一种用于确定非线性多项式函数阿多米安多项式的新算法

Mithun Bairagi*Mangal Chandi High School, Khosalpur, 722206, Patrasayer, Bankura, West Bengal, India

本文提出了一种新算法，可用于确定希尔伯特空间中标量值非线性多项式泛函的阿多米安多项式。该算法无需参数化、展开、重新分组、求导等复杂运算，即可计算阿多米安多项式。该算法仅涉及一些矩阵运算。由于数学运算简单，与文献中此前报道的其他算法相比，新算法速度更快、效率更高。我们还用 MATHEMATICA 代码实现了该算法，并在一维情形下将新算法的计算速度和效率与其他一些算法进行了比较。

Key关键词：阿多米安分解法；阿多米安多项式；非线性算子；矩阵；常微分方程 E; Seri解。

arXiv:2305.04867v1 [cs.CE] 11 Apr 2023

I. 引言

阿多米安分解法（ADM）[1–5] 因其能够求解非线性泛函方程，在科学与工程的不同领域备受关注。实际上，许多非线性问题无法得到精确解，在大多数情况下，我们必须采用数值或解析近似技术求得近似解。阿多米安分解法是一种可靠的方法，可用于求解多种非线性系统，包括常微分方程、偏微分方程、积分微分方程、代数方程、微分代数方程、非整数阶微分方程、积分方程等 [6–12]）。该方法能够以级数形式给出精确解的解析近似，且级数收敛极快[13–15]。阿多米安分解法与拉普拉斯变换相结合，形成了一种称为拉普拉斯阿多米安分解法（LADM）的强大方法。许多文献也采用 LADM 求解分数阶非线性微分方程的数值解，相关示例见[16–20]。

接下来[6, 7, 21], 让我们回顾阿多米安分解法的基本思想。考虑如下形式的非线性常微分方程 p，其中自变量 x （实标量）和因变量 u 满足一般形式 [6, 7]

$$\mathcal{F}u = g(x), \tag{1}$$

其中 $\mathcal{F}$ 是非线性算子，它将希尔伯特空间 H 映射到 H。在阿多米安分解法中，假设 $\mathcal{F}$ 可分解为

$$Lu + Ru + Nu = g(x), \tag{2}$$

其中 L 是最高阶线性微分算子 $L[.] = \frac{d^p}{dx^p}[.]$，假设其可逆； R 是线性微分算子，包含阶数低于 L的线性导数， N 是非线性算子，包含所有其他非线性项； $g(x) \in H$ 是给定的解析函数。这里需要注意，算子 L的选取通常并不唯一 [22–24]。例如，在 [23], A. Wazwaz 将线性微分算子 $L[.]$ 选为$L[.] = x^{-2}\frac{d}{dx}\left(x^2\frac{d}{dx}\right)$，用于 Lane-Emden 方程。还值得注意的是， u 是实变量 x 在式(2)中的标量函数。对于微分方程组， u 将是一个向量值函数。不过，本文的研究仅限于单个常微分方程，其中 u 是一个标量值函数。分解法的关键步骤是假设存在如下定义的级数解：

$$u = \sum_{i=0}^{\infty} u_i, \tag{3}$$

然后，与函数方程（2）相对应的阿多米安分解法方案迅速收敛至 $u \in H$，它是函数方程 [6, 25] 的唯一解。方程（3）将非线性项 Nu 分解为一个无穷

*电子邮箱：bairagirasulpur@gmail.com

级数

$$Nu = \sum_{i=0}^{\infty} A_i, \tag{4}$$

其中 A_i 是所谓的阿多米安多项式，它们取决于解分量 u_0，u_1、.....、u_i。对于给定的非线性泛函 $Nu = F(u)$（$F(u)$ 假设是希尔伯特空间 u 中变量 H的解析函数），阿多米安多项式由 G. Adomian 提出的以下定义公式确定 [1–3, 26]:

$$A_M = \frac{1}{M!}\frac{d^M}{d\lambda^M} F\left(\sum_{k=0}^{\infty} u_k \lambda^k\right)\Bigg|_{\lambda=0}, \quad M = 0, 1, 2, \ldots, \tag{5}$$

其中解析参数 λ 只是一个分组参数。阿多米安多项式 A_M的一个重要性质是，按其构造，它只取决于解分量（u_0, u_1.....，u_M），而不取决于高阶解分量 u_k，其中 $k > M$ [27, 28]。因此，$k > M$ 的高阶项不会参与式（5）中的求和。

阿多米安分解法的主要步骤是确定非线性项 Nu的阿多米安多项式。使用定义公式（5）计算高阶阿多米安项较为困难，因为高阶导数的计算十分复杂。后来，许多作者提出了若干便捷算法，用于快速生成单变量和多变量阿多米安多项式。Adomian and Rach [29] 提出了一种递推规则，提供了确定阿多米安多项式的系统化计算过程。此后，Rach 在其论文 [30] 中为 Adomian and Rach 的算法建立了简单的对称规则（称为 Rach规则），借此可以快速确定高阶阿多米安多项式。采用 Wazwaz 在 [27], 中提出的算法时，需要从展开中收集各项，这在高阶情况下会耗费大量计算时间。采用 [31],中的算法时，则需要在阿多米安多项式之间的递推关系中进行代入后求导。近期，[32], 中的作者修改了公式（5），以确定非线性多项式泛函的阿多米安多项式。在 [21, 33], 中，Duan 提出了更高效、快速的递推算法，用于快速生成单变量（在我们的研究中即一维情形）和多变量的阿多米安多项式。Duan推论1算法[21]（称为指标递推算法）和 Duan推论3算法 [33] 在一维情况下确定约化多项式时无需使用微分算子。我们只需进行加法和乘法运算，因此这些算法更加快速、高效。

本文提出了一种新算法，通过矩阵形式而非递推过程，快速计算希尔伯特空间中指标为正整数的标量值非线性多项式泛函的阿多米安多项式 H。所提出的算法无需参数化、展开、重新分组和求导等复杂的数学运算。利用该算法，只需进行少量矩阵运算即可确定高阶阿多米安多项式，因此比文献中现有的其他算法更快、更高效。我们还将新算法推广到二维情形，此时解 u 取决于两个状态变量，例如 t、x。

本文结构如下：在第 II 节中，我们介绍用于确定非线性多项式泛函的阿多米安多项式的算法。在第 III节中，我们将算法应用于多项式函数，并将计算时间与文献中此前报道的一些常用算法进行比较。在第 IV节中，我们讨论结果，并对本研究作出总结。附录 3 A、B 中分别列出了用于一维情形和二维情形的新算法的 MATHEMATICA 代码，见列表 3 和列表 6此外，我们还列出了其他一些算法的 MATHEMATICA 代码，包括 Duan推论1算法 [21] 和 Duan推论3算法 [33, 34]，其一维情形的代码见附录 4、5 A中的列表。

II. 我们所提出算法的描述

本节介绍了一种计算阿多米安多项式的新算法。该算法仅适用于二维情形下希尔伯特空间H 中的标量值非线性多项式泛函（指标为正整数）。为提高该算法的计算效率，所有数学运算均以矩阵形式进行。

现在我们考虑一个非线性多项式泛函 F，它在 H中依赖于两个不同的函数 u 和 v。函数 u 和 v 可以展开为以下二重级数

$$u = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} u_{ij} \quad \text{and} \quad v = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} v_{ij}\,. \tag{6}$$

为了说明我们的算法，我们将非线性项 F 取为简单形式

$$F = uv. \tag{7}$$

而这个非线性函数可以展开为一个级数

$$F = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} A_{ij}, \tag{8}$$

其中 A_{ij}称为分量 u_{ij}、$v_{ij}(i=0$、1、.....、$j=0$、1、.....）的 Adomian多项式。现在，我们将算法分为六个主要步骤（标记为步骤1至步骤6），并使用非线性多项式函数（7）来说明各个步骤。

步骤1 （将函数 u 和 v 表示为矩阵形式）：在此步骤中，函数 u 和 v 表示为矩阵形式。为了在计算机上进行计算，我们将无穷级数（6）截断至有限项$i=m$，$j=n$。尽可能增大 m、n的值，可以提高结果的准确度。方程（ u、 v）中的函数可表示为（6）中的（$m+1$） $\times$ （$n+1$）矩阵

$$U = \begin{pmatrix} u_{00} & u_{01} & \dots & u_{0l} & \dots & u_{0n} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ u_{k0} & u_{k1} & \dots & u_{kl} & \dots & u_{kn} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ u_{m0} & u_{m1} & \dots & u_{ml} & \dots & u_{mn} \end{pmatrix} \text{ and } V = \begin{pmatrix} v_{00} & v_{01} & \dots & v_{0l} & \dots & v_{0n} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ v_{k0} & v_{k1} & \dots & v_{kl} & \dots & v_{kn} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ v_{m0} & v_{m1} & \dots & v_{ml} & \dots & v_{mn} \end{pmatrix}. \tag{9}$$

步骤2（从矩阵 U 和 V中提取子矩阵）：方程（9）中任意矩阵元素（设矩阵元素 u_{kl}、v_{kl}位于第 $k+1$行、第 $l+1$列）所对应的阿多米安多项式，取决于行号（r）和列号（c）分别小于或等于 $k+1$和 $l+1$的其他矩阵元素，但不取决于位于 $r>k+1$和 $c>l+1$处的矩阵元素。为了计算 U 和 V中的元素 u_{kl} 和 v_{kl} 对应的阿多米安多项式，我们从方程（9）中的矩阵 U和 V提取由行 $r\le k+1$和列 $c\le l+1$中的元素构成的子矩阵。这些子矩阵如下：

$$U[0,1,\dots,k;0,1,\dots,l] = \begin{pmatrix} u_{00} & u_{01} & \dots & u_{0l} \\ \vdots & \vdots & \dots & \vdots \\ u_{k0} & u_{k1} & \dots & u_{kl} \end{pmatrix} \text{ and } V[0,1,\dots,k;0,1,\dots,l] = \begin{pmatrix} v_{00} & v_{01} & \dots & v_{0l} \\ \vdots & \vdots & \dots & \vdots \\ v_{k0} & v_{k1} & \dots & v_{kl} \end{pmatrix}. \tag{10}$$

步骤3 （翻转子矩阵）：在此步骤中，将式（10）中任一子矩阵的所有矩阵元素先水平翻转，再垂直翻转，反之亦可。此处，我们对子矩阵 $V[0,1,\dots,k;0,1,\dots,l]$执行翻转操作。沿水平轴的翻转操作可表示如下

$$\xrightarrow{\text{flipping horizontally}} \quad \begin{pmatrix} v_{00} & v_{01} & \dots & v_{0l} \\ \vdots & \vdots & \dots & \vdots \\ v_{k0} & v_{k1} & \dots & v_{kl} \end{pmatrix} \longrightarrow \begin{pmatrix} v_{0l} & v_{0l-1} & \dots & v_{00} \\ \vdots & \vdots & \dots & \vdots \\ v_{kl} & v_{kl-1} & \dots & v_{k0} \end{pmatrix} = V[0,1,\dots,k;l,l-1,\dots,0]. \tag{11}$$

随后，对上述已翻转的子矩阵沿垂直轴执行翻转操作，可表示如下

$$\text{flipping vertically} \downarrow \quad \begin{pmatrix} v_{0l} & v_{0l-1} & \dots & v_{00} \\ \vdots & \vdots & \dots & \vdots \\ v_{kl} & v_{kl-1} & \dots & v_{k0} \end{pmatrix} \longrightarrow \begin{pmatrix} v_{kl} & v_{kl-1} & \dots & v_{k0} \\ \vdots & \vdots & \dots & \vdots \\ v_{0l} & v_{0l-1} & \dots & v_{00} \end{pmatrix} = V[k,k-1,\dots,0;l,l-1,\dots,0]. \tag{12}$$

步骤 4（矩阵逐元素乘法）：在逐元素乘法（也称为哈达玛积）中，两个矩阵中的每个元素 i、j 都相乘。我们对两个矩阵 $U[0,1,\dots,k;$ 0、1、...、$l]$ 和 $V[k,k-1,\dots,0;$ l、$l-1$、...、$0]$进行逐元素乘法，结果为

$$U[0,1,\dots,k;0,1,\dots,l]\circ V[k,k-1,\dots,0;l,l-1,\dots,0] = W[0,1,\dots,k;0,1,\dots,l] \tag{13}$$

用矩阵表示法，上述方程可以表示为

$$\begin{pmatrix} u_{00} & u_{01} & \dots & u_{0l} \\ \vdots & \vdots & \dots & \vdots \\ u_{k0} & u_{k1} & \dots & u_{kl} \end{pmatrix} \circ \begin{pmatrix} v_{kl} & v_{kl-1} & \dots & v_{k0} \\ \vdots & \vdots & \dots & \vdots \\ v_{0l} & v_{0l-1} & \dots & v_{00} \end{pmatrix} = \begin{pmatrix} u_{00}v_{kl} & u_{01}v_{kl-1} & \dots & u_{0l}v_{k0} \\ \vdots & \vdots & \dots & \vdots \\ u_{k0}v_{0l} & u_{k1}v_{0l-1} & \dots & u_{kl}v_{00} \end{pmatrix}. \tag{14}$$

这里的符号 $\circ$ 表示两个矩阵之间的逐元素乘法。

步骤5（矩阵元素求和）：在此步骤中，我们对矩阵$W[0,1,\dots,k$的所有元素 0、1、.....、 $l]$ 求和，结果为

$$A_{kl} = \sum_{i=0}^{k}\sum_{j=0}^{l} W_{ij} = u_{00}v_{kl} + u_{01}v_{kl-1} + \dots + u_{kl}v_{00}. \tag{15}$$

这里 A_{kl} 是两个矩阵元素的阿多米安多项式 u_{kl}， v_{kl}。在阿多米安多项式 A_{kl}中，值得注意的是， u的各项分量下标中的第一个指标之和 v 在 A_{kl} 的每一项中都相同。同样， u的各项分量的第二个指标之和 v 在 A_{kl} 的每一项中也相同（这里，第一个指标之和为 k ，第二个指标之和为 l），这符合 [27]中给出的阿多米安多项式的重要性质。

步骤6（构造Adomian矩阵）：重复步骤1至步骤5，确定式（9）中与各矩阵元素对应的阿多米安多项式。所有计算得到的阿多米安多项式都存储在一个矩阵中，可表示为

$$A = \begin{pmatrix} A_{00} & A_{01} & \dots & A_{0l} & \dots & A_{0n} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ A_{k0} & A_{k1} & \dots & A_{kl} & \dots & A_{kn} \\ \vdots & \vdots & \dots & \vdots & \dots & \vdots \\ A_{m0} & A_{m1} & \dots & A_{ml} & \dots & A_{mn} \end{pmatrix}. \tag{16}$$

We c所有矩阵 A 在（16）中，作为给定多项式非线性项（7） .

我们给出了步骤1至步骤6中所述算法的伪代码，见清单 1 ，这些算法用于计算公式(7)的阿多米安矩阵。值得注意的是，步骤1至步骤6中的少量简单矩阵运算如何生成公式(7)的阿多米安多项式。从步骤1至步骤6可以清楚地看出，计算公式$(m+1)(n+1) - (m+n+2)$所需的矩阵运算次数（$2(m+1)(n+1)-(m+n+2)$次翻转，$(m+1)(n+1)$次逐元素矩阵乘法，以及$(m+1)(n+1)$次矩阵求和）仅为这些运算，用以计算公式(7)的阿多米安矩阵，其中 $i = m$, $j = n$ 见公式(6)。这种数学运算上的简洁性提高了该算法的计算效率。

清单1 Adomian矩阵的计算 A 方程 (7) 的伪代码。

1 输入：方程（7）的函数 u和 v2 输出：Adomian矩阵 A3 函数 AdomianMatrix (u, v)4 将 u 表示为矩阵形式 U： $U \leftarrow$ Matrix ($\sum_{i=0}^{m}\sum_{j=0}^{n} u_{ij}$)5 将 v 表示为矩阵形式 V： $V \leftarrow$ Matrix ($\sum_{i=0}^{m}\sum_{j=0}^{n} v_{ij}$)6 对于 $k \leftarrow m$ 到 $k \geq 0$ ，执行7 对于 $l \leftarrow n$ 到 $l \geq 0$ ，执行 8 $U[0,1,\dots,k;\ 0,1,\dots,l] \leftarrow$， U 中对应元素 U_{kl}9 $V[0,1,\dots,k$ 的子矩阵；$0,1,\dots,l] \leftarrow$， V 中对应元素 V_{kl}10 $V[k,k-1,\dots,0$ 的子矩阵；$l,l-1,\dots,0] \leftarrow V[0,1,\dots,k;\ 0,1,\dots,l]$ 水平翻转，然后垂直翻转11 按元素相乘： $W[0,1,\dots,k;\ 0,1,\dots,l] \leftarrow U[0,1,\dots,k;\ 0,1,\dots,l] \circ V[k,k-1,\dots,0;\ l,l-1,\dots,0]$12 $A_{kl} \leftarrow \sum_{i=0}^{k}\sum_{j=0}^{l} W_{ij}$13 结束循环14 结束循环15 返回 A16 结束函数

列表 2 Adomian矩阵的计算 A 方程 (17) 的伪代码。

1 输入：方程 (17) 的函数 $u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P-2)}, u^{(P-1)}, u^{(P)}$ 2 输出：Adomian 矩阵 A3 函数 AdomianMatrix2 ($u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P-2)}, u^{(P-1)}, u^{(P)}$)4 将 $u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P-2)}, u^{(P-1)}, u^{(P)}$ 表示为矩阵形式： $U^{(P)} \leftarrow$ Matrix ($\sum_{i=0}^{m} \sum_{j=0}^{n} u_{ij}^{(P)}$) 5 $A \leftarrow U^{(P)}$6 对于 $k \leftarrow P$ 到 $k \geq 2$ 执行7 A $\leftarrow$ Adomian矩阵 ($U^{(k-1)}$, A)8 结束循环9 返回 A10 结束函数

A. F 的一般形式

Le现在考虑以下一般形式的非线性多项式泛函 F

$$F = u^{(1)} u^{(2)} u^{(3)} \ldots u^{(P-2)} u^{(P-1)} u^{(P)}, \tag{17}$$

其中， F取决于 P个二维函数 $u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P)}$。对于 $P = 2$和 $u^{(1)} = u, u^{(2)} = v$， 方程 (17) 可简化为方程 (7)。步骤1至步骤6中介绍的算法也可按以下方式用于方程 (17)。设 $U(1)\,U(2)\,U(3) \ldots, U(P)$ 分别为二维函数 $u^{(1)}$, $u^{(2)}, u^{(3)}, \ldots, u^{(P)}$ 的矩阵形式。为了确定方程 (17) 的 Adomian矩阵，我们首先使用步骤1至步骤6中介绍的算法，确定前两个矩阵 $U(1)\,U(2)$ 或最后两个矩阵 $U(P-1)\,U(P)$ 的 Adomian矩阵。设 $A^{(P-1)(P)}$ 为最后两个矩阵 $U(P-1)$ 和 $U(P)$ 的 Adomian矩阵。接下来，我们确定两个矩阵 $A^{(P-1)(P)}$ 与前一个矩阵 $U(P-2)$ 的 Adomian矩阵。此过程持续进行，直到最前面的矩阵 $U^{(1)}$。完成此过程后，最终即可得到方程 (17) 中 F 的 Adomian矩阵。我们在列表 2 中以伪代码形式展示这一过程，该过程用于确定方程 (17) 的 Adomian矩阵。

现在，我们考虑更一般且更复杂形式的非线性多项式泛函 F （和的幂次形式）

$$F = \left(u^{(1)} + u^{(2)} + u^{(3)} + \ldots + u^{(P-2)} + u^{(P-1)} + u^{(P)} \right)^{\mathcal{N}} \tag{18}$$

其中幂指数 $\mathcal{N}$ 是一个正整数。在这种情况下，首先将式（18）展开为乘积项之和。然后，我们可以利用上述算法，轻松确定式（17）中展开式各项的 Adomian矩阵。最后，将各项对应的 Adomian矩阵相加，即可得到式（18）的 Adomian矩阵。

在一维情形下，级数（6）只有一个指标（例如 i）。因此，所有矩阵都是一维的；在这种情况下，步骤3中只需执行水平翻转操作。除此之外，从步骤1到步骤6所述的所有算法在一维情形下均相同。下文将我们提出的新算法称为 Adomian矩阵算法。

III. 软件

E 实现及与其他算法的比较

我们已将第 II 节中描述的算法（称为 Adomian矩阵算法）编写为附录：3 （一维情形）、6 （二维情形）中的 MATHEMATICA 代码： A、B 分别列于其中。这些 MATHE- MATICA 程序可以确定以下多项式泛函的一维（使用清单 3）和二维（使用清单 6）Adomian多项式

$$F = u^{\mathcal{N}}, \tag{19}$$

其中，幂指数 $\mathcal{N}$ 是表示非线性阶数的正整数。为了确定方程（19）的Adomian多项式，我们必须在 MATHE- MATICA 函数的函数参数中输入幂指数 $\mathcal{N}$ 和 Adomian矩阵的阶数（这些函数参数的详细说明见附录），这些函数随后会在 MATHEMATICA笔记本的输出单元格中打印出 Adomian多项式。

此外，附录中还列出了适用于一维情形的其他算法的 MATHEMATICA 代码，例如 Duan推论1算法 [21], Duan推论 3算法 [33] ， 见附录中的代码清单4、5 ： 附录B。MATHEMATICA

代码清单4 和代码清单5 中的程序分别取自 [21] 附录 A.1 和 [34] 附录 A。为提高程序的运行速度，我们对程序进行了修改（见[21], [34]），使其仅适用于多项式泛函（19），并使用阶乘公式 $\frac{d^i F}{du^i} = \frac{\mathcal{N}!}{(\mathcal{N}-i)!} u^{\mathcal{N}-i}$计算（19）的导数。

我们使用清单中给出的 MATHEMATICA 程序，将 Adomian矩阵算法与其他算法进行了比较：3、4、5、6 ，并采用多项式泛函（19）。在表 I中，我们展示了一维情形下 Adomian矩阵算法（第 3 列）与另外两种算法（第 4、5 列）的计算速度（以秒计）比较，所用程序见附录 3、4、5 中的清单： A。我们使用 MATHEMATICA 9.0，在搭载 Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz 处理器和 8 GB RAM 的笔记本电脑上，通过 MATHEMATICA 命令`Timing[]`并抑制输出（即将结果保留在内存中）来测量计算时间。表 I 显示，Adomian矩阵算法比另外两种算法更快、更高效：Duan推论1算法 [21] 和 Duan推论3算法 [33]。例如，我们发现，在计算前 50 个阿多米安多项式时，与另外两种算法相比，Adomian矩阵算法在 10^4 时快了近 $\mathcal{N} = 3$ 倍，在 10^3 时快了近 $\mathcal{N} = 10$ 倍。此外，在计算前 100 个阿多米安多项式时，Adomian矩阵算法耗时 $\sim 10^{-2}$ 秒；但值得注意的是，另外两种算法在经过 600 秒后仍无法得出结果。

我们还使用列表6中的 MATHEMATICA 代码，检验了 Adomian矩阵算法在二维情形下的计算效率。例如，式（19）中阶数为40×40 的阿多米安多项式，可在 2.6 秒内生成（当 $\mathcal{N} = 3$ 时）；当 $\mathcal{N} = 10$时，则可在 19.5 秒内生成。

表 I：Adomian矩阵算法与其他一些算法在不同 $\mathcal{N}$ 值下计算时间（单位：秒）的比较，适用于（19）中的情形，以及一维情况下阿多米安多项式的不同数量（n）。在某些表格单元格中， × 符号表示对应列中的算法耗时近 600 秒后仍无法计算阿多米安多项式。

非线性项 ($\mathcal{N}$)	阿多米安多项式的数量 (n)	Adomian矩阵算法	Duan推论1算法 [21]	Duan推论3算法 [33]
3	10	0.00047	0.0020	0.0025
	30	0.002	0.83	0.76
	50	0.0047	62	46
	100	0.017	×	×
5	10	0.00078	0.0026	0.0025
	30	0.0039	0.87	0.68
	50	0.0092	62.5	46.4
	100	0.037	×	×
10	10	0.0033	0.0037	0.0029
	30	0.012	0.96	0.65
	50	0.026	62.7	46.7
	100	0.095	×	×

IV. 结论

我们提出了一种新算法（称为 Adomian矩阵算法），用于确定希尔伯特空间中标量值非线性多项式泛函（其指标为正整数）的 Adomian多项式 H。Adomian矩阵算法中的计算不需要参数化、展开、重新分组、求导等复杂的数学运算。从第 II 节中的步骤1到步骤6II 可以清楚地看出，Adomian多项式完全通过一些简单的矩阵运算来确定。由于数学运算简单，该算法在快速生成 Adomian多项式方面更加高效。我们基于 Adomian矩阵算法设计了两个 MATHEMATICA 程序（清单3 中的一维情形和清单6中的二维情形），并将其在一维情形下的计算效率与另外两种流行且功能强大的算法进行了比较，即 Duan推论1算法 [21] 和 Duan推论3算法[33]。我们观察到，Adomian矩阵算法的计算效率优于另外两种算法。例如，在非线性项指标为 $\mathcal{N} = 3$ 的情况下，计算方程（19）中的前 50 个 Adomian多项式时，Adomian矩阵算法的速度几乎是另外两种算法的 10^4 倍。对于 $\mathcal{N} = 10$，使用这种新算法，我们仅需 10^{-2} s 就能够求出前 100 个 Adomian多项式；而对于 $\mathcal{N} = 3$ 和 $n = 100$，另外两种算法在经过 600 s 后仍无法得出任何结果。因此，我们可以得出结论：Adomian矩阵算法可用于确定大量非线性多项式泛函的 Adomian 多项式，从而使所得解更加精确。

附录A：一维情形的 Mathematica 程序

以下三个 MATHEMATICA 程序可以确定非线性函数（19）的一维 Adomian 多项式。函数参数 N 和 n 分别表示方程（ $\mathcal{N}$19）中的非线性幂指数和前 n 个 Adomian 多项式的数量。

清单 3：基于 Adomian矩阵算法的程序。

```
AdomMatAlgo1D[N_,n_]:=模块[{h,j,k},u=.;mat=表格[下标[u,h],
  {h,0,n-1}];temmat=表格[下标[u,h],{h,0,n-1}];对于[j
  =1,j<=N-1,j++,对于[k=n,k>=1,k--,mat[[k]]=总计[
temmat[[;;k]]*反转[mat[[;;k]]]];];mat]
```

清单4 基于Duan推论1算法的程序 [21]。

```
Duan指标算法Adomian[N_,n_]:=模块[{Apoly,Zpoly,dirClt},Subscript[Apoly,0]=Subscript[u,0]
^N;Zpoly=表格[0,{i,1,n-1},{j,1,i}];执行[Zpoly[[suInd,1]]=Subscript[u,suInd],{suInd,
1,n-1}];对于[i=2,i<=n-1,i++,对于[j=2,j<=i,j++,Zpoly[[i,j]]=展开[Subscript[u,
  1]*Zpoly[[i-1,j-1]]]];如果[Head[Zpoly[[i,j]]]===为Plus,Zpoly[[i,j]]=映射[#/Exponent[#,
Subscript[u,1]]&,Zpoly[[i,j]]],Zpoly[[i,j]]=映射[#/Exponent[#,Subscript[u,1]]&,Zpoly[[i,j]],
  {0}]]];对于[j=2,j<=Floor[i/2],j++,Zpoly[[i,j]]=Zpoly[[i,j]]+(Zpoly[[i-j,j]]/.
Subscript[u,sub_]->Subscript[u,sub+1])]];dirClt=表格[阶乘[N]/阶乘[N-j]*(Subscript[u,0]
^(N-j)),{j,1,n-1}];执行[Subscript[Apoly,suInd]=取[dirClt,suInd].Zpoly[[suInd]],{s
uInd,1,n-1}];表格[Subscript[Apoly,suInd],{suInd,0,n-1}]]
```

清单5 基于Duan推论3算法的程序 [33, 34]。

```
DuanCoro3AlgoAdm[N_,n_]:=模块[{cPoly,i,k,j,derClt},
表格[多项式列[i,k],{i,1,n-1},{k,1,i}];
derClt=表格[阶乘[N]/阶乘[N-k]*(下标[u,0]^(N-k)),
  {k,1,n-1}];
阿多米安多项式[0]=下标[u,0]^N;
对于[i=1,i<=n-1,i++,
  cPoly[i,1]=下标[u,i];
  对于[k=2,k<=i,k++,
    多项式系数[i,k]=展开[1/i*求和[(j+1)*下标[u,j+1]*多项式系数[i-1-j,k-1],
      {j,0,i-k}]]];
阿多米安多项式[i]=取[derClt,i]。计算[cPoly[i,k],{k,1,i}]];
计算[Apoly[i],{i,0,n-1}]]
```

附录 B：二维情形的 Mathematica 程序

以下 MATHEMATICA 程序可以求出非线性函数（19）的二维阿多米安多项式。函数参数 N、m 和 n 分别表示式（ $\mathcal{N}$19）中的非线性幂指数、Adomian矩阵的行数和列数。

基于 Adomian矩阵算法的清单6程序。

```
AdomMatAlgo2D[N_, m_, n_] := Module[{g, h, j, k, l}, u =.; mat = Table[Subscript[u, g, h], {g, 0, m - 1}, {h, 0, n - 1}]; temmat = Table[Subscript[u, g, h], {g, 0, m - 1}, {h, 0, n - 1}]; For[j = 1, j <= N - 1, j++, For[k = m, k >= 1, k--, For[l = n, l >= 1, l--, mat[[k, l]] = Total[temmat[[;; k, ;; l]]] * Reverse[Reverse[mat[[;; k, ;; l]], 1], 2]]]]; mat]
```

参考文献

[1] G. Adomian, 《随机系统》, Academic Press, 纽约, (1983)。[2] G. Adomian, 《非线性随机算子方程》, Academic Press, 奥兰多, (1986)。[3] G. Adomian, 《非线性随机系统理论及其在物理学中的应用》, Kluwer Academic, 多德雷赫特, (1989)。[4] R. Rach、G. Adomian、R.E. Meyers, 《一种改进的分解法》, Comput. Math. Appl. 23 (1992) 17–23。[5] G. Adomian, 《求解物理学前沿问题: 分解法》, Kluwer Academic, 多德雷赫特, (1994)。[6] T. Mavoungou、Y. Cherruault, 《阿多米安方法的收敛性及其在非线性偏微分方程中的应用》, Kybernetes, 21 (1992) 13-25。[7] N. Ngarhasta, *et al.*, 《阿多米安方法应用于扩散模型的新数值研究》, Kybernetes, 31 (2002) 61–75。[8] A.M. Wazwaz, 《利用分解法求得非线性扩散方程的精确解》, Appl. Math. Comput. 123 (2001) 109–122。[9] A.M. Wazwaz, 《偏微分方程与孤立波理论》, 高等教育出版社, 北京, (2009)。[10] E. Yee, 《分解法在求解反应–对流–扩散方程中的应用》, Appl. Math. Comput. 56 (1993) 1–27。[11] J. Biazar, 《利用阿多米安分解法求解积分微分方程组》, Appl. Math. Comput. 168 (2005) 1232–1238。[12] J.S. Duan、J.Y. An、M.Y. Xu, 《利用阿多米安分解法求解分数阶微分方程组》, Appl. Math. J. Chin. Univ. B 22 (2007) 7–12。[13] Y. Cherruault, 《阿多米安方法的收敛性》, Kybernetes 18 (1989) 31–38。[14] K. Abbaoui、Y. Cherruault, 《阿多米安方法应用于微分方程时的收敛性》, Comput. Math. Appl. 28 (1994) 103–109。[15] K. Abbaoui、Y. Cherruault, 《证明分解法收敛性的新思路》, Comput. Math. Appl. 29 (1995) 103–108。[16] F. Haq 等, 《利用拉普拉斯阿多米安分解法对媒介传播疾病分数阶流行病模型进行数值求解》, Punjab University Journal of Mathematics 49 (2017) 13。[17] A. Ali、Humaira、Laila、K. Shah, 《利用拉普拉斯阿多米安分解法求解广义 Fisher 方程的解析解》, J. Pure Appl. Math 2 (2018) 01。[18] K. Shah、S. Bushnaq, 《利用拉普拉斯阿多米安分解法对分数阶地方病模型进行数值处理》, Journal of Science and Arts 39 (2017) 257。[19] F. Haq、K. Shah、G. ur Rahman、M. Shahzad, 《利用拉普拉斯阿多米安分解法对分数阶吸烟模型进行数值求解》, Alexandria Engineering Journal, 57 (2018) 1061。[20] A. Ali、K. Shah、R. Ali Khan, 《分数阶 Whitham-Broer-Kaup 方程行波解的数值处理》, Alexandria Engineering Journal 57 (2018) 1991。[21] J.S. Duan, 《多变量阿多米安多项式的高效算法》, Appl. Math. Comput. 217 (2010) 2456–2467。[22] G. Adomian、R. Rach、N.T. Shawagfeh, 《关于 Lane–Emden 方程的解析解》, Found. Phys. Lett. 8 (1995) 161–181。[23] A.M. Wazwaz, 《一种求解二阶常微分方程奇异初值问题的新方法》, Appl. Math. Comput. 128 (2002) 45–57。[24] A.M. Wazwaz, 《用于解析处理微分方程的修正分解法》, Appl. Math. Comput. 173 (2006) 165–176。[25] L. Bougoffa, J. Appl. Math. Comput. 43 (2013) 31–54。[26] J.S. Duan, 《非经典阿多米安多项式的新递推算法》, Computers and Mathematics with Applications, 62 (2011) 2961。

[27] A.M. Wazwaz, 《一种计算非线性算子阿多米安多项式的新算法》, Appl. Math. Comput. 111 (2000) 53–69。[28] M. Azreg-A¨ınou, 《一种用于计算阿多米安多项式的新型改进算法》, Comput. Model. Eng. Sci. 42 (2009) 1–18。[29] G. Adomian、R. Rach, 《非线性随机算子的求逆》, J. Math. Anal. Appl. 91 (1983) 39–46。[30] R. Rach, 《阿多米安多项式的一种便捷计算形式》, J. Math. Anal. Appl. 102 (1984) 415–419。[31] J. Biazar、S. M. Shafiof, 《计算阿多米安多项式的一种简单算法》, Int. J. Contemp. Math. Sciences, 2 (2007) 975-982。[32] E. U. Agom、F. O. Ogunfiditimi, 《具有整数指数的非线性泛函的修正阿多米安多项式》, IOSR-JM, 11 (2015) 40-45。[33] J.S. Duan, 《阿多米安多项式的便捷解析递推算法》, Appl. Math. Comput. 217 (2011) 6337–6348。[34] J.S. Duan、R. Rach、A.M. Wazwaz, 《利用多阶段阿多米安分解法求解非线性边值问题正解的可靠算法》, Open Engineering, 5 (2014) 59–74。