Operations Research and Fuzziology 运筹与模糊学, 2023, 13(5), 5571-5580
Published Online October 2023 in Hans. https://www.hanspub.org/journal/orf
https://doi.org/10.12677/orf.2023.135556

Hans汉斯

卷积型Volterra积分微分方程的一种快速算法

李海洋*，胡怀青，刘婧雅

贵州大学数学与统计学院，贵州 贵阳

收稿日期：2023年9月4日；录用日期：2023年10月17日；发布日期：2023年10月25日

摘　要

卷积型Volterra积分微分方程是一类重要问题，广泛应用于生物学、经济学，本文研究了一种快速算法求解卷积型Volterra积分微分方程。该方法采用多步配置方法对卷积型Volterra积分微分方程进行离散，结合卷积核的特性，得到离散方程的线性系统由Toeplitz矩阵、对角矩阵和稀疏矩阵组合而成。考虑Toeplitz矩阵与向量的快速算法，设计系数矩阵与向量的快速计算格式。本文利用GMRES算法与快速计算格式结合，获得一种快速求解线性系统的改进算法，并通过实验验证改进算法的高效性。

关键词

Volterra积分微分方程，快速计算，GMRES，广义多步配置方法

A Fast Algorithm for Convolutional Volterra Integral Differential Equations

Haiyang Li*, Huaiqing Hu, Jingya Liu

School of Mathematics and Statistics, Guizhou University, Guiyang Guizhou

Received: Sep. 4th, 2023; accepted: Oct. 17th, 2023; published: Oct. 25th, 2023

Abstract

Convolutional Volterra integral differential equations are an important class of problems, widely used in biology, economics, among other fields. This study presents a fast algorithm for solving convolutional Volterra integral differential equations. The method involves discretizing the equations using a multi-step collocation approach, combined with the characteristics of the convolution kernels, resulting in a linear system of equations composed of Toeplitz matrices, diagonal matrices, and sparse matrices. Considering fast algorithms for Toeplitz matrices and vectors, a fast computation format for the coefficient matrix and vector is designed. By combining the GMRES algorithm with the fast computation format, an improved algorithm for efficiently solving linear systems is obtained. Experimental results verify the effectiveness of the improved algorithm.

*通讯作者。

文章引用: 李海洋, 胡怀青, 刘婧雅. 卷积型 Volterra 积分微分方程的一种快速算法[J]. 运筹与模糊学, 2023, 13(5): 5571-5580. DOI: 10.12677/orf.2023.135556

Keywords

Volterra Integral Differential Equation, Fast Computation, GMRES, Generalized Multi-Step Collocation Method

1. 引言

研究卷积型 Volterra 积分微分方程(下面简称为卷积型 VIDE)的快速算法，其方程形式如下

$$\begin{cases} y'(t) = a(t)y(t) + g(t) + \int_0^t K(t-s)y(s)\mathrm{d}s, \ t \in I := [0,T], \\ y(0) = y_0, \end{cases} \tag{1}$$

其中 $a(\cdot), g(\cdot) \in C(I)$， $K(t-s)$ 是区域 $D := \{(t,s) | 0 \le s \le t \le T\}$ 上的连续函数，这些条件保证了式(1)解的存在唯一性[1]。

由于卷积型 VIDE 在生物学、经济学中有着广泛的应用，如计算生物学[2]、风险管理[3]等，因此对卷积型 VIDE 解的研究受到大量学者的关注。通常情况下，方程(1)的解析解很难求出，因此研究该方程的数值解法具有重要意义。迄今为止，国内外研究者提出和改进了很多有效求解 VIDE 数值解的方法，如差分求积法[4]、龙格－库塔法[5]、可约求积规则[6]、一般线性法[7] [8]、伽辽金法[9] [10]、配置方法[1]等。在文献[10]中的配置方法是显式类型的求解格式，可以逐步求解得到整个 $y(t)$ 在 I 上的数值逼近。[11]中的数值稳定性分析表示一些特殊的配置节点选取，可以导出 A_0 稳定方法。此外，VDIE 稳定数值解的另一种方法是使用超隐式技术[12]。通过使用下一个子区间的配置节点和 Hermite-Birkhoff 插值来研究求解非判别和刚性 VIDE 的多步配置方法。研究发现，使用计算的配置点明显扩大了稳定域。

在本文中，我们使用广义多步配置方法求解卷积型 VIDE，广义多步配置方法通过在大区间上划分均匀网格，使用子区间两侧的网格节点构造局部多项式，类似的方法已经广泛用于 Volterra 积分方程中[13] [14] [15] [16]，这种方法具有简洁的形式，在不改变线性系统维度的情况下能够取得较高收敛阶的数值解。根据离散时产生线性系统的结构研究他的快速算法。在第 2 节中，构造求解卷积型 VIDE 的广义多步配置方法，分析求解卷积型 VIDE 的广义多步配置方法产生线性系统的结构，并结合 GMRES [17]算法得到快速求解算法。在第 3 节中，我们使用数值算例验证数值方法的高效性。

2. 快速广义多步配置方法

考虑使用广义多步配置方法 GMCM^{k_1,k_2} 离散 Volterra 积分微分方程。设均匀网格 $I_h = \{t_n = nh \,|\, n = 0,1,\cdots,N\}$，其中步长 $h = T/N$。基于 I_h，定义局部基函数

$$\varnothing_j^{\alpha_n,\beta_n}(s) = \prod_{i=-\alpha_n, i\neq j}^{\beta_n+1} \frac{s-j}{j-i}, \ j = -\alpha_n, \cdots, \beta_n + 1. \tag{2}$$

进而导数在 $[t_n, t_{n+1}]$ 上的导数 $y'(t)$ 具有如下表示

$$Y_h'(t_n + sh) = \sum_{i=-\alpha_n}^{\beta_n+1} Y_{n+j}' \varnothing_j^{\alpha_n,\beta_n}(s), \ s \in [0,1], \tag{3}$$

其中 $Y_{n+i}' := Y_h'(t_n + jh)$，

$$(\alpha_n,\beta_n)=\begin{cases}(n,k_1+k_2-n),0\le n\le k_1-1,\\(k_1,k_2),k_1\le n\le N-k_2-1,\\(k_1+k_2+n+1-N,N-n-1),N-k_2\le n\le N-1,\end{cases}$$

k_1,k_2 是非负整数。

在 $[t_n,t_{n+1}]$ 上对(3)积分

$$Y_h(t_n+sh)=Y_n+h\sum_{i=-\alpha_n}^{\beta_n+1}Y'_{n+j}\mathbf{MOM}_{n,j}(s),s\in[0,1],\tag{4}$$

其中 $Y_n:=Y_h(t_n)$，$\mathbf{MOM}_{n,j}(s)=\int_0^s\varnothing_j^{\alpha_n,\beta_n}(v)\mathrm{d}v$。为了描述方程，定义下列符号

$\mathbf{MOMK1}_{n,j}=\int_0^1K(t_{n+1}-t_j-sh)\mathrm{d}s$，$\mathbf{MOMK2}_{n,j}^i=\int_0^1K(t_{n+1}-t_j-sh)\mathbf{MOM}_{j,i}(s)\mathrm{d}s$。

进而我们得到

$$\begin{aligned}\int_0^{t_{n+1}}K(t_{n+1}-s)Y_h(s)\mathrm{d}s&=\sum_{j=0}^n\int_{t_j}^{t_{j+1}}K(t_{n+1}-s)Y_h(s)\mathrm{d}s\\&=h\sum_{j=0}^n\int_0^1K(t_{n+1}-t_j-vh)Y_h(t_j+vh)\mathrm{d}v\\&=h\sum_{j=0}^n\int_0^1K(t_{n+1}-t_j-vh)\left(Y_j+h\sum_{i=-\alpha_j}^{\beta_j+1}Y'_{j+i}\mathbf{MOM}_{j,i}(v)\right)\mathrm{d}v\\&=h\sum_{j=0}^nY_j\mathbf{MOMK1}_{n,j}+h^2\sum_{j=0}^n\sum_{i=-\alpha_j}^{\beta_j+1}Y'_{j+i}\mathbf{MOMK2}_{n,j}^i.\end{aligned}$$

注意到

$$Y'_n=a(t_{n+1})Y_{n+1}+g(t_{n+1})+\int_0^{t_{n+1}}K(t_{n+1}-s)Y_h(s)\mathrm{d}s,n=0,1,\cdots,N-1.\tag{5}$$

因此，我们有，

$$\hat{\mathbf{P}}\mathbf{Y}_D=\hat{\mathbf{G}},\tag{6}$$

其中

$$\hat{\mathbf{P}}=\mathbf{E}_N-h\mathbf{AF}-h^2\mathbf{BF}-h^2\mathbf{C},$$

$$\hat{\mathbf{G}}=\mathbf{G}_N+Y_0(\mathbf{A}+h\mathbf{B})1+hY'_0(\mathbf{AF}_0+h\mathbf{BF}_0+h\mathbf{C}_0)+hY_0\mathbf{B}_0,$$

$$\hat{\mathbf{B}}=\left(\hat{b}_{i,j}\right)_{N\times(N+1)},\hat{b}_{i,j}=\begin{cases}\mathbf{MOMK1}_{n,j}&i>j,\\0,&i\le j,\end{cases}$$

$$\hat{\mathbf{C}}=\left(\hat{c}_{i,j}\right)_{N\times(N+1)},\hat{c}_{i,j}=\sum_{k=-k_1-k_2}^{k_1+k_2+1}\mathbf{MOMK3}_{i-1,j-k}^k,$$

$$\hat{\mathbf{F}}=\left(\hat{f}_{i,j}\right)_{N\times(N+1)},\hat{f}_{i,j}=\sum_{k=-k_1-k_2}^{k_1+k_2+1}\mathbf{MOMK4}_{i-1,j-k}^k,$$

$$\mathbf{MOMK3}_{i-1,j-k}^k=\begin{cases}\mathbf{MOMK2}_{i-1,j-k}^k,&\left(0<j-k\le\min(i-1,N-1)\&\left(\alpha_{j-k}\le k\le\beta_{j-k}\right)\right),\\0,&\text{其他},\end{cases}$$

$$\mathbf{MOMK1}_{i-1,j-k}^k=\begin{cases}\mathbf{MOM}_{j-k,k}(1),&\left(0<j-k\le\min(i-1,N-1)\&\left(\alpha_{j-k}\le k\le\beta_{j-k}\right)\right),\\0,&\text{其他},\end{cases}$$

$$\mathbf{B}=\hat{\mathbf{B}}(1:N,2:N+1),\mathbf{B}_0=\hat{\mathbf{B}}(1:N,1),$$

$$\mathbf{C}=\hat{\mathbf{C}}(1:N,2:N+1),\mathbf{C}_0=\hat{\mathbf{C}}(1:N,1),$$

$$\mathbf{F}=\hat{\mathbf{F}}(1:N,2:N+1),\mathbf{F}_0=\mathbf{F}(1:N,1),$$

$$\mathbf{Y}_D=\begin{pmatrix}Y_1'\\Y_2'\\\vdots\\Y_N'\end{pmatrix},\mathbf{G}_N=\begin{pmatrix}g(t_1)\\g(t_2)\\\vdots\\g(t_N)\end{pmatrix},\mathbf{A}_N=\begin{pmatrix}a(t_1)&&&\\&a(t_1)&&\\&&\ddots&\\&&&a(t_1)\end{pmatrix}.$$

$\mathbf{1}$表示元素全是 1 的 N 行 1 列的列向量，上述矩阵描述中 $i=1,2,\cdots,N$， $j=1,2,\cdots,N+1$。注意到 $\mathbf{MOMK1}_{n,j}=\mathbf{MOMK1}_{n+1,j+1}$， $n=0,1,\cdots,N-2$， $j=0,1,\cdots,N-1$。

当 $n=k_1,k_1+1,\cdots,N-k_2-2$ 时， $\mathbf{MOM}_{n,j}(s)=\mathbf{MOM}_{n-1,j}(s)$，进而 $\mathbf{MOMK2}^i_{n,j}=\mathbf{MOMK2}^i_{n+1,j+1}$，因此，式(6)可以写为如下形式，

$$\left(\mathbf{E}_N-\left(h\mathbf{A}+h^2\mathbf{B}\right)\left(\mathbf{F}_T+\mathbf{F}_S\right)-h^2\left(\mathbf{C}_T+\mathbf{C}_S\right)\right)\mathbf{Y}_\mathbf{D}=\hat{\mathbf{G}},\tag{7}$$

即 $\hat{\mathbf{P}}=\mathbf{E}_N-h\mathbf{A}\mathbf{F}-h^2\mathbf{B}\mathbf{F}-h^2\mathbf{C}=\mathbf{E}_N-\left(h\mathbf{A}+h^2\mathbf{B}\right)\left(\mathbf{F}_T+\mathbf{F}_S\right)-h^2\left(\mathbf{C}_T+\mathbf{C}_S\right)$，其中 $\mathbf{B}$ 是 Toeplitz 矩阵，其第一行元素为 $\mathbf{B}(1,1:N)$，第一列元素为 $\mathbf{B}(1:N,1)$， $\mathbf{F}_T+\mathbf{F}_S=\mathbf{F}$， $\mathbf{C}_T+\mathbf{C}_S=\mathbf{C}$， $\mathbf{F}_T$ 是 Toeplitz 矩阵，其第一行元素为 $F_1=\left(\mathbf{F}\left(k_1+k_2+1,k_1+k_2+1:N\right),\mathbf{0}_{1\times(k_1+k_2)}\right)$，第一列元素为 $F_2=\left(\mathbf{F}\left(k_1+k_2+1:N,k_1+k_2+1\right);\mathbf{0}_{(k_1+k_2)\times1}\right)$， $\mathbf{F}_S$ 是稀疏矩阵，其非零元素及位置为 $\mathbf{F}_S(:,i)=\mathbf{F}(:,i)-\left(\mathbf{0}_{(i-1)\times1};F_2(1:N-i+1)\right)$， $i=1,2,\cdots,k_1+k_2$，

$\mathbf{F}_S\left(N-j+1,N-k_1-k_2-1:i\right)=\mathbf{F}\left(N-j+1,N-k_1-k_2-1:i\right)-\left(F_2\left(k_1+k_2+3-j:-1:1\right)^{\mathrm{T}},F_1\left(1:j-1\right)\right)$，

$j=1,2,\cdots,k_2$。 $\mathbf{C}_T$ 是 Toeplitz 矩阵，其第一行元素为 $C_1=\left(\mathbf{C}\left(k_1+k_2+1,k_1+k_2+1:N\right),\mathbf{0}_{1\times(k_1+k_2)}\right)$，第一列元素为 $C_2=\left(\mathbf{C}\left(k_1+k_2+1:N,k_1+k_2+1\right);\mathbf{0}_{(k_1+k_2)\times1}\right)$， $\mathbf{C}_S$ 是稀疏矩阵，其非零元素及位置为

$\mathbf{C}_S(:,i)=\mathbf{C}(:,i)-\left(\mathbf{0}_{(i-1)\times1};C_2(1:N-i+1)\right)$， $i=1,2,\cdots,k_1+k_2$，

$\mathbf{C}_S\left(N-j+1,N-k_1-k_2-1:i\right)=\mathbf{C}\left(N-j+1,N-k_1-k_2-1:i\right)-\left(C_2\left(k_1+k_2+3-j:-1:1\right)^{\mathrm{T}},C_1\left(1:j-1\right)\right)$，

$j=1,2,\cdots,k_2$。注：上述过程中矩阵元素的选择参考 Matlab 矩阵元素的选取。此外在表 1 中，我们给出式子(7)中组成系数矩阵 $\hat{\mathbf{P}}$ 的各部分矩阵的情况。

Table 1. Detailed table of matrices for each part during coefficient matrix $\hat{\mathrm{P}}$ decomposition
表 1. 系数矩阵 $\hat{\mathrm{P}}$ 分解时各部分矩阵详细表

	矩阵类型	存储量
$\mathbf{A}$	对角矩阵	N
$\mathbf{B}$	Toeplitz 矩阵	$2N$
$\mathbf{F}_T$	Toeplitz 矩阵	$2N$
$\mathbf{F}_S$	稀疏矩阵	D_{num}
$\mathbf{C}_T$	Toeplitz 矩阵	$2N$
$\mathbf{C}_S$	稀疏矩阵	D_{num}

其中，

$$D_{num}=\begin{cases}k_2N+(k_1+k_2+2)k_2, & k_1=0,\\ (k_1+k_2+1)N+(k_1+k_2+2)k_2, & k_1>0,\end{cases}$$

对于 Toeplitz 矩阵，仅需要存储第一行和第一列即可，因此其存储量为 $2N$。

如果不考虑 $\hat{\mathbf{P}}$ 的结构，那么我们可以选择直接求解(6)式，需要 $O(N^3)$ 的计算量，或者使用迭代算法广义极小残差(GMRES)求解(6)式，其计算量主要来自于 $\hat{\mathbf{P}}$ 与向量的乘法运算，因此需要 $O(N^2)$ 的计算量。根据表 1 可知，系数矩阵 $\hat{\mathbf{P}}$ 由 Toeplitz 矩阵，稀疏矩阵，以及对角矩阵组合而成，充分利用 $\hat{\mathbf{P}}$ 的结构，优化 $\hat{\mathbf{P}}$ 与向量相乘时的计算量。

首先考虑 Toeplitz 矩阵与向量的快速计算，即根据已有的 Toeplitz 矩阵构造循环矩阵，进而通过快速 Fourier 方法计算。由 Toeplitz 矩阵 T 构造 $\hat{T}$，

$$T=\begin{pmatrix}t_0 & t_{-1} & t_{-2} & \cdots & t_{-n}\\ t_1 & t_0 & t_{-1} & \cdots & t_{-n+1}\\ t_2 & t_1 & t_0 & \cdots & t_{-n+2}\\ \vdots & \vdots & \vdots & & \vdots\\ t_n & t_{n-1} & t_{n-2} & \cdots & t_0\end{pmatrix},\quad \hat{T}=\begin{pmatrix}0 & t_n & t_{n-1} & \cdots & t_1\\ t_{n-1} & 0 & t_n & \cdots & t_2\\ t_{n-2} & t_{n-1} & 0 & \cdots & t_3\\ \vdots & \vdots & \vdots & & \vdots\\ t_{-1} & t_{-2} & t_{-3} & \cdots & 0\end{pmatrix}$$

将 T 与向量 b 的乘法嵌入式(8)计算，

$$\begin{pmatrix}T & \hat{T}\\ \hat{T} & T\end{pmatrix}\begin{pmatrix}b\\ 0\end{pmatrix}=\begin{pmatrix}Tb\\ *\end{pmatrix} \tag{8}$$

其中，$\begin{pmatrix}T & \hat{T}\\ \hat{T} & T\end{pmatrix}$ 是循环矩阵，与向量 $\begin{pmatrix}b\\ 0\end{pmatrix}$ 的乘法使用快速 Fourier 方法计算，计算量为 $O(N\log(N))$。而稀疏矩阵与向量的乘法所需计算量为 $O(N)$，对角矩阵与向量的乘法也是 $O(N)$。进而 $\hat{\mathbf{P}}$ 与任意向量 H 的乘法 $\bar{G}:=\hat{\mathbf{P}}H$ 可分为公式(9)中的 5 个步骤计算：

$$\begin{aligned}&\bar{G}_F=\mathbf{F}_TH+\mathbf{F}_SH, && (\text{step-1})\\ &\bar{G}_A=\mathbf{A}\bar{G}_F, && (\text{step-2})\\ &\bar{G}_B=\mathbf{B}\bar{G}_F, && (\text{step-3})\\ &\bar{G}_C=\mathbf{C}_TH+\mathbf{C}_SH, && (\text{step-4})\\ &\bar{G}=H-h(\bar{G}_A+h\bar{G}_B+h\bar{G}_C). && (\text{step-5})\end{aligned} \tag{9}$$

Table 2. Formula (9) Calculation amount for each step
表 2. 公式(9)各步的计算量

step	计算公式	乘法计算量	
step-1	$\mathbf{F}_TH+\mathbf{F}_SH$	$O(N\log(N))+O(N)$	Toeplitz 矩阵与向量乘法，稀疏矩阵与向量乘法
step-2	$\mathbf{A}\bar{G}_F$	$O(N)$	对角矩阵与向量乘法
step-3	$\mathbf{B}\bar{G}_F$	$O(N\log(N))$	Toeplitz 矩阵与向量乘法
step-4	$\mathbf{C}_TH+\mathbf{C}_SH$	$O(N\log(N))+O(N)$	Toeplitz 矩阵与向量乘法，稀疏矩阵与向量乘法
step-5	$H-h(\bar{G}_A+h\bar{G}_B+h\bar{G}_C)$	$O(N)$	数与向量乘法

如表 2 所示，我们给出了使用公式(9)计算 $\hat{\mathbf{P}}H$ 的各步骤的计算量，其中计算量最大的是 Toeplitz 矩阵与向量的快速计算，为 $O(N\log(N))$，其余矩阵与向量的乘积中，计算量均为 $O(N)$，因此使用公式(9)计算 $\hat{\mathbf{P}}H$ 的计算量为 $O(N\log(N))$。

进而我们考虑 GMRES 算法(如算法 1 所示)，该算法中计算量主要来源于矩阵 $\hat{P}$ 与向量的乘法即第 2 行 $\hat{P}Y_D^{(0)}$ 与第 4 行 $\hat{P}v^{(i)}$ 的计算，这个两步的计算量均为 $O\left(N^2\right)$。考虑矩阵 $\hat{P}$ 与向量的快速计算格式公式(9)，结合 GMRES 算法，我们得到算法 2 所示的改进算法，进而可以将第 2 行 $\hat{P}Y_D^{(0)}$ 与第 4 行 $\hat{P}v^{(i)}$ 的计算量降为 $O\left(N\log\left(N\right)\right)$，从而优化整个算法的计算量。

Algorithm 1. GMRES algorithm for solving linear systems $\hat{\mathrm{P}}\mathbf{Y}_D=\hat{\mathrm{G}}$

算法 1. GMRES 算法求解线性系统 $\hat{\mathrm{P}}\mathbf{Y}_D=\hat{\mathrm{G}}$

1. 给定初值 $Y_D^{(0)}$；
2. 计算初始残差 $r=\hat{G}-\hat{P}Y_D^{(0)}$，$v^{(1)}=r/\|r\|_2$，$s=\|r\|_2 e_1$，其中 e_1 为第一个元素为 1，其余元素为 0 的列向量。
3. FOR $i=1,2,\cdots$
4. $w=\hat{P}v^{(i)}$.
5. FOR $k=1,2,\cdots,i$
6. $h_{k,i}=\left(w,v^{(k)}\right)$.
7. $w=w-h_{k,i}v^{(k)}$.
8. ENDFOR
9. $h_{i+1,i}=\|w\|_2$.
10. $v^{(i+1)}=w/h_{i+1,i}$.
11. 在 $\left(h_{1,i},\cdots,h_{i+1,i}\right)$ 上应用 $J_1,\cdots,J_{i-1}$
12. 构造 J_i，作用于 $h_{\cdot,i}$ 的第 i 和第 $i+1$ 个分量，使得 $J_ih_{\cdot,i}$ 的第 $i+1$ 个分量为 0。
13. $s=J_is$；
14. 如果 $s(i+1)$ 足够小，则运行算法 3 UPDATA $\left(\bar{\mathbf{Y}}_D,i\right)$ 并结束循环。
15. ENDFOR

Algorithm 2. Improved algorithm for solving linear systems $\hat{\mathrm{P}}\mathbf{Y}_D=\hat{\mathrm{G}}$

算法 2. 改进算法求解线性系统 $\hat{\mathrm{P}}\mathbf{Y}_D=\hat{\mathrm{G}}$

1. 给定初值 $Y_D^{(0)}$；
2. 计算初始残差 $r=\hat{G}-\hat{P}Y_D^{(0)}$，其中 $\hat{P}Y_D^{(0)}$ 使用公式(9)计算，$v^{(1)}=r/\|r\|_2$，$s=\|r\|_2 e_1$，其中 e_1 为第一个元素为 1，其余元素为 0 的列向量。
3. FOR $i=1,2,\cdots$
4. $w=\hat{P}v^{(i)}$，其中 $\hat{P}v^{(i)}$ 使用公式(9)计算。
5. FOR $k=1,2,\cdots,i$
6. $h_{k,i}=\left(w,v^{(k)}\right)$.
7. $w=w-h_{k,i}v^{(k)}$.
8. ENDFOR
9. $h_{i+1,i}=\|w\|_2$.
10. $v^{(i+1)}=w/h_{i+1,i}$.
11. 在 $\left(h_{1,i},\cdots,h_{i+1,i}\right)$ 上应用 $J_1,\cdots,J_{i-1}$
12. 构造 J_i，作用于 $h_{\cdot,i}$ 的第 i 和第 $i+1$ 个分量，使得 $J_ih_{\cdot,i}$ 的第 $i+1$ 个分量为 0。
13. $s=J_is$;
14. 如果 $s(i+1)$ 足够小，则运行算法 3 UPDATA $\left(\bar{\mathbf{Y}}_D,i\right)$ 并结束循环。
15. ENDFOR

Algorithm 3. UPDATA $\left(\bar{\mathbf{Y}}_D, i\right)$ algorithm

算法 3. UPDATA $\left(\bar{\mathbf{Y}}_D, i\right)$ 算法

1. 求 $Hy=\bar{s}$ 的解 y，其中 H 是 $i\times i$ 的上三角矩阵，以 $h_{i,j}$ 作为其元素(如果 H 是奇异的，则在最小二乘意义下求解)，$\bar{s}$ 表示 s 的前 i 个分量。
2. $\bar{\mathbf{Y}}_D = Y_D^{(0)} + y_1 v^{(1)} + y_2 v^{(2)} + \cdots + y_i v^{(i)}$.

3. 实例分析与应用

在本节中，我们通过数值算例说明改进算法求解线性系统(7)的高效性，实验主要对比 GMRES 求解线性系统(7)与改进算法求解线性系统(7)的 CPU 计算时间，收敛时的相对残差，迭代次数，迭代过程中的相对残差变化情况。

例：求解卷积型 Volterra 积分微分方程

$$y'(t)=\frac{2}{1+t}y(t)+\mathrm{e}^t+\int_0^t 2\cos(t-s)y(s)\mathrm{d}s,$$

其中 $y(0)=1, t\in[0,8]$，其解析解是 $y=(1+t)^2\mathrm{e}^t$。数值实验是在 Matlab2019a 中实现的，其中，给定收敛条件为相对残 $\varepsilon<10^{-10}$，

$$\varepsilon=\left\|\hat{\mathbf{G}}-\hat{\mathbf{P}}\mathbf{Y}_D\right\|_2 \Big/ \left\|\hat{\mathbf{G}}\right\|_2.$$

Table 3. GMCM^{0,2} : Comparison of GMRES and improved algorithm solution time
表 3. GMCM^{0,2}：GMRES 与改进算法求解时长对比

	GMRES			改进算法		
N	相对残差	迭代次数	CPU 计算时间	相对残差	迭代次数	CPU 计算时间
100	1.26×10^{-11}	17	4.18×10^{-3}	1.26×10^{-11}	17	5.17×10^{-3}
200	1.52×10^{-11}	17	5.85×10^{-3}	1.52×10^{-11}	17	7.12×10^{-3}
400	1.62×10^{-11}	17	7.43×10^{-3}	1.62×10^{-11}	17	9.91×10^{-3}
800	1.65×10^{-11}	17	9.55×10^{-3}	1.65×10^{-11}	17	1.16×10^{-2}
1600	1.67×10^{-11}	17	1.77×10^{-3}	0.67×10^{-11}	17	1.71×10^{-2}
3200	1.67×10^{-11}	17	6.55×10^{-3}	0.67×10^{-11}	17	2.96×10^{-2}

Table 4. GMCM^{1,1} : Comparison of GMRES and improved algorithm solution time
表 4. GMCM^{1,1}：GMRES 与改进算法求解时长对比

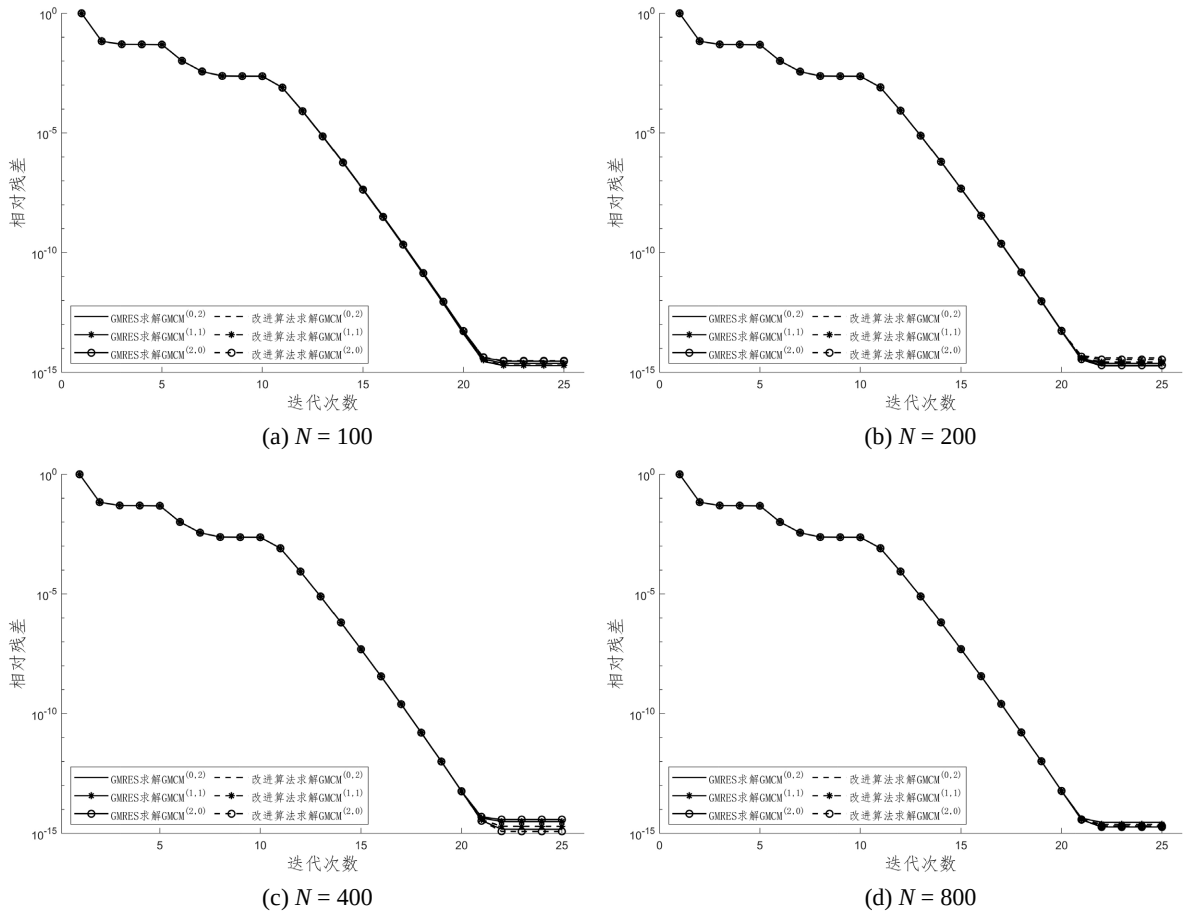
	GMRES			改进算法		
N	相对残差	迭代次数	CPU 计算时间	相对残差	迭代次数	CPU 计算时间
100	1.26×10^{-11}	17	4.18×10^{-3}	1.26×10^{-11}	17	5.17×10^{-3}
200	1.52×10^{-11}	17	5.85×10^{-3}	1.52×10^{-11}	17	7.12×10^{-3}
400	1.62×10^{-11}	17	7.43×10^{-3}	1.62×10^{-11}	17	9.91×10^{-3}
800	1.65×10^{-11}	17	9.55×10^{-3}	1.65×10^{-11}	17	1.16×10^{-2}
1600	1.67×10^{-11}	17	1.77×10^{-3}	1.67×10^{-11}	17	1.71×10^{-2}
3200	1.67×10^{-11}	17	6.55×10^{-3}	1.67×10^{-11}	17	2.96×10^{-2}

对区间$[0,8]$作等距划分，取不同 N，对于 GMCM^{0,2},GMCM^{1,1},GMCM^{2,0} 得到的线性系统分别使用 GMRES 算法和改进算法求解所得 CPU 计算时长如表 3，表 4 和表 5，图 1 给出了迭代过程中相对残差的变化情况，图 2 给出了两种求解算法的 CPU 计算时间。

从表 3、表 4 和表 5 中可知对于指定算法结束条件相对残$\varepsilon<10^{-10}$，GMRES 算法和改进算法结束时相对残差几乎相同，且迭代次数相同，但随 N 增大，改进算法的 CPU 计算时间会小于 GMRES 算法的

Table 5. GMCM^{2,0} : Comparison of GMRES and improved algorithm solution time
表 5. GMCM^{2,0} ：GMRES 与改进算法求解时长对比

	GMRES			改进算法		
N	相对残差	迭代次数	CPU 计算时间	相对残差	迭代次数	CPU 计算时间
100	1.26×10^{-11}	17	4.18×10^{-3}	1.26×10^{-11}	17	5.17×10^{-3}
200	1.52×10^{-11}	17	5.85×10^{-3}	1.52×10^{-11}	17	7.12×10^{-3}
400	1.62×10^{-11}	17	7.43×10^{-3}	1.62×10^{-11}	17	9.91×10^{-3}
800	1.65×10^{-11}	17	9.55×10^{-3}	1.65×10^{-11}	17	1.16×10^{-2}
1600	1.67×10^{-11}	17	1.77×10^{-3}	1.67×10^{-11}	17	1.71×10^{-2}
3200	1.67×10^{-11}	17	6.55×10^{-3}	1.67×10^{-11}	17	2.96×10^{-2}



(a) N = 100　(b) N = 200

(c) N = 400　(d) N = 800

Figure 1. Changes in relative residuals during the iteration process
图 1. 迭代过程中相对残差变化情况

CPU 计算时间，当 N 增大一倍，GMRES 算法的计算时间增加倍数逐渐趋近于 4，改进算法的 CPU 计算时间增长倍数逐渐趋近于 2。

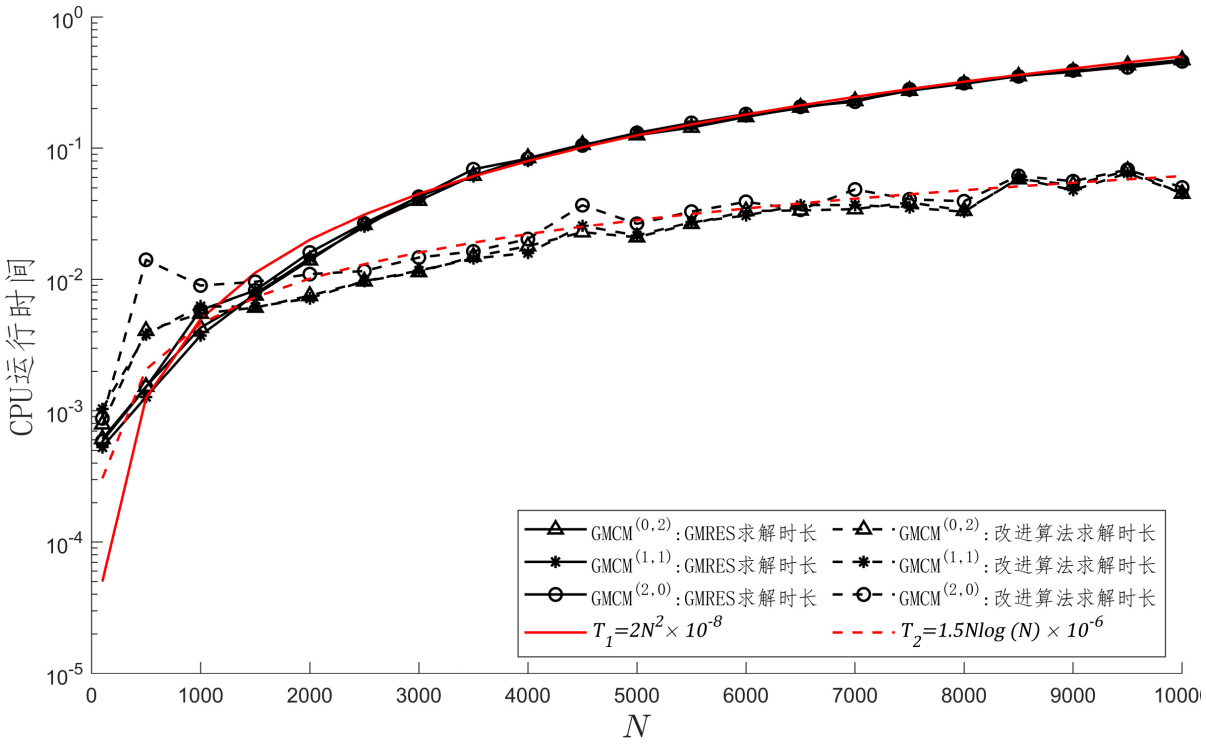


Figure 2. Comparison between GMRES and improved algorithms in solving problems
图 2. GMRES 与改进算法求解时对比

图 1 给出迭代过程中相对残差的变化情况，针对本文所取的例子，不同方法获得的线性系统在求解过程中相对残差没有较大差距，且 GMRES 方法和改进算法的相对残差也没有较大差距。在图 2 中，我们绘制了 GMRES 算法与改进算法求解 $\mathrm{GMCM}^{0,2},\mathrm{GMCM}^{1,1},\mathrm{GMCM}^{2,0}$ 的 CPU 计算时长，以及两条参考曲线 $T_1=2N^2\times10^{-8},T_2=1.5N\log N\times10^{-6}$。当 N 增大时，GMRES 算法的求解时间靠近 T_1，改进算法的求解时间靠近 T_2，也就是说 GMERS 的算法求解时间为 $O\left(N^2\right)$，GMRES 的求解时间为 $O\left(N\log\left(N\right)\right)$，验证了第 1 节 GMRES 算法的计算量为 $O\left(N^2\right)$，改进算法的计算量为 $O\left(N\log\left(N\right)\right)$。而且 k_1,k_2 的值对 GMRES 和改进算法的 CPU 计算时间影响不大，两种求解线性系统的算法与线性系统的维度 N 有关。此外，从图中可知随着 N 变大，改进算法的计算时间小于 GMRES 的计算时间且改进算法的 CPU 计算时间的增长率小于 GMRES，因此改进算法能够快速高效计算卷积型 VIDE 离散出的大型线性系统。

4. 结论

使用广义多步配置方法(GMCM^{k_1,k_2})离散卷积型 VIDE，分析离散后的线性系统的结构组成，结合 Toeplitz 矩阵与向量的快速计算，进而得到系数矩阵与向量的快速计算。使用 GMRES 算法与快速计算格式结合，构建了一种适合求解该类线性系统的改进算法，改进后的算法的计算量只需 $O\left(N\log\left(N\right)\right)$，比 GMRES 算法的计算量更小，从实验结果看，改进算法与 GMRES 算法在求解同一问题时，改进算法需要的求解时间比 GMRES 算法的求解时间更短，当 N 值增大时，改进算法在计算时间上更小，更具优势。因此改进算法可以高效求解这一类大规模线性系统。

参考文献

[1] Brunner, H. (2004) Collocation Methods for Volterra Integral and Related Functional Equations. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511543234

[2] De Gaetano, A. and Arino, O. (2000) Mathematical Modelling of the Intravenous Glucose Tolerance Test. *Journal of Mathematical Biology*, **40**, 136-168. https://doi.org/10.1007/s002850050007

[3] Makroglou, A. (2003) Integral Equations and Actuarial Risk Management: Some Models and Numerics. *Mathematical Modelling and Analysis*, **8**, 143-154. https://doi.org/10.3846/13926292.2003.9637219

[4] Abdi, A. and Hosseini, S. (2018) The Barycentric Rational Difference-Quadrature Scheme for Systems of Volterra Integro-Differential Equations. *SIAM Journal on Scientific Computing*, **40**, A1936-A1960. https://doi.org/10.1137/17M114371X

[5] Wen, J., Huang, C. and Li, M. (2019) Stability Analysis of Runge-Kutta Methods for Volterra Integro-Differential Equations. *Applied Numerical Mathematics*, **146**, 73-88. https://doi.org/10.1016/j.apnum.2019.07.004

[6] Chen, H. and Zhang, C. (2011) Boundary Value Methods for Volterra Integral and Integro-Differential Equations. *Applied Mathematics and Computation*, **218**, 2619-2630. https://doi.org/10.1016/j.amc.2011.08.001

[7] Mahdi, H., Abdi, A. and Hojjati, G. (2018) Efficient General Linear Methods for a Class of Volterra Integro-Differential Equations. *Applied Numerical Mathematics*, **127**, 95-109. https://doi.org/10.1016/j.apnum.2018.01.001

[8] Almasoodi, A., Abdi, A. and Hojjati, G. (2021) A GLMs-Based Difference-Quadrature Scheme for Volterra Integro-Differential Equations. *Applied Numerical Mathematics*, **163**, 292-302. https://doi.org/10.1016/j.apnum.2021.02.001

[9] Lin, T., Lin, Y., Rao, M. and Zhang, S. (2000) Petrov-Galerkin Methods for Linear Volterra Integro-Differential Equations. *SIAM Journal on Numerical Analysis*, **38**, 937-963. https://doi.org/10.1137/S0036142999336145

[10] Yi, L. and Guo, B. (2015) An h-p Version of the Continuous Petrov-Galerkin Finite Element Method for Volterra Integro-Differential Equations with Smooth and Nonsmooth Kernels. *SIAM Journal on Numerical Analysis*, **53**, 2677-2704. https://doi.org/10.1137/15M1006489

[11] Cardone, A. and Conte, D. (2013) Multistep Collocation Methods for Volterra Integro-Differential Equations. *Applied Mathematics and Computation*, **221**, 770-785. https://doi.org/10.1016/j.amc.2013.07.012

[12] Fazeli, S. and Hojjati, G. (2015) Numerical Solution of Volterra Integro-Differential Equations by Superimplicit Multistep Collocation Methods. *Numerical Algorithms*, **68**, 741-768. https://doi.org/10.1007/s11075-014-9870-8

[13] Ma, J. and Xiang, S. (2017) A Collocation Boundary Value Method for Linear Volterra Integral Equations. *Journal of Scientific Computing*, **71**, 1-20. https://doi.org/10.1007/s10915-016-0289-3

[14] Liu, L. and Ma, J. (2021) Block Collocation Boundary Value Solutions of the First-Kind Volterra Integral Equations. *Numerical Algorithms*, **86**, 911-932. https://doi.org/10.1007/s11075-020-00917-6

[15] Liu, L. and Ma, J. (2022) Collocation Boundary Value Methods for Auto-Convolution Volterra Integral Equations. *Applied Numerical Mathematics*, **177**, 1-17. https://doi.org/10.1016/j.apnum.2022.03.004

[16] Chen, H. and Ma, J. (2022) Solving the Third-Kind Volterra Integral Equation via the Boundary Value Technique: Lagrange Polynomial versus Fractional Interpolation. *Applied Mathematics and Computation*, **414**, Article 126685. https://doi.org/10.1016/j.amc.2021.126685

[17] Barrett, R., *et al.* (1994) Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Society for Industrial and Applied Mathematics.