} \tag{56}$$

3.1.2 使用改进的 ST-RFM 做数据同化

由于改进后的 ST-RFM 具有更快的收敛速度和收敛阶，所以我们基于改进的 ST-RFM 做数据同化。

考虑如下数据同化问题：

$$\begin{cases}\mathcal{L}u(\boldsymbol{x},t) = f(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega\times[0,T]\\ \mathcal{B}u(\boldsymbol{x},t) = g(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega\times[0,T]\\ \mathcal{I}u(\boldsymbol{x},0) = s(\boldsymbol{x}) & \boldsymbol{x}\in\Omega\\ \mathcal{H}u(\boldsymbol{x},t) = h(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega\times[0,T]\end{cases} \tag{57}$$

其中，Ω 是空间域，$[0,T]$ 是时间域，$\mathcal{L}$ 是线性微分算子，$\mathcal{B}$ 是边界算子，$\mathcal{I}$ 是初值算子，这三者构成动力系统。$\mathcal{H}$ 是观测算子，一般在空间的某些固定位置，进行连续或间断观测。接下来基于改进的随机特征方法，解决这一类数据同化问题。步骤如下：

1. 求解区域离散化

   将空间域 Ω 划分为 N_x 个子域 $\{\Omega_n\}_{n=1}^{N_x}$，每个子域 Ω_n 的中心点记为 $\hat{\boldsymbol{x}}_n$。将时间域 $[0,T]$ 划分为 N_t 个子域，即 $[0,T]=[t_0,t_1)\cup[t_1,t_2)\cup\cdots\cup[t_{N_t-1},t_{N_t}]$，每个子域的中心点 $\hat{t}_n=\frac{t_{n-1}+t_n}{2}$。则总的区域数为 $M_p=N_x\cdot N_t$，对第 n 个区域随机初始化 J_n 个随机特征函数。

2. 微分算子离散化

   利用改进的 ST-RFM 微分算子离散化公式 56，可得：

$$\begin{aligned}f(\boldsymbol{x},t) &= \mathcal{L}u(\boldsymbol{x},t)\\ &\approx \mathcal{L}u_M(\boldsymbol{x},t)\\ &= \sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n} u_{n_x,n_t,j}\mathcal{L}\phi_{n_x,n_t,j}(\boldsymbol{x},t)\\ &= \sum_{n=1}^{M_p}\sum_{j=1}^{J_n} u_{nj}\mathcal{L}\phi_{nj}(\boldsymbol{y}) \quad \Big[\boldsymbol{y}\triangleq(\boldsymbol{x},t)\Big]\end{aligned} \tag{58}$$

我们在 $\Omega \times [0,T]$ 上取一组配点 $\boldsymbol{y}_P^{(1)}, \boldsymbol{y}_P^{(2)}, \cdots, \boldsymbol{y}_P^{(N_P)}$，则有

$$\begin{bmatrix} \mathcal{L}\phi_{11}(\boldsymbol{y}_P^{(1)}) & \cdots & \mathcal{L}\phi_{1J_1}(\boldsymbol{y}_P^{(1)}) & \cdots & \mathcal{L}\phi_{M_p1}(\boldsymbol{y}_P^{(1)}) & \cdots & \mathcal{L}\phi_{M_pJ_{M_p}}(\boldsymbol{y}_P^{(1)}) \\ \mathcal{L}\phi_{11}(\boldsymbol{y}_P^{(2)}) & \cdots & \mathcal{L}\phi_{1J_1}(\boldsymbol{y}_P^{(2)}) & \cdots & \mathcal{L}\phi_{M_p1}(\boldsymbol{y}_P^{(2)}) & \cdots & \mathcal{L}\phi_{M_pJ_{M_p}}(\boldsymbol{y}_P^{(2)}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_{11}(\boldsymbol{y}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_{1J_1}(\boldsymbol{y}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_{M_p1}(\boldsymbol{y}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_{M_pJ_{M_p}}(\boldsymbol{y}_P^{(N_P)}) \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix} \approx \begin{bmatrix} f(\boldsymbol{y}_P^{(1)}) \\ f(\boldsymbol{y}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{y}_P^{(N_P)}) \end{bmatrix} \tag{59}$$

把上述公式简记为：$Pu = f$

3. 边界算子离散化

我们在 $\partial\Omega \times [0,T]$ 上取一组配点 $\boldsymbol{y}_B^{(1)}, \boldsymbol{y}_B^{(2)}, \cdots, \boldsymbol{y}_B^{(N_B)}$，则有

$$\begin{bmatrix} \phi_{11}(\boldsymbol{y}_B^{(1)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_B^{(1)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_B^{(1)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_B^{(1)}) \\ \phi_{11}(\boldsymbol{y}_B^{(2)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_B^{(2)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_B^{(2)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_B^{(2)}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{11}(\boldsymbol{y}_B^{(N_B)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_B^{(N_B)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_B^{(N_B)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_B^{(N_B)}) \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix} \approx \begin{bmatrix} g(\boldsymbol{y}_B^{(1)}) \\ g(\boldsymbol{y}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{y}_B^{(N_B)}) \end{bmatrix} \tag{60}$$

把上述公式简记为：$Bu = g$

4. 初值算子离散化

我们在 $\Omega \times [0]$ 上取一组配点 $\boldsymbol{y}_I^{(1)}, \boldsymbol{y}_I^{(2)}, \cdots, \boldsymbol{y}_I^{(N_I)}$，则有

$$\begin{bmatrix} \phi_{11}(\boldsymbol{y}_I^{(1)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_I^{(1)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_I^{(1)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_I^{(1)}) \\ \phi_{11}(\boldsymbol{y}_I^{(2)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_I^{(2)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_I^{(2)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_I^{(2)}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{11}(\boldsymbol{y}_I^{(N_I)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_I^{(N_I)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_I^{(N_I)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_I^{(N_I)}) \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix} \approx \begin{bmatrix} s(\boldsymbol{y}_I^{(1)}) \\ s(\boldsymbol{y}_I^{(2)}) \\ \vdots \\ s(\boldsymbol{y}_I^{(N_I)}) \end{bmatrix} \tag{61}$$

把上述公式简记为：$Iu = s$

5. 观测算子离散化

我们在 $\Omega \times [0,T]$ 上取一组观测数据 $\boldsymbol{y}_H^{(1)}, \boldsymbol{y}_H^{(2)}, \cdots, \boldsymbol{y}_H^{(N_H)}$，则有

$$\begin{bmatrix} \phi_{11}(\boldsymbol{y}_H^{(1)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_H^{(1)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_H^{(1)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_H^{(1)}) \\ \phi_{11}(\boldsymbol{y}_H^{(2)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_H^{(2)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_H^{(2)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_H^{(2)}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_{11}(\boldsymbol{y}_H^{(N_H)}) & \cdots & \phi_{1J_1}(\boldsymbol{y}_H^{(N_H)}) & \cdots & \phi_{M_p1}(\boldsymbol{y}_H^{(N_H)}) & \cdots & \phi_{M_pJ_{M_p}}(\boldsymbol{y}_H^{(N_H)}) \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix} \approx \begin{bmatrix} h(\boldsymbol{y}_H^{(1)}) \\ h(\boldsymbol{y}_H^{(2)}) \\ \vdots \\ h(\boldsymbol{y}_H^{(N_H)}) \end{bmatrix} \tag{62}$$

把上述公式简记为：$Hu = h$

6. 转换为线性最小二乘问题

将上述方程组合并为一个大的线性方程组：

$$\begin{bmatrix} P \\ B \\ I \\ H \end{bmatrix} u = \begin{bmatrix} f \\ g \\ s \\ h \end{bmatrix} \tag{63}$$

至此，我们将数据同化的求解问题转换成了形如 $Au = f$ 的线性方程组求解问题，其中 u 是待求解的参数。这种问题可以采用线性最小二乘方法进行求解。

7. 设置损失函数和罚参数

损失函数在配点上进行计算，使用强形式在配点上构造损失函数：

$$\begin{aligned} Loss = & \sum_{\boldsymbol{y}_i \in \Omega \times [0,T]} \lambda_i \|\mathcal{L}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{y}_i) - \boldsymbol{f}(\boldsymbol{y}_i)\|_2^2 + \sum_{\boldsymbol{y}_j \in \partial\Omega \times [0,T]} \lambda_j \|\mathcal{B}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{y}_j) - \boldsymbol{g}(\boldsymbol{y}_j)\|_2^2 + \\ & \sum_{\boldsymbol{y}_k \in \Omega \times [0]} \lambda_k \|\mathcal{I}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{y}_k) - \boldsymbol{s}(\boldsymbol{y}_k)\|_2^2 + \sum_{\boldsymbol{y}_l \in \Omega \times [0,T]} \lambda_l \|\mathcal{H}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{y}_l) - \boldsymbol{h}(\boldsymbol{y}_l)\|_2^2 \end{aligned} \tag{64}$$

其中，$\lambda_i, \lambda_j, \lambda_k, \lambda_l$ 是罚参数，由公式 32，我们知道罚参数为 $\lambda_i = \frac{c_i}{max(A_i, -A_i)}$，其中 c_i 是关键的缩放因子，当没有具体的先验知识的情况下，取常数即可。但是对于数据同化问题，一般而言观测数据更为可信，所以一般会把观测损失项的罚参数的缩放因子设置的大一些，以增加观测数据的影响权重。

3.2 基于 DeepONet 的数据同化

由章节 2.5可以知道，深度算子网络（DeepONet）可以作为参数化的偏微分方程的解算子。但是该方法只适用于单个偏微分方程，不适用于偏微分方程组的情况，对此我们进行扩展和改进，并基于改进的算子网络做数据同化问题。

3.2.1 DeepONet 的改进

观察章节 2.5描述的算子网络架构，我们可以发现其主要分为三个部分：主干网络、分支网络和聚合网络。主干网络和分支网络的输出维度是任意的，因而它们的表现力是丰富的。但是聚合网络进行了降维操作，通过内积操作把目标结果降到了一维，所以我们对聚合网络进行改造。

假设主干网络和分支网络分别输出两个 p 维的向量：$\mathbf{a} = (a_1, a_2, a_3, \cdots, a_p)^T$ 和 $\mathbf{b} = (b_1, b_2, b_3, \cdots, b_p)^T$，我们将内积操作进行拆解：

1. 逐元素相乘

$$\mathbf{c} = (a_1 \times b_1, a_2 \times b_2, a_3 \times b_3, \cdots, a_p \times b_p)^T \tag{65}$$

2. 元素聚合（求和）

$$Output = \sum_{i=1}^{p} c_i \tag{66}$$

可以看到内积的降维操作主要在第二步，所以我们把第二步去掉，在第一步的后面拼接一个单层的全连接网络，将第一步的输出向量作为网络的输入，而输出维度等于我们偏微分方程的个数，假设为 d。

$$\mathbf{Output} = \mathbf{W} \cdot \mathbf{c} + \mathbf{b} \tag{67}$$

这个改动是向下兼容的，因为当偏微分方程的个数是 1 的时候，只需把修改后的聚合网络67的权重设置为 1，偏置设置为 0，即可退化为公式 66。也就是说，改进前的算子网络架构是改进后的架构的一种特殊情况。新的网络架构如图

改进后的算子网络表现能力变得丰富，不仅可以作为单个偏微分方程的解算子，还可以作为偏微分方程组的解算子，这一点也通过数值算例得到了验证。

但是仔细观察，算子网络还存在一个缺陷，就是主干网络和分支网络是两个独立的网络。这就会带来训练效率的问题，因为当一组训练数据分别输入主干网络和分支网络的时候，它们各自进行前向计算，当某一个网络计算完成后，必须等待另一个网络计算完成，才能进行后续的聚合计算。这显然不符合神经网络并行计算的设计初衷，所以我们对这一块进行修改。

在章节 2.3.4中，我们介绍了两种神经网络架构：公式 35代表的 STC 架构和公式 36代表的 SoV 架构，这两种网络架构在理论上和数值算例上被证明是等效的。恰好，我们改进后的算子网络与 SoV 架构相同，那么很自然的可以对算子网络进一步改进，切换为 STC 架构。这种改动，把我们复杂的算子网络架构退化为普通的全连接神经网络。

这样一来，就避免了多个子网络之间在训练过程中的阻塞等待问题。算子网络架构由最初的算子通用逼近定理对应的堆叠式架构，到非堆叠式架构，再到类 SoV 架构，最后到类 STC 架构，即最朴素的全连接网络。虽然结构变简单了，但是表现力却变得丰富了，训练速度也提升了，真是让人不禁感慨：大道至简。

此时的训练算法也和 PINN 高度相似，唯一的区别就是训练数据集的构造不一样。PINN 的输入只包括偏微分方程定义域上的数据配点和物理配点，而 DeepONet 的输入不仅包括数据配点和物理配点的信息，还需要涵盖偏微分方程的参数信息。简单来说，就是把参数训练集和配点训练集做笛卡尔积，得到最终的训练集。最终简化的算子网络架构如图

3.2.2 使用改进的 DeepONet 做数据同化

由章节 2.5可知，深度算子神经网络可以作为参数化偏微分方程的解算子，这里的参数包含一切可以影响偏微分方程解的元素，比如初值条件、边值条件、源项等。

在数据同化领域，动力系统往往由描述物理定律的偏微分方程表示。动力系统随着时间演变，其真实状态往往需要依靠观测数据进行推演，即不同的观测数据对应着不同的系统状态，对应着不同的偏微分方程的解。所以可以把观测数据作为偏微分方程的参数，使用算子神经网络作为数据同化问题的解算子。

由章节 3.2.1可知，改进后的算子网络具有更简单的网络架构和更快的训练速度，所以我们基于改进的算子网络做数据同化。

考虑如下数据同化问题：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x},t) = f(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega \times [0,T] \\ \mathcal{B}u(\boldsymbol{x},t) = g(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega \times [0,T] \\ \mathcal{I}u(\boldsymbol{x},0) = s(\boldsymbol{x}) & \boldsymbol{x} \in \Omega \\ \mathcal{H}u(\boldsymbol{x},t) = h(\boldsymbol{x},t) & \boldsymbol{x},t \in \Omega \times [0,T] \end{cases} \tag{68}$$

其中，Ω 是空间域，$[0,T]$ 是时间域，$\mathcal{L}$ 是非线性微分算子，$\mathcal{B}$ 是边界算子，$\mathcal{I}$ 是初值算子，这三者构成动力系统。$\mathcal{H}$ 是观测算子，一般在空间的某些固定位置，进行连续或间断观测。接下来基于简化的算子网络，解决这一类数据同化问题。算法如下：

1. 构造网络的训练数据集

   (a) 构造配点集

   配点集包括两部分：数据配点和物理配点。数据配点指的是那些带有目标值的配点，在训练的时候通过计算网络输出值和目标值的残差来计算损失项。物理配点指的是偏微分方程定义域内的任意点，这些点大部分情况下是不带目标值的，通过物理定律进行约束，即通过计算偏微分方程等式两边的残差来计算损失项。

   数据配点包括三部分：初值配点、边值配点和观测配点。初值配点和边值配点不必多说，就是满足初值条件和边值条件的配点。对于初边值配点，一般使用等间距采样策略即可，即对初值条件和边值条件的定义域分别等分，取等分点作为配点。我们统一记为：

   $$\boldsymbol{P}_d = \{\mathbf{y}_{d1}, \mathbf{y}_{d2}, \mathbf{y}_{d3}, \cdots, \mathbf{y}_{dM}\} \tag{69}$$

   观测配点就是实际采取观测值的配点，这些配点是带有目标值的，目标值就是观测值。从这个角度看，观测配点和初边值配点没什么区别，但是观测数据在数据同化中还有别的重要含义，所以我们它记为：

   $$\boldsymbol{P}_o = \{\mathbf{y}_{o1}, \mathbf{y}_{o2}, \mathbf{y}_{o3}, \cdots, \mathbf{y}_{oK}\} \tag{70}$$

   物理配点一般使用拉丁超立方采样策略，在偏微分方程的整个定义域内进行随机采点，记为：

   $$\boldsymbol{P}_p = \{\mathbf{y}_{p1}, \mathbf{y}_{p2}, \mathbf{y}_{p3}, \cdots, \mathbf{y}_{pL}\} \tag{71}$$

   (b) 构造参数集

   数据同化问题中，每一组观测数据对应偏微分方程的一个参数，因为一组观测数据对应着方程的一个解。观测数据就是观测配点上对应的观测值，记为：

   $$\boldsymbol{H} = \{H_1, H_2, H_3, \cdots, H_N\} \tag{72}$$

   其中，$H_i = \{h_1^{(i)}, h_2^{(i)}, h_3^{(i)}, \cdots, h_m^{(i)}\}$ 表示观测配点上的一组观测数据。

   (c) 构造训练集

由于改进后的算子网络实际上已经退化为一个全连接网络，不再分为主干网络和分支网络，训练集也不再分为两部分输入网络。所以“算子”二字就体现在网络输入的构造上，它是由配点训练集和参数训练集做笛卡尔积得到。

记数据配点训练集为：

$$\boldsymbol{X}_d = \boldsymbol{P}_d \times \boldsymbol{H} \tag{73}$$

记观测配点训练集为：

$$\boldsymbol{X}_o = \boldsymbol{P}_o \times \boldsymbol{H} \tag{74}$$

记物理配点训练集为：

$$\boldsymbol{X}_p = \boldsymbol{P}_p \times \boldsymbol{H} \tag{75}$$

所以最终的训练集为：

$$\boldsymbol{X} = \boldsymbol{X}_d + \boldsymbol{X}_o + \boldsymbol{X}_p \tag{76}$$

2. 构造神经网络

改进后的算子网络架构和 PINN 在形式上是一致的，只需要构造一个全连接的深度神经网络即可，我们仍然把改进后的算子网络记为 G_θ，θ 为网络的参数；真解算子记为 G。网络的输入维度等于观测配点的个数加上配点的维度，网络的输出维度等于偏微分方程组的个数。

关于网络的其他部分，文献 [13] 给出了一些关于 PINN 的网络架构的经验，可以基于这个在训练中调整：

(a) 网络的宽度一般设置为 128 到 512 个神经元；

(b) 网络的深度一般设置为 3 到 6 层；

(c) 激活函数一般选取双曲正切函数（$Tanh$），如果对求解问题有先验知识的话，比如在做过频谱分析的基础上，可以采用正弦激活函数（sin），但是千万不要用 $ReLU$，因为它的二阶导数恒为 0；

(d) 网络参数的初始化采用 Glorot 策略，这种策略能够有效地避免梯度消失和梯度爆炸的问题；

3. 训练算子网络

(a) 前向传播

将数据配点训练集 73喂入网络进行前向运算，计算数据项损失：

$$\mathcal{L}_{Data}(\theta) = \frac{1}{NM}\sum_{i=1}^{NM}\left\|G_\theta(\boldsymbol{X}_d^{(i)}) - G(\boldsymbol{X}_d^{(i)})\right\|^2 \tag{77}$$

将观测配点训练集 74喂入网络进行前向运算，计算观测项损失：

$$\mathcal{L}_{Obs}(\theta) = \frac{1}{NK}\sum_{i=1}^{NK}\left\|G_\theta(\boldsymbol{X}_o^{(i)}) - h(\boldsymbol{X}_o^{(i)})\right\|^2 \tag{78}$$

将物理配点训练集 75喂入网络进行前向运算，计算物理项损失：

$$\mathcal{L}_{Physics}(\theta) = \frac{1}{NL}\sum_{i=1}^{NL}\left\|\mathcal{L}G_\theta(\boldsymbol{X}_p^{(i)}) - f(\boldsymbol{X}_p^{(i)})\right\|^2 \tag{79}$$

将所有损失加权求和，得到最终损失：

$$\mathcal{L}(\theta) = \lambda_d \cdot \mathcal{L}_{Data}(\theta) + \lambda_o \cdot \mathcal{L}_{Obs}(\theta) + \lambda_p \cdot \mathcal{L}_{Physics}(\theta) \tag{80}$$

其中，$\lambda_d, \lambda_o, \lambda_p$ 分别为数据损失、观测损失、物理损失所占的权重。

(b) 反向传播
将网络的损失反向传播，计算梯度。

(c) 更新网络参数
根据梯度及学习率，优化网络参数。这里涉及到两个设置：

i. 优化器
在使用神经网络解偏微分方程领域，Adam（Adaptive Moment Estimation）是一种常用的优化器。Adam 优化器结合了动量（momentum）和自适应学习率的思想。它的主要思路是在更新权重的时候，结合梯度的一阶矩估计（动量项）和二阶矩估计（自适应学习率项）。

ii. 学习率退火策略
学习率退火是一种优化算法中用于调整学习率的策略，旨在提高模型在训练过程中的性能。学习率退火策略的目标是在训练的不同阶段使用不同的学习率，以更好地适应模型的收敛和泛化需求。
训练神经网络的时候，往往一开始的随机初始点离最小值点比较远，所以需要将学习率设置的大一些，以使损失函数能够尽快逼近最小值。当训练到后期的时候，学习率需要设置的小一些，因为如果太大，则损失值会在最小值周围振荡，导致无法收敛到最小值，甚至远离最小值点。此时最好采用学习率退火策略，如 PyTorch 的 ReduceLROnPlateau。

重复上述步骤，直至达到指定条件停止。停止的条件可以是达到指定的循环训练次数，也可以是损失达到某个阈值。

4 数值实验

4.1 改进的 RFM 验证

我们使用一维 Helmholtz 方程验证改进的 RFM，方程如下：

$$\begin{cases} \frac{d^2u(x)}{dx^2} - \lambda u(x) = f(x) \quad x \in [0,8] \\ u(0) = c_1, \quad u(8) = c_2 \end{cases} \tag{81}$$

表 1: 一维 Helmholtz 方程三种单位分解策略的比较

M_p	ψ^a		ψ^b		无 ψ	
	L^∞ error	L^2 error	L^∞ error	L^2 error	L^∞ error	L^2 error
2	39.58	10.55	8.00	1.58	0.34	0.057
4	1.20E-2	3.09E-3	1.26E-4	1.51E-05	5.48E-08	1.11E-08
8	1.77E-05	4.50E-06	2.14E-07	3.68E-08	6.43E-10	1.04E-10
16	1.66E-08	4.75E-09	1.38E-09	2.22E-10	3.17E-13	1.46E-13
32	2.15E-09	8.69E-10	1.27E-10	2.64E-11	3.89E-14	1.31E-14

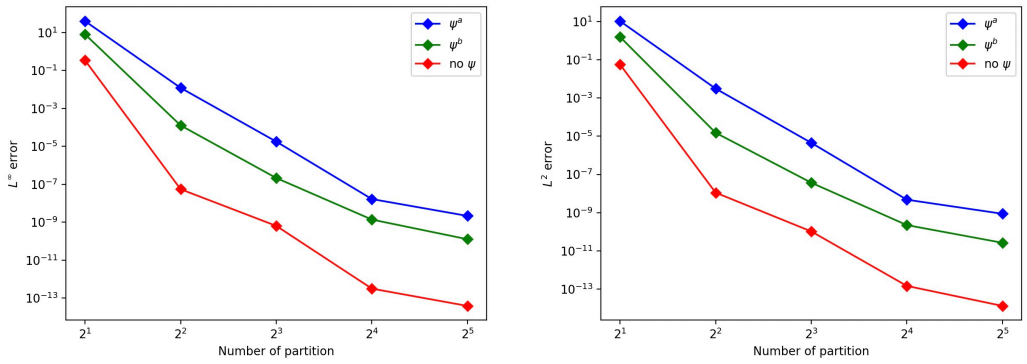
我们取 $\lambda = 4$，真解 $u(x) = \sin(3\pi x + \frac{3\pi}{20})\cos(2\pi x + \frac{\pi}{10}) + 2$，则 c_1, c_2 和 $f(x)$ 可相应的计算出来：

$$\begin{cases} f(x) = -13\pi^2 \sin(3\pi x + \frac{3\pi}{20})\cos(2\pi x + \frac{\pi}{10}) - 12\pi^2 \cos(3\pi x + \frac{3\pi}{20})\sin(2\pi x + \frac{\pi}{10}) \\ u(0) = u(8) = \sin\frac{3\pi}{20}\cos\frac{\pi}{10} + 2 \end{cases} \tag{82}$$

对方程使用 RFM 进行数值求解，超参数的设置如下：

- 每个子区域的特征函数的数量为 $J_n = 50$；
- 每个子区域的配点的数量为 $Q_n = 50$；
- 子区域划分的个数依次取 $M_p = 2, 4, 8, 16, 32$
- 单位分解函数的策略分别为：ψ^a、ψ^b 和无 ψ；

计算结果见图 3 和表格 1，可以看出在相同超参数使用 RFM 求解 Helmholtz 方程的情况下，无 ψ 策略相比于使用 ψ^a 和 ψ^b 作为单位分解函数，收敛速度更快，并且 L^∞ 误差和 L^2 误差降低了好几个阶。数值结果验证了改进后的 RFM 方法对于静态方程的有效性。



(a) L^∞ 误差　　(b) L^2 误差

图 3: 改进后的 RFM 效果验证。横坐标是子区域划分的个数，纵坐标是误差，横纵坐标都使用对数刻度。

4.2 改进的 ST-RFM 验证

我们使用一维扩散方程验证改进的 ST-RFM，方程如下：

$$\begin{cases} \frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} = f(x,t) \quad (x,t) \in [0,5] \times [0,1] \\ u(0,t) = g_1(t) \quad t \in [0,1] \\ u(5,t) = g_2(t) \quad t \in [0,1] \\ u(x,0) = h(x) \quad x \in [0,5] \end{cases} \tag{83}$$

我们取定真解为 $u(x,t) = [2\cos(\pi x + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi x - \frac{3\pi}{5})][2\cos(\pi t + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi t - \frac{3\pi}{5})]$，则 $f(x,t), g_1(t), g_2(t), h(x)$ 可相应的计算出来：

$$\begin{cases} f(x,t) & = [2\cos(\pi x + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi x - \frac{3\pi}{5})][-2\pi\sin(\pi t + \frac{\pi}{5}) - 3\pi\sin(2\pi t - \frac{3\pi}{5})] \\ & \quad +\nu[2\pi^2\cos(\pi x + \frac{\pi}{5}) + 6\pi^2\cos(2\pi x - \frac{3\pi}{5})][2\cos(\pi t + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi t - \frac{3\pi}{5})] \\ g_1(t) & = [2\cos\frac{\pi}{5} + \frac{3}{2}\cos\frac{3\pi}{5}][2\cos(\pi t + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi t - \frac{3\pi}{5})] \\ g_2(t) & = [-2\cos\frac{\pi}{5} + \frac{3}{2}\cos\frac{3\pi}{5}][2\cos(\pi t + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi t - \frac{3\pi}{5})] \\ h(x) & = [2\cos(\pi x + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi x - \frac{3\pi}{5})][2\cos\frac{\pi}{5} + \frac{3}{2}\cos\frac{3\pi}{5}] \end{cases} \tag{84}$$

其中，扩散系数取 $\nu = 0.01$。

对方程使用 ST-RFM 进行数值求解，超参数的设置如下：

- 空间维度的划分数量为 $N_x = 5$；
- 时间维度的划分数量为 $N_t = 2$；
- 每个子区域的空间维度的配点数量为 $Q_x = 30$；
- 每个子区域的时间维度的配点数量为 $Q_t = 30$；
- 每个子区域的特征函数的数量依次取 $J_n = 50, 100, 150, 200, 250$；
- 单位分解函数的策略分别为：ψ^a、ψ^b 和无 ψ；

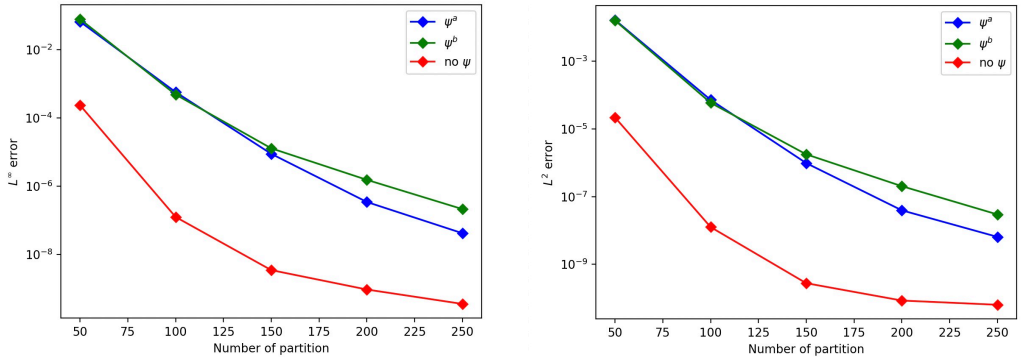
所以每个子区域的配点数量为 $Q_n = Q_x \times Q_t = 900$，求解结果见图 4 和表格 2。可见无 ψ 的策略下收敛速度更快，在同等超参数的情况下相较于 ψ^a 和 ψ^b，误差小几个数量级。

通过数值算例可以看到，去除单位分解函数，确实大大提高了训练的收敛速度和收敛精度。

4.3 使用改进的 ST-RFM 解数据同化问题

我们使用一维对流扩散方程做数据同化问题，原始方程如下：

$$\begin{cases} \frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2} + u = f(x,t) \quad (x,t) \in [0,5] \times [0,1] \\ u(0,t) = g_1(t) \quad t \in [0,1] \\ u(5,t) = g_2(t) \quad t \in [0,1] \\ u(x,0) = h(x) \quad x \in [0,5] \end{cases} \tag{85}$$



(a) L^∞ 误差　　(b) L^2 误差

图 4: 改进后的 RFM 效果验证。横坐标是子区域划分的个数，纵坐标是误差，纵坐标使用对数刻度。

表 2: 一维 Diffusion 方程三种单位分解策略的比较

J_n	ψ^a		ψ^b		无 ψ	
	L^∞ error	L^2 error	L^∞ error	L^2 error	L^∞ error	L^2 error
50	6.60E-02	1.64E-02	7.82E-02	1.58E-02	2.37E-04	2.18E-05
100	5.70E-04	7.17E-05	4.83E-04	5.87E-05	1.26E-07	1.26E-08
150	8.89E-06	9.73E-07	1.29E-05	1.77E-06	3.51E-09	2.78E-10
200	3.46E-07	3.93E-08	1.54E-06	2.03E-07	9.48E-10	8.54E-11
250	4.21E-08	6.46E-09	2.14E-07	2.97E-08	3.59E-10	6.41E-11

我们取扩散系数 $\nu = 0.01$，对流速度 $v = 0.01$，真解 $u(x,t) = e^{-t}\sin(\pi(x + t + x \cdot t))$，则 $f(x,t), g_1(t), g_2(t), h(x)$ 可相应的计算出来。

接下来我们使用高斯径向基函数生成一个值域为 $[-0.1, 0.1]$ 的波函数，作为噪声添加到初值函数和边值函数。

$$\begin{cases} \frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} + u = f(x,t) \quad (x,t) \in [0,5] \times [0,1] \\ u(0,t) = g_1(t) + RBF(t) \quad t \in [0,1] \\ u(5,t) = g_2(t) + RBF(t) \quad t \in [0,1] \\ u(x,0) = h(x) + RBF(x) \quad x \in [0,5] \end{cases} \tag{86}$$

并使用改进的 ST-RFM 进行数值求解，超参数的设置如下：

- 空间维度的划分数量为 $N_x = 5$；
- 时间维度的划分数量为 $N_t = 2$；
- 每个子区域的空间维度的配点数量为 $Q_x = 10$；
- 每个子区域的时间维度的配点数量为 $Q_t = 10$；

表 3: 一维对流扩散方程带噪声的误差情况

J_n	L^∞ error	L^2 error
50	1.01E-01	2.89E-02
100	2.10E-01	6.29E-02
150	5.07E-01	7.85E-02
200	2.89E-01	7.29E-02

表 4: 一维对流扩散方程数据同化的误差情况

J_n	L^∞ error	L^2 error
50	1.34E-02	5.36E-04
100	2.57E-04	6.63E-06
150	1.41E-04	3.57E-06
200	8.57E-05	1.90E-06

- 每个子区域的特征函数的数量依次取为 $J_n = 50, 100, 150, 200$;

所以每个子区域的配点数量为 $Q_n = Q_x \times Q_t = 100$，运行结果见表格 3，可以看到误差基本稳定在固定的阶，最大范数误差的阶数是 10^{-1}，二范数误差的阶数是 10^{-2}。图 5 展示了 $J_n = 150$ 时，真解和带噪声的数值解的对比。

接下来我们引入观测，使用观测配点对应的真解作为观测值。此外，由于我们对观测数据是信任的，所以需要对损失函数公式 64里面的罚参数进行调整，调整的方法是通过设置公式 32里面的缩放因子 c_i，具体修改如下:

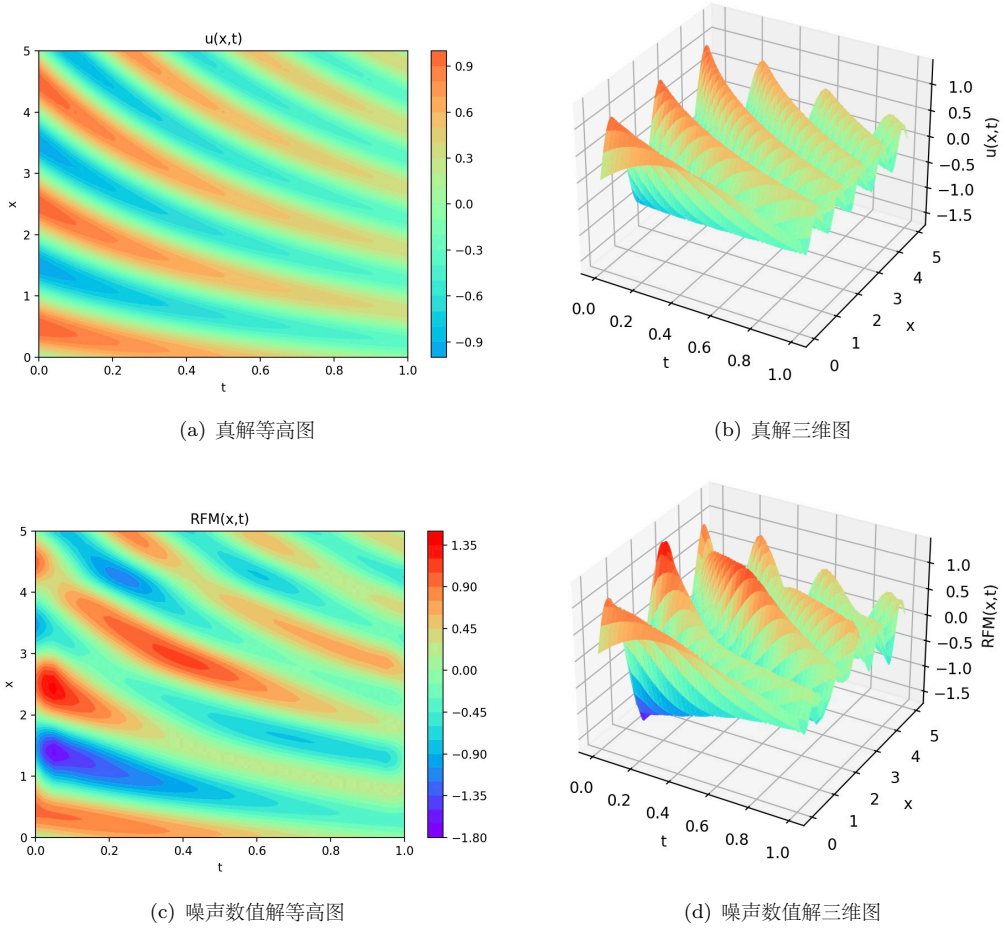
1. 观测点为 $X = \{0.1, 0.2, 0.3, \cdots, 4.8, 4.9\}$;
2. 观测时间为 $T = \{0.1, 0.2, 0.3, \cdots, 0.9, 1.0\}$;
3. 将观测配点和物理配点的缩放因子设置为 100;
4. 将初值配点和边界配点的缩放因子设置为 10^{-5};

运行结果见表格 4，可以看到引入观测后，模型的输出和真解的误差大大减少，系统的演变向真实状态靠近。图 6 展示了 $J_n = 150$ 时，真解和数据同化解的对比。

参考文献

[1] Jingrun Chen, Xurong Chi, Weinan E, and Zhouwang Yang. Bridging traditional and machine learning-based algorithms for solving pdes: The random feature method. 1(2):1–31, Jul 2022.

[2] Jingrun Chen, Weinan E, and Yixin Luo. The random feature method for time-dependent problems. 1:1–26, Apr 2023.



(a) 真解等高图　(b) 真解三维图

(c) 噪声数值解等高图　(d) 噪声数值解三维图

图 5: 一维对流扩散方程引入噪声后的情形

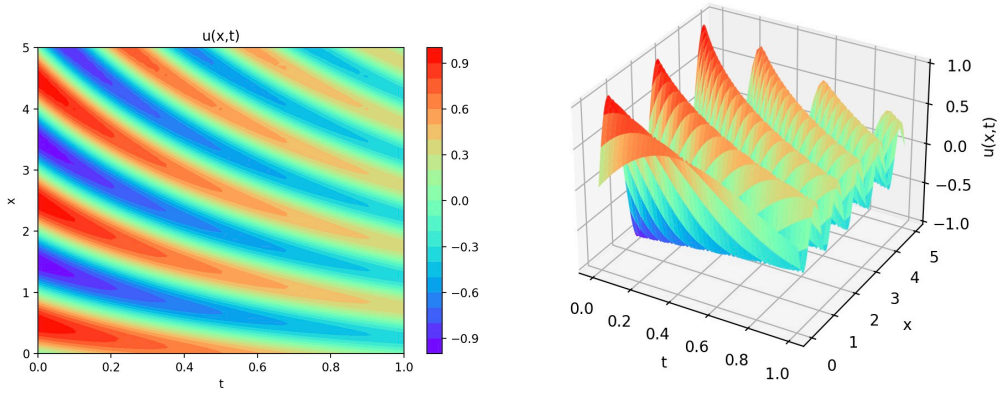
[3] T. Chen and H. Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918, 1993.

[4] T. Chen and H. Chen. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. *IEEE Transactions on Neural Networks*, 6(4):904–910, 1995.

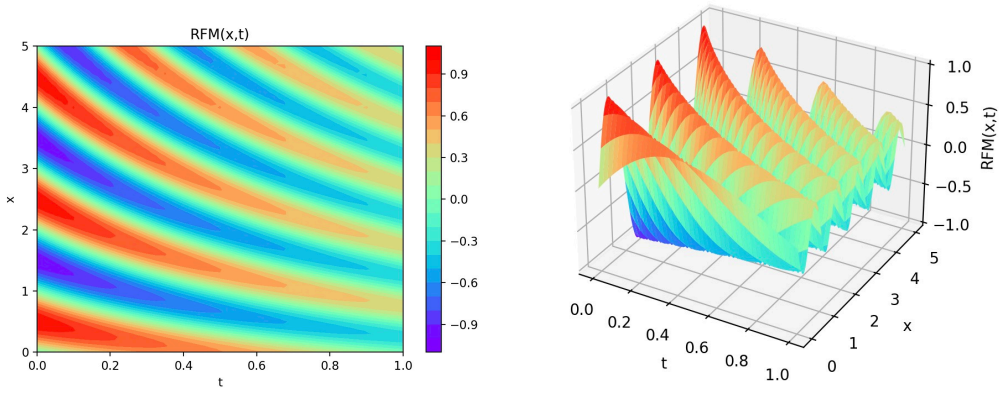
[5] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[6] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[7] Kody Law, Andrew Stuart, and Konstantinos Zygalakis. *Data Assimilation: A Mathe-*



(a) 真解等高图　　(b) 真解三维图



(c) 噪声数值解等高图　　(d) 噪声数值解三维图

图 6: 一维对流扩散方程数据同化的效果

matical Introduction. Texts in Applied Mathematics. Springer, Cham Heidelberg New York Dordrecht London, 2015. ISSN 0939-2475, ISSN 2196-9949 (electronic).

[8] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint*, Apr 2020. arXiv:1910.03193v3 [cs.LG] 15 Apr 2020.

[9] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. 2019. ArXiv preprint arXiv:1907.04502v1.

[10] H. N. Mhaskar and N. Hahm. Neural networks for functional approximation and system identification. *Neural Computation*, 9(1):143–159, 1997.

[11] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Article history: Received 13 June 2018; Received in revised form 26 October 2018; Accepted 28 October 2018; Available online 3 November 2018.

[12] F. Rossi and B. Conan-Guez. Functional multi-layer perceptron: A non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60, 2005.

[13] Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. An expert' s guide to training physics-informed neural networks. *arXiv preprint*, Aug 2023. arXiv:submit/5061730 [cs.LG] 16 Aug 2023.