An Efficient Alternative to PPIA for Tensor Product Interpolation Surfaces

Chengzhi Liu, and Juncheng Li

***Abstract*—To improve the computational efficiency of data interpolation, we have exploited the diagonally compensated splitting iteration for the collocation matrix equation. The proposed splitting iteration is the matrix form of the preconditioned progressive iterative approximation (PPIA). Therefore, the proposed iteration format is convergent. Moreover, the proposed iteration involves only BLAS-3 operations, and we can expect that the proposed iteration would perform better than the PPIA regarding computational efficiency. The numerical results also show that the proposed splitting iteration outperforms the PPIA regarding computational time and stability.**

***Index Terms*—splitting iteration, PPIA, tensor product Bézier surface, diagonally compensated reduction.**

I. Introduction

CONSIDER the data interpolation by using the progressive iterative approximation (PIA) for tensor product Bézier surfaces [1], [2]. Let $\{\mathbf{v}_{ij}\}_{i=0,1,\dots,n}^{j=0,1,\dots,m} \in \mathbb{R}^3$ be a set of data to be interpolated, and let (x_i, y_j) be the parameters of $\mathbf{v}_{ij}$. Then, using the procedure of PIA, we can iteratively generate a Bézier surface sequence

$$\mathbf{B}^{(k)}(x,y) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{u}_{ij}^{(k)} b_i^n(x) b_j^m(y), k = 0,1,\dots,$$

where $\{b_i^n(t)\}_{i=0}^n$, $\{b_j^m(t)\}_{i=0}^m$ are the Bernstein bases, and

$$\begin{cases} \mathbf{u}_{ij}^{(k)} = \mathbf{v}_{ij}, & k=0, \\ \mathbf{u}_{ij}^{(k)} = \mathbf{u}_{ij}^{(k-1)} + [\mathbf{v}_{ij} - \mathbf{B}^{(k-1)}(x_i,y_j)], & k = 1,2,\cdots. \end{cases} \tag{1}$$

As noted in [1], for $k = 0,1,\dots,$ the sequence $\mathbf{B}^{(k)}(x,y)$ iteratively approximates to the Bézier interpolation surface of $\{\mathbf{v}_{ij}\}_{i=0,1,\dots,n}^{j=0,1,\dots,m}$, i.e.,

$$\lim_{k\to\infty} \mathbf{B}^{(k)}(x_i,y_j) = \mathbf{v}_{ij}, i = 0,1,\dots,n; j = 0,1,\dots,m.$$

Denote by I_{nm} the identity matrix of order nm, by $\mathbf{vec}(A)$ the column-stacking vector of a matrix A, by $\otimes$ the Kronecker product, and by $B_1 = (b_j^n(x_i))_{n\times n}$, $B_2 = (b_j^m(y_i))_{m\times m}$ the collocation matrices resulting from the Bernstein bases. By using the Kronecker product, Equation (1) can be written in the matrix-vector form

$$\mathbf{vec}(\mathbf{U}^{(k+1)}) = \mathbf{vec}(\mathbf{V}) + (I_{nm} - B_2 \otimes B_1)\mathbf{vec}(\mathbf{U}^{(k)}), \tag{2}$$

Manuscript received April 26, 2023; revised November 25, 2023.

This work was supported in part by the National Natural Science Foundation of China under Grant 12101225, the Natural Science Foundation of Hunan Province under Grant 2023JJ50080, and the Scientific Research Funds of Hunan Provincial Education Department under Grant 21B0790.

C. Z. Liu is an associate professor of School of Mathematics and Finance, Hunan University of Humanities, Science and Technology, Loudi 417000, P. R. China (corresponding author to provide e-mail: it-rocket@163.com).

J. C. Li is a professor of School of Mathematics and Finance, Hunan University of Humanities, Science and Technology, Loudi 417000, P. R. China (e-mail: lijuncheng82@126.com).

where

$$\mathbf{U}^{(0)} = \mathbf{V} = (\mathbf{v}_{ij})_{i=0,1,\dots,n}^{j=0,1,\dots,m}$$

and

$$\mathbf{U}^{(k)} = (\mathbf{u}_{ij}^{(k)})_{i=0,1,\dots,n}^{j=0,1,\dots,m}.$$

The iteration (2) is the so-called PIA, which has played an important role in data interpolation in recent years; see [3] and a large literature therein. To improve the interpolation efficiency, numerous techniques have been vigorously developed to accelerate the convergence rate of PIA, see [2], [4], [5], [6], [8], [9], [10], [11] and so on.

Based on the technique of diagonally compensate reduction, the preconditioned progressive iterative approximation (PPIA) was proposed in [11], then this preconditioning technique was extended to the PIA for tensor product Bézier surfaces ([2]). We remark here that the system (2) can be solved by equivalently solving its corresponding matrix equation ([12], [13]). Moreover, numerical algorithms for the matrix equation are typically the preferred choice because the system (2) is ill-conditioned and expensive to solve. This motivates us to exploit numerical algorithms for the corresponding matrix equation to improve the computational efficiency and the stability of data interpolation.

The remainder of this paper is organized as follows: Section II introduces the splitting iteration for the matrix equation and the residual-updating iteration for the collocation matrix equation. Then, the diagonally compensated splitting iteration is exploited in Section III. In Section IV, some numerical examples are given to illustrate the computational efficiency of our proposed method. Finally, some concluding remarks are given in the last section.

II. Related work

A. Splitting iteration of $AXB = C$

Consider solving the matrix equation $AXB = C$, where A is an $n \times n$ matrix, B is an $m \times m$ matrix, and X, C are $n \times m$ matrices. Let $A = F - G$ and $B = \hat{F} - \hat{G}$ be the splittings of A and B, respectively. Then, if F and $\hat{F}$ are nonsingular, we can obtain the splitting iteration sequence ([12])

$$X^{(k+1)} = X^{(k)} + F^{-1}(C - AX^{(k)}B)\hat{F}^{-1}. \tag{3}$$

B. Alternative of PIA

Note that the solution to the system (2) is the column-stacking vector of the solution to the matrix equation ([12], [13])

$$B_1\mathbf{U}B_2^T = \mathbf{V}. \tag{4}$$

Remark 1: Although the solution to (2) is the column-stacking vector of the solution to the matrix equation (4). As stated in [14], the system (2) is rather ill-conditioned, and numerical solvers for (2) are expensive compared to those for (4). In [13], Liu et al. also pointed out that the system (2) is suitable for theoretical analysis but impractical to solve numerically. Furthermore, we are more likely to solve (4) directly because it involves Level 3 Basic Linear Algebra Subprograms (BLAS-3) operations, which have better performance on computers [15].

In fact, for $k = 0, 1, \cdots$, Equation (1) can also be written as the matrix form

$$\mathbf{U}^{(k+1)} = \mathbf{U}^{(k)} + (\mathbf{V} - B_1\mathbf{U}^{(k)}B_2^T), \tag{5}$$

which is known as the residual-updating iteration ([13]). We remark here that the iteration (5) is mathematically equivalent to the splitting iteration for solving (4), where $B_1 = I_n - (I_n - B_1)$ and $B_2 = I_m - (I_m - B_2)$ are the splittings of B_1 and B_2, respectively.

We end this section with some properties of the Kronecker product.

Lemma 1 ([16]): Let A be an $n \times n$ matrix, B be an $m \times m$ matrix, and let X be an $n \times m$ matrix. Then, we have the following properties of the Kronecker product:

(1) $\mathbf{vec}(AXB) = (B^T \otimes A)\mathbf{vec}(X)$;
(2) $A \otimes B$ is invertible if and only if both A and B are invertible, and $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$;
(3) If A, B, C, and D are matrices of such size that the matrix products AC and BD can be formed, then $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$.

III. Diagonally compensated splitting iteration

In this section, we derive the iterative method for solving Equation (4) after introducing the splitting for the collocation matrices B_1 and B_2.

First, let

$$B_1 = \widetilde{B}_{q_1} + R_{q_1},$$

where $\widetilde{B}_{q_1}$ is a band matrix with half-bandwidth q_1 $(0 \leq q_1 \leq n-1)$ and

$$R_{q_1} = \begin{pmatrix} 0 & \cdots & 0 & b^n_{q_1+1}(t_0) & \cdots & b^n_n(t_0) \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & b^n_n(t_{q_1+1}) \\ b^n_0(t_{q_1+1}) & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ b^n_0(t_n) & \cdots & b^n_{q_1+1}(t_n) & 0 & \cdots & 0 \end{pmatrix}.$$

Then, we define a diagonal matrix

$$D_1 = \text{diag}(d_0, d_1, \ldots, d_n),$$

where

$$d_i = \sum_{|i-j|>q_1} b^n_i(t_j), i = 0, 1, \ldots, n.$$

Let

$$M_{q_1} = \widetilde{B}_{q_1} + D_1. \tag{6}$$

Therefore, the collocation matrix B_1 can be split as

$$B_1 = M_{q_1} - (D_1 - R_{q_1}).$$

Similarly, the collocation matrix B_2 can be split as

$$B_2 = M_{q_2} - (D_2 - R_{q_2}),$$

where $M_{q_2} = \widetilde{B}_{q_2} + D_1$ and $\widetilde{B}_{q_2}$ being a band matrix with half-bandwidth q_2 $(0 \leq q_2 \leq m-1)$.

The method to split a matrix is the diagonally compensated reduction ([11]). In [11], Liu et al. constructed a class of preconditioners for PIA and exploited the PPIA format

$$\begin{aligned}\mathbf{vec}(\mathbf{U}^{(k+1)}) = (M_{q_2}^{-1} \otimes M_{q_1}^{-1})\mathbf{vec}(\mathbf{V}) +& \\ [I_{nm} - (M_{q_2}^{-1} \otimes M_{q_1}^{-1})(B_2 \otimes B_1)]&\mathbf{vec}(\mathbf{U}^{(k)}),\end{aligned} \tag{7}$$

where M_{q_1} and M_{q_2} are the preconditioners defined as in (6).

A. Diagonally compensated splitting iteration format

Having gotten the splittings of B_1 and B_2, according to (3), we can obtain the following iteration format

$$\mathbf{U}^{(k+1)} = \mathbf{U}^{(k)} + M_{q_1}^{-1}(\mathbf{V} - B_1\mathbf{U}^{(k)}B_2^T)M_{q_2}^{-1}. \tag{8}$$

We call the iteration (8) as the diagonally compensated splitting iteration (DCSI).

By Lemma 1 it is easy to verify that the iteration format (7) is the column-stacking form of (8). Thus, the iteration format (8) can be seen as an alternative to the PPIA for tensor product Bézier patches. Unlike the iteration (7), the iteration (8) contains only BLAS-3 operations. Therefore, the latter would perform better than the former regarding computational efficiency. Due to the equivalence between these two iterative formats, the convergence of (7) is the same as that of (8). In other words, the limit of $\mathbf{B}^{(k)}(x,y)$ interpolates the points $\{\mathbf{v}_{ij}\}_{i=0,\ldots,n}^{j=0,\ldots,m}$.

Obviously, the choice of the half-bandwidth q_1 and q_2 will affect the performance of the algorithm (8). If both q_1 and q_2 are set to 0, the iteration (8) will degenerate into the residual-updating iteration (5), and the convergence rate will slow down. If $q_1 = n-1$ and $q_2 = m-1$, the iteration (8) will be equivalent to the direct solver for solving the matrix equation (4) and the computational complexity will increase. The analytical determination of the optimal half-bandwidth q_1 and q_2 is confusing. Due to the equivalence between the iterations (7) and (8), we adopt the same strategy used in [2] to find a balance between the convergence rate and the computational complexity.

IV. Numerical results

In this section, we present several numerical experiments to evaluate the computational efficiency of the proposed method. The experiments were conducted using Matlab R2016a on a PC with an AMD Ryzen 7 4800U CPU and 16 GB RAM.

We utilized

$$\varepsilon^{(k)} = \max_{0\leq i\leq n, 0\leq j\leq m} \|\mathbf{v}_{ij} - \mathbf{B}^{(k)}(x_i, y_j)\| \tag{9}$$

to measure the interpolation error of the k-th approximate interpolation surface $\mathbf{B}^{(k)}(x,y)$.

In our experiments, we utilized the PPIA and the DCSI to iteratively interpolate the points given in Examples 1 – 4 and utilized the PIA, the PPIA, and the DCSI to iteratively interpolate the points given in Example 5. We took the half-bandwidth $q_1 = \lfloor (n+1)/2 \rfloor$ and $q_2 = \lfloor (m+1)/2 \rfloor$, where $\lfloor \cdot \rfloor$ is the round operation.

For simplicity, we introduce some notations in the tables below, "k" represents the number of iterations, "$\varepsilon^{(k)}$" represents the interpolation error, and "T" represents the runtime (in seconds).

Example 1: Consider data interpolation of 20 points: $(1,1,1)$, $(1,2,4)$, $(1,3,2)$, $(1,4,2)$, $(1,5,2)$, $(2,1,2)$, $(2,2,2)$, $(2,3,3)$, $(2,4,6)$, $(2,5,4)$, $(3,1,3)$, $(3,2,2)$, $(3,3,4)$, $(3,4,4)$, $(3,5,3)$, $(4,1,4)$, $(4,2,6)$, $(4,3,1)$, $(4,4,1)$, $(4,5,2)$.

Example 2: Consider data interpolation of 21×21 points:

$$\left(\frac{i}{20}, \frac{j}{20}, f\left(\frac{i}{20}, \frac{j}{20}\right)\right), i = 0, 1, \ldots, 20; j = 0, 1, \ldots, 20,$$

where

$$f(x,y) = \frac{3}{4}e^{-\frac{(9x-2)^2+(9y-2)^2}{4}} + \frac{3}{4}e^{-\frac{(9x+1)^2}{49} - \frac{9y+1}{10}} + \frac{1}{2}e^{-\frac{(9x-7)^2+(9y-3)^2}{4}} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2}.$$

Example 3: Consider data interpolation of 25×26 points:

$$\left(-8 + \frac{2i}{3}, -8 + \frac{16j}{25}, g\left(-8 + \frac{2i}{3}, -8 + \frac{16j}{25}\right)\right), \quad i = 0, 1, \ldots, 24; j = 0, 1, \ldots, 25,$$

where $g(x,y) = \sin(\sqrt{x^2+y^2})/\sqrt{x^2+y^2}$.

Example 4: Consider data interpolation of 29×29 points sampled uniformly from the shell surface

$$\begin{cases} x = \frac{1}{5}(1 - \frac{v}{2\pi})\cos(2v)(1+\cos u) + \frac{1}{10}\cos(2u) \\ y = \frac{1}{5}(1 - \frac{v}{2\pi})\sin(2v)(1+\cos u) + \frac{1}{10}\sin(2u) \\ z = \frac{v}{2\pi} + \frac{1}{5}(1 - \frac{v}{2\pi})\sin(2u) \end{cases},$$

where $0 \le u, v \le 2\pi$.

Example 5: Consider data interpolation of $(n+1) \times (m+1)$ points sampled uniformly from the function

$$z = \frac{\cos(10x(1+y^2))}{1+10(x+2y)^2},$$

where $0 \le x, y \le 1$.

TABLE I
COMPARISON OF PPIA WITH DCSI IN EXAMPLES 1 – 4.

Example	k	PPIA		DCSI	
		$\varepsilon^{(k)}$	T	$\varepsilon^{(k)}$	T
Example 1	1	4.4137e-02	3.28e-03	4.4137e-02	3.19e-03
	2	7.7061e-04	4.50e-03	7.7061e-04	3.55e-03
	5	5.5664e-09	4.63e-03	5.5664e-09	3.70e-03
	8	4.6185e-14	4.76e-03	4.6185e-14	3.80e-03
Example 2	1	1.3105e-04	5.57e-03	1.3096e-04	3.65e-03
	5	4.3060e-05	8.96e-03	4.3060e-05	4.72e-03
	10	2.3587e-05	1.09e-02	2.3587e-05	5.56e-03
	20	8.2450e-06	1.45e-02	8.2450e-06	7.24e-03
	50	3.5525e-07	2.62e-02	3.5522e-07	1.20e-02
Example 3	1	1.2983e-02	5.14e-03	3.0047e-08	2.32e-03
	2	2.9035e-04	6.68e-03	1.6211e-10	2.69e-03
	5	1.7334e-09	1.08e-02	3.5238e-11	3.11e-03
	10	1.8234e-11	1.53e-02	1.4202e-11	3.59e-03
Example 4	1	3.7865e-02	6.24e-03	2.6816e-09	2.31e-03
	2	4.4019e-02	8.85e-03	1.0057e-10	2.78e-03
	5	4.0183e-02	1.14e-02	1.6311e-11	3.34e-03
	10	1.9669e-01	2.32e-02	8.1278e-12	3.38e-03

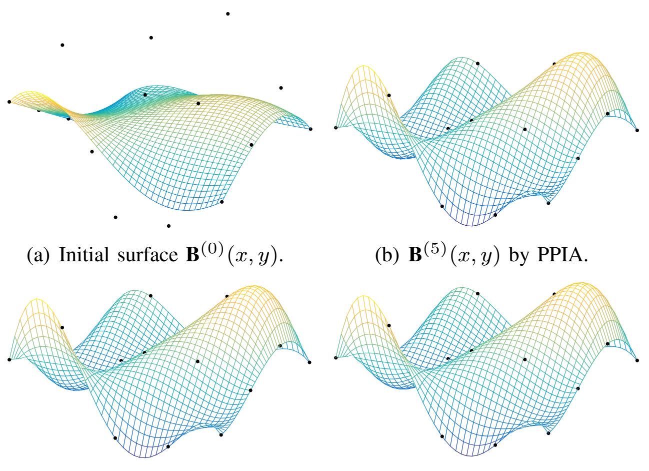

(a) Initial surface $\mathbf{B}^{(0)}(x,y)$.
(b) $\mathbf{B}^{(5)}(x,y)$ by PPIA.
(c) $\mathbf{B}^{(1)}(x,y)$ by DCSI.
(d) $\mathbf{B}^{(5)}(x,y)$ by DCSI.

Fig. 1. The points to be interpolated and the approximate interpolation surfaces obtained by PPIA and DCSI for Example 1.

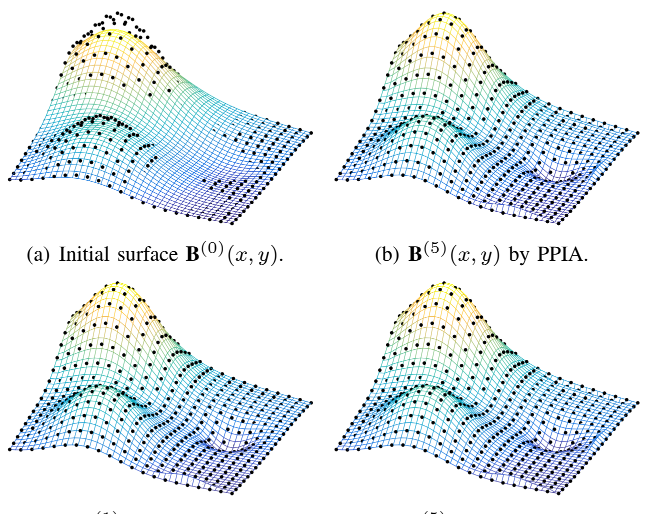

(a) Initial surface $\mathbf{B}^{(0)}(x,y)$.
(b) $\mathbf{B}^{(5)}(x,y)$ by PPIA.
(c) $\mathbf{B}^{(1)}(x,y)$ by DCSI.
(d) $\mathbf{B}^{(5)}(x,y)$ by DCSI.

Fig. 2. The points to be interpolated and the approximate interpolation surfaces obtained by PPIA and DCSI for Example 2.

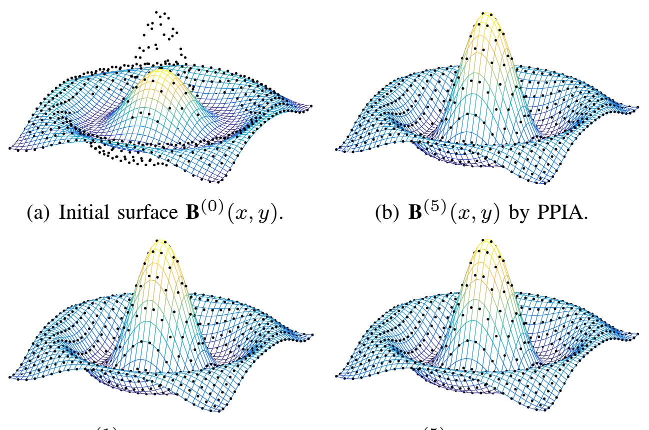
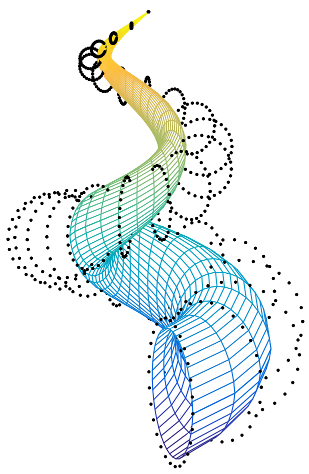
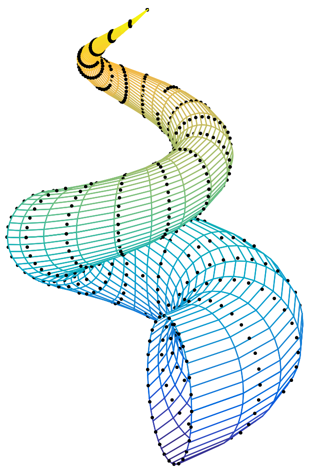

(a) Initial surface $\mathbf{B}^{(0)}(x,y)$.
(b) $\mathbf{B}^{(5)}(x,y)$ by PPIA.
(c) $\mathbf{B}^{(1)}(x,y)$ by DCSI.
(d) $\mathbf{B}^{(5)}(x,y)$ by DCSI.

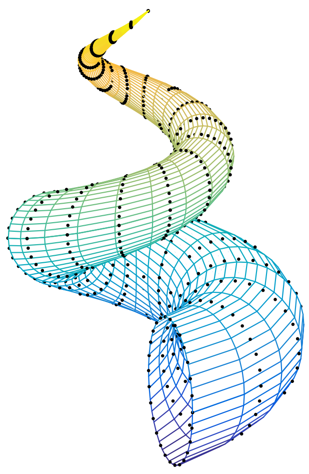
Fig. 3. The points to be interpolated and the approximate interpolation surfaces obtained by PPIA and DCSI for Example 3.



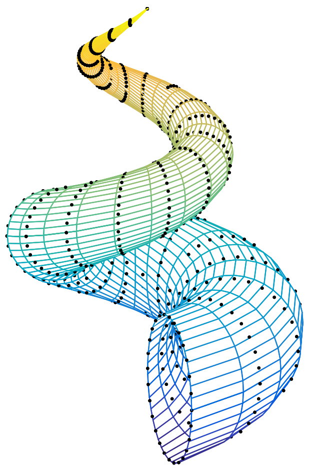
(a) Initial surface $\mathbf{B}^{(0)}(x,y)$.



(b) $\mathbf{B}^{(10)}(x,y)$ by PPIA.

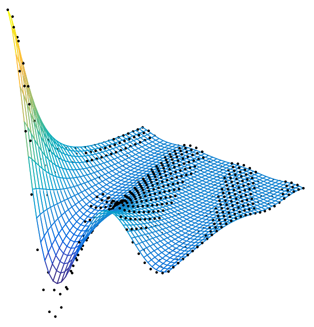


(c) $\mathbf{B}^{(1)}(x,y)$ by DCSI.

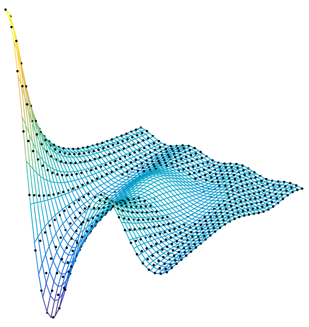


(d) $\mathbf{B}^{(10)}(x,y)$ by DCSI.

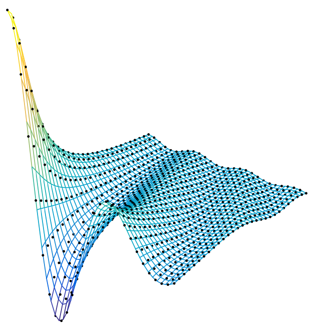
Fig. 4. The points to be interpolated and the approximate interpolation surfaces obtained by PPIA and DCSI for Example 4.



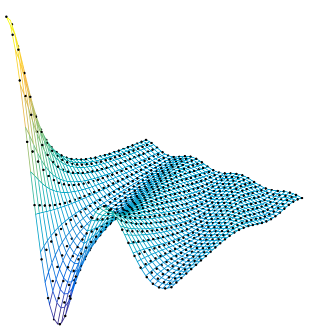
(a) Initial surface $\mathbf{B}^{(0)}(x,y)$.



(b) $\mathbf{B}^{(5)}(x,y)$ by PPIA.



(c) $\mathbf{B}^{(1)}(x,y)$ by DCSI.



(d) $\mathbf{B}^{(5)}(x,y)$ by DCSI.

Fig. 5. The points to be interpolated and the approximate interpolation surfaces obtained by PPIA and DCSI for Example 5.

TABLE II
INTERPOLATION ERROR OF PIA, PPIA, AND DCSI WITH DIFFERENT n AND m FOR EXAMPLE 5.

n	m	k	PIA	PPIA	DCSI
16	15	1	1.3047e-01	4.9748e-06	4.9748e-06
		2	7.6177e-02	1.2574e-06	1.2574e-06
		5	2.4709e-02	1.8611e-07	1.8611e-07
		10	8.9637e-03	2.4524e-08	2.4524e-08
20	20	1	9.7735e-02	1.1684e-06	2.3058e-07
		2	5.1139e-02	4.4987e-08	4.4987e-08
		5	1.3216e-02	1.9527e-08	1.9527e-08
		10	5.3223e-03	1.0413e-08	1.0413e-08
27	28	1	6.3552e-02	1.2354e-02	7.8685e-10
		2	2.7938e-02	9.1171e-03	2.7415e-10
		5	5.2011e-03	3.6081e-02	2.0242e-10
		10	2.3782e-03	2.3586e-02	1.5838e-10

In Table I, we list the interpolation errors and the runtime by implementing the PPIA and the DCSI to interpolate the points given in Examples 1 – 4. In Table II, we list the interpolation errors of the PIA, the PPIA, and the DCSI with different n and m for Examples 5.

We can see from Table I that, with the same number of iterations, the runtime of the DCSI is less than that of the PPIA. And the advantage becomes more apparent as the number of iterations or the size of the data increases. Moreover, it is evident from Tables I and II that for a small number of data points, the interpolation errors obtained by the DCSI are the same as those obtained by the PPIA, which verifies the equivalence of the DCSI and the PPIA. While in Examples 3 – 5, the interpolation errors obtained by the DCSI are smaller than those obtained by the PPIA, especially in the first few iterations. It should be pointed out that the interpolation error in Examples 4 and 5 increases sharply as we continue to increase the values of m and n; the interpolation error of the PIA is even smaller than that of the PPIA. This is due to the numerical instability caused by the increase in the condition number. As mentioned in Remark 1 and Reference [2], for large data, the system (2) is more ill-conditioned than (4), so the results of the PPIA may be less unstable and accurate.

In Figures 1 – 4, we show the approximate Bézier interpolation surfaces implemented by the PPIA and the DCSI. Figure 5 shows the approximate Bézier interpolation surfaces with $n = 27$ and $m = 28$ obtained by the PPIA and the DCSI. It is apparent that these surfaces obtained by the DSCI effectively interpolate the given data sets.

The numerical results indicate that our method is a competent replacement for PPIA and warrants further consideration for data interpolation.

V. CONCLUSIONS

Based on the diagonally compensated splitting, in this paper, we have proposed a splitting iteration for the collocation matrix equation, namely DCSI, to iteratively interpolate data sets. The proposed method is proved to be the matrix form of PPIA, thereby demonstrating its convergence. Additionally, the DCSI exclusively includes BLAS-3 operations, inducing superior computational efficiency compared to PPIA. The numerical results also show that the proposed DCSI outperforms the PPIA in terms of computational time and stability of the numerical algorithm.

References

[1] H. Lin, H. Bao and G. Wang, “Totally positive bases and progressive iterative approximation,” *Computers and Mathematics with Applications*, vol. 50, no. 3-4, pp. 575-586, 2005.

[2] C. Liu and Z. Liu and X. Han, “Preconditioned progressive iterative approximation for tensor product Bézier patches,” *Mathematics and Computers in Simulation*, vol. 185, pp. 372-383, 2021.

[3] H. Lin, T. Maekawa and C. Deng, “Survey on geometric iterative methods and their applications,” *Computer-Aided Design*, vol. 95, pp. 40-51, 2018.

[4] L. Lu, “Weighted progressive iterative approximation and convergence analysis,” *Computer Aided Geometric Design*, vol. 2, no. 2, pp. 129-137, 2010.

[5] J. Carnicer, J. Delgado and J. Peña, “Richardson method and totally nonnegative linear systems,” *Linear Algebra and its Applications*, vol. 433, no. 11-12, pp. 2010-2017, 2010.

[6] J. Carnicer, J. Delgado and J. Peña, “On the progressive iterative approximation property and alternative iterations,” *Computer Aided Geometric Design*, vol. 28, no. 9, pp. 523-526, 2011.

[7] C. Liu, J. Li and L. Hu, “Jacobi-PIA algorithm for bi-cubic B-spline interpolation surfaces,” *Graphic Models*, vol. 120, pp. 101134, 2022.

[8] L. Zhang, J. Tan, X. Ge and Z. Guo, “Generalized B-splines’ geometric iterative fitting method with mutually different weights,” *Journal of Computational and Applied Mathematics*, vol. 329, pp. 331-343, 2018.

[9] A. Ebrahimi and G.B. Loghmani, “A composite iterative procedure with fast convergence rate for the progressive-iteration approximation of curves,” *Journal of Computational and Applied Mathematics*, vol. 359, pp. 1-15, 2019.

[10] C. Liu, X. Han and J. Li, “Preconditioned progressive iterative approximation for triangular Bézier patches and its application,” *Journal of Computational and Applied Mathematics*, vol. 366, pp. 112389, 2020.

[11] C. Liu and Z. Liu, “Progressive iterative approximations with preconditioners,” *Mathematics*, vol. 9, no. 9, pp. 1503, 2020.

[12] Z. Tian, “The Jacobi and Gauss-Seidel-type iteration methods for the matrix equation AXB = C,” *Applied Mathematics and Computation*, vol. 292, pp. 63-75, 2017.

[13] Z. Liu, Z. Li, C. Ferreira and Y. Zhan, “Stationary splitting iterative methods for the matrix equation AXB = C,” *Applied Mathematics and Computation*, vol. 378, pp. 125195, 2020.

[14] M. K. Zak and F. Toutounian, “Nested splitting conjugate gradient method for matrix equation AXB = C and preconditioning,” *Computers and Mathematics with Applications*, vol. 66, no. 3, pp. 269-278, 2013.

[15] G. H. Golub and C. Van Loan, “Matrix Computations,” *Johns Hopkins University Press*, Baltimore and London, 1996.

[16] V. Simoncini, “Computational methods for linear matrix equations,” *SIAM Review*, vol. 58, no. 3, pp. 377-441, 2016.