基于神经网络的数据同化研究

MG21210021 李庆春

2023 年 10 月 26 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

1.1 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：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x}) = f(\boldsymbol{x}) & \boldsymbol{x} \in \Omega \\ \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}) & \boldsymbol{x} \in \partial\Omega \end{cases} \tag{1}$$

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。

1.1.1 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：

$$u_M(\boldsymbol{x}) = \sum_{m=1}^{M} u_m \phi_m(\boldsymbol{x}) \tag{2}$$

$$\phi_m(\boldsymbol{x}) = \sigma(\boldsymbol{k}_m \cdot \boldsymbol{x} + b_m) \tag{3}$$

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

$$\begin{bmatrix} \mathcal{L}\phi_1(\boldsymbol{x}_P^{(1)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(1)}) \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(2)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{L}\phi_1(\boldsymbol{x}_P^{(N_P)}) & \mathcal{L}\phi_2(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\phi_M(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} \approx \begin{bmatrix} f(\boldsymbol{x}_P^{(1)}) \\ f(\boldsymbol{x}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \tag{4}$$

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

$$\begin{bmatrix} \phi_1(\boldsymbol{x}_B^{(1)}) & \phi_2(\boldsymbol{x}_B^{(1)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(1)}) \\ \phi_1(\boldsymbol{x}_B^{(2)}) & \phi_2(\boldsymbol{x}_B^{(2)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_B^{(N_B)}) & \phi_2(\boldsymbol{x}_B^{(N_B)}) & \cdots & \phi_M(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} \approx \begin{bmatrix} g(\boldsymbol{x}_B^{(1)}) \\ g(\boldsymbol{x}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \tag{5}$$

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 4和 5 整合成一个线性方程组:

$$\begin{bmatrix} P \\ B \end{bmatrix} u \approx f \tag{6}$$

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。

1.1.2 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 $\{\boldsymbol{r}_n\}_{n=1}^{M_p}$。

$$\begin{aligned}\hat{\boldsymbol{x}}_n &= (\hat{x}_{n1}, \hat{x}_{n2}, \cdots, \hat{x}_{nd_x})^T \\ &= (\frac{x_{n1\_max} + x_{n1\_min}}{2}, \frac{x_{n2\_max} + x_{n2\_min}}{2}, \cdots, \frac{x_{nd_x\_max} + x_{nd_x\_min}}{2})^T\end{aligned} \tag{7}$$

$$\begin{aligned}\boldsymbol{r}_n &= (r_{n1}, r_{n2}, \cdots, r_{nd_x})^T \\ &= (\frac{x_{n1\_max} - x_{n1\_min}}{2}, \frac{x_{n2\_max} - x_{n2\_min}}{2}, \cdots, \frac{x_{nd_x\_max} - x_{nd_x\_min}}{2})^T\end{aligned} \tag{8}$$

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

$$l_n(\boldsymbol{x}) = \frac{1}{\boldsymbol{r}_n}(\boldsymbol{x} - \hat{\boldsymbol{x}}_n), \quad n = 1, 2, \cdots, M_p, \quad \boldsymbol{x} \in \Omega \tag{9}$$

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：

$$\phi_{nj}(\boldsymbol{x}) = \sigma(\boldsymbol{k}_{nj}\boldsymbol{x} + b_{nj}), \quad j = 1, \cdots, J_n. \tag{10}$$

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：

$$\psi_n^a(x) = \mathbb{I}_{[-1,1]}(x) \tag{11a}$$

$$\psi_n^b(x) = \begin{cases} \frac{1+\sin(2\pi x)}{2} & -\frac{5}{4} \le x < -\frac{3}{4}, \\ 1 & -\frac{3}{4} \le x < \frac{3}{4}, \\ \frac{1-\sin(2\pi x)}{2} & \frac{3}{4} \le x < \frac{5}{4}, \\ 0 & otherwise. \end{cases} \tag{11b}$$

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：

$$u_M(\boldsymbol{x}) = \sum_{n=1}^{M_p} \psi_n(l_n(\boldsymbol{x})) \sum_{j=1}^{J_n} u_{nj}\phi_{nj}(l_n(\boldsymbol{x})) \tag{12}$$

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 1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：

$$
\begin{aligned}
\mathcal{L}u(\boldsymbol{x}) &\approx \mathcal{L}u_M(\boldsymbol{x}) \\
&= \mathcal{L}\sum_{n=1}^{M_p}\psi_n(l_n(\boldsymbol{x}))\sum_{j=1}^{J_n}u_{nj}\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \mathcal{L}\sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\mathcal{L}\psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x})) \\
&= \sum_{n=1}^{M_p}\sum_{j=1}^{J_n}u_{nj}\mathcal{L}\Phi_{nj}(\boldsymbol{x}) \quad \left[\Phi_{nj}(\boldsymbol{x}) \triangleq \psi_n(l_n(\boldsymbol{x}))\phi_{nj}(l_n(\boldsymbol{x}))\right] \qquad (13) \\
&= \begin{bmatrix}\mathcal{L}\Phi_{11}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x})\end{bmatrix}
\begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix}
\end{aligned}
$$

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

$$
\begin{bmatrix}
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(1)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(1)}) \\
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(2)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(2)}) \\
\vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\mathcal{L}\Phi_{11}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{1J_1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_p1}(\boldsymbol{x}_P^{(N_P)}) & \cdots & \mathcal{L}\Phi_{M_pJ_{M_p}}(\boldsymbol{x}_P^{(N_P)})
\end{bmatrix}
\begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix}
\approx
\begin{bmatrix} f(\boldsymbol{x}_P^{(1)}) \\ f(\boldsymbol{x}_P^{(2)}) \\ \vdots \\ f(\boldsymbol{x}_P^{(N_P)}) \end{bmatrix} \qquad (14)
$$

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

$$\begin{bmatrix} \Phi_{11}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(1)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(1)}) \\ \Phi_{11}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(2)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(2)}) \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \Phi_{11}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{1J_1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_p1}(\boldsymbol{x}_B^{(N_B)}) & \cdots & \Phi_{M_pJ_{M_p}}(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \begin{bmatrix} u_{11} \\ \vdots \\ u_{1J_1} \\ \vdots \\ u_{M_p1} \\ \vdots \\ u_{M_pJ_{M_p}} \end{bmatrix} \approx \begin{bmatrix} g(\boldsymbol{x}_B^{(1)}) \\ g(\boldsymbol{x}_B^{(2)}) \\ \vdots \\ g(\boldsymbol{x}_B^{(N_B)}) \end{bmatrix} \tag{15}$$

记：

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

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

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

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 $M = \sum_{n=1}^{M_p} J_n$：

$$Au = f \quad , A = \begin{bmatrix} P \\ B \end{bmatrix} \tag{16}$$

1.1.3 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

$$Loss = \sum_{\boldsymbol{x}_i \in \Omega} \lambda_i \|\mathcal{L}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{x}_i) - \boldsymbol{f}(\boldsymbol{x}_i)\|_2^2 + \sum_{\boldsymbol{x}_j \in \partial\Omega} \lambda_j \|\mathcal{B}\boldsymbol{u}_{\boldsymbol{M}}(\boldsymbol{x}_j) - \boldsymbol{g}(\boldsymbol{x}_j)\|_2^2. \tag{17}$$

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。

$$\begin{cases} \lambda_i = \frac{c_i}{max(A_i, -A_i)} \\ A_i = \lambda_i \times A_i \\ f_i = \lambda_i \times f_i \end{cases} \tag{18}$$

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。

1.1.4 时空随机特征方法

考虑如下时空偏微分方程：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x}, t) = f(\boldsymbol{x}, t) \quad \boldsymbol{x}, t \in \Omega \times [0, T] \\ \mathcal{B}u(\boldsymbol{x}, t) = g(\boldsymbol{x}, t) \quad \boldsymbol{x}, t \in \partial\Omega \times [0, T] \\ \mathcal{I}u(\boldsymbol{x}, t) = h(\boldsymbol{x}) \quad \boldsymbol{x}, t \in \Omega \end{cases} \tag{19}$$

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：

$$\psi_{n_x, n_t}(\boldsymbol{x}, t) = \psi_{n_x}(l_{n_x}(\boldsymbol{x})) \cdot \psi_{n_t}(l_{n_t}(t)) \tag{20}$$

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。

$$\phi_{n_x,n_t,j}(\boldsymbol{x},t)=\sigma(\boldsymbol{k}_{n_x,n_t,j}l_{n_x,n_t}(\boldsymbol{x},t)+b_{n_x,n_t,j}) \tag{21}$$

   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：

$$\phi_{n_x,n_t,j}(\boldsymbol{x},t)=\sigma(\boldsymbol{k}^x_{n_x,n_t,j}l_{n_x}(\boldsymbol{x})+b^x_{n_x,n_t,j})\cdot\sigma(k^t_{n_x,n_t,j}l_{n_t}(t)+b^t_{n_x,n_t,j}) \tag{22}$$

   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：

$$u_M(\boldsymbol{x},t)=\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\psi_{n_x,n_t}(\boldsymbol{x},t)\sum_{j=1}^{J_n}u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \tag{23}$$

   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 基于 ST-RFM 的数据同化

2.1 ST-RFM 的改进

在 RFM 章节，公式 13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进一步的简化：

$$
\begin{aligned}
\mathcal{L}u(\boldsymbol{x},t) &\approx \mathcal{L}u_M(\boldsymbol{x},t) \\
&= \mathcal{L}\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\psi_{n_x,n_t}(\boldsymbol{x},t)\sum_{j=1}^{J_n}u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \\
&= \mathcal{L}\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n}\psi_{n_x,n_t}(\boldsymbol{x},t)u_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \\
&= \mathcal{L}\sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n}\tilde{u}_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \quad \left[\tilde{u}_{n_x,n_t,j}\triangleq\psi_{n_x,n_t}(\boldsymbol{x},t)u_{n_x,n_t,j}\right] \\
&= \sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n}\mathcal{L}\tilde{u}_{n_x,n_t,j}\phi_{n_x,n_t,j}(\boldsymbol{x},t) \\
&= \sum_{n_x=1}^{N_x}\sum_{n_t=1}^{N_t}\sum_{j=1}^{J_n}\tilde{u}_{n_x,n_t,j}\mathcal{L}\phi_{n_x,n_t,j}(\boldsymbol{x},t)
\end{aligned}
\tag{24}
$$

其中，$\tilde{u}_{n_x,n_t,j}\triangleq\psi_{n_x,n_t}(\boldsymbol{x},t)u_{n_x,n_t,j}$ 是待训练的参数，这一步是关键，经过这一步的转换，微分算子离散化的结果就不再包含单位分解函数了。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。

2.1.1 数值算例验证

我们使用一维 Helmholtz 方程验证 RFM，方程如下：

$$
\begin{cases}
\frac{d^2u(x)}{dx^2}-\lambda u(x)=f(x) \quad x\in[0,8] \\
u(0)=c_1, \quad u(8)=c_2
\end{cases}
\tag{25}
$$

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。

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；

计算结果见表格 1。见图，由图和表格 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。

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

我们使用一维扩散方程验证 ST-RFM，方程如下：

$$\begin{cases} \frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} = f(x,t) \quad (x,t) \in [0,5] \times [0,1] \\ u(0,t) = g_1(t) \quad t \in [0,1] \\ u(5,t) = g_2(t) \quad t \in [0,1] \\ u(x,0) = h(x) \quad x \in [0,5] \end{cases} \tag{26}$$

我们取 $\nu = 0.01$，真解 $u(x,t) = [2\cos(\pi x + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi x - \frac{3\pi}{5})][2\cos(\pi t + \frac{\pi}{5}) + \frac{3}{2}\cos(2\pi t - \frac{3\pi}{5})]$，则 $f(x,t), g_x(t), g_2(t), h(x)$ 可相应的计算出来。

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 $N_x = 30$；
- 每个子区域的时间维度的配点数量为 $N_t = 30$；
- 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 $Q_n = N_x \times N_t = 900$，求解结果见表格 2。见图，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。

考虑如下数据同化问题：

$$\begin{cases} \mathcal{L}u(\boldsymbol{x},t) = f(\boldsymbol{x},t) \quad \boldsymbol{x},t \in \Omega \times [0,T] \\ \mathcal{B}u(\boldsymbol{x},t) = b(\boldsymbol{x},t) \quad \boldsymbol{x},t \in \Omega \times [0,T] \\ \mathcal{I}u(\boldsymbol{x},0) = i(\boldsymbol{x}) \quad \boldsymbol{x} \in \Omega \\ \mathcal{H}u(\boldsymbol{x},t) = h(\boldsymbol{x},t) \quad \boldsymbol{x},t \in \Omega \times [0,T] \end{cases} \tag{27}$$

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，本文基于随机特征方法，解决这一类数据同化问题。

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

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.