Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. 12, No. 4, 2024, pp. 687-702
DOI:10.22034/cmde.2024.51747.2159

A generalized adaptive Monte Carlo algorithm based on a two-step iterative method for linear systems and its application to option pricing

Mahboubeh Aalaei
Insurance Research Center, Saadat Abad, Tehran, Iran.

Abstract

In this paper, we present a generalized adaptive Monte Carlo algorithm using the Diagonal and Off-Diagonal Splitting (DOS) iteration method to solve a system of linear algebraic equations (SLAE). The DOS method is a generalized iterative method with some known iterative methods such as Jacobi, Gauss-Seidel, and Successive Overrelaxation methods as its special cases. Monte Carlo algorithms usually use the Jacobi method to solve SLAE. In this paper, the DOS method is used instead of the Jacobi method which transforms the Monte Carlo algorithm into the generalized Monte Carlo algorithm. we establish theoretical results to justify the convergence of the algorithm. Finally, numerical experiments are discussed to illustrate the accuracy and efficiency of the theoretical results. Furthermore, the generalized algorithm is implemented to price options using the finite difference method. We compare the generalized algorithm with standard numerical and stochastic algorithms to show its efficiency.

Keywords. Adaptive Monte Carlo algorithm, Iterative methods, Finite difference method, Black Scholes model, Option pricing.
2010 Mathematics Subject Classification. 65C05, 65F10, 65N06.

1. Introduction

Within the past years, many researchers have developed Monte Carlo (MC) algorithms in different areas, specially in engineering and finance problems (see e.g. [4, 16, 17, 19, 25]). In real world problems, high dimensional systems of linear algebraic equations (SLAE) can be obtained directly or after the discretization of integral equations and partial differential equations [3, 5, 12]. It is well known that MC methods are more effective than direct and iterative methods for solving high dimensional sparse SLAE. In other words, even though conventional MC methods obtain less accurate solutions than direct or iterative numerical methods for high dimensional SLAE, they are more efficient, [24].
An important parameter of the algorithm efficiency is the computational complexity or the time taken by the algorithm. For instance, the time taken by direct methods such as the non-pivoting Gaussian elimination or Gauss-Jordan methods in [7] is $T_{\text{Direct}} = O(n^3)$ for solving SLAE

$$Bx = b, \quad B \in \mathbb{R}^{n\times n}, x, b \in \mathbb{R}^n, \tag{1.1}$$

While the time taken by the iterative methods, such as Jacobi, Gauss Seidel and relaxation techniques in [23] is

$$T_{\text{Iterative}} = O(n^2 k),$$

for k iterations. Also, the time taken by the MC algorithms in [8] to calculate the solution vector is

$$T_{\text{Monte Carlo}} = O(nkN),$$

where k and N are the length and the number of random paths respectively. Therefore, the computational complexity of the MC method is linear with the size of the matrix. Thus, we have clearly

$$\lim_{n\to\infty} \frac{\text{Complexity of Monte Carlo methods}}{\text{Complexity of Iterative methods}} = 0,$$

Received: 24 May 2022 ; Accepted: 09 February 2024.
Corresponding author. Email: aalaei@irc.ac.ir; maaalaei@gmail.com.

Therefore, for high dimensional SLAE, the order of complexity for the MC method is asymptotically better than that of the iterative method. Furthermore, the MC algorithms have some other significant advantages. For example, without calculating the whole solution vector, they can approximate individual components of the solution, [15]. Also, because of using many independent sample paths to estimate the solution, they have a good ability to parallelize. These several advantages of the MC algorithms have motivated the author for study in this work.

However, despite all advantages, the conventional MC method converges slowly. Some research papers worked on this problem to improve the convergence of the MC method. Halton proposed adaptive MC methods (AMC) in [15], which improve the convergence exponentially. Two Monte Carlo algorithms for solving SLAE were discussed in [18], which both methods achieved geometric convergence. A new Monte Carlo algorithm based on Jacobi over-relaxation in conjunction with the iterative refinement technique is proposed for solving high dimensional SLAE in [12]which had high accuracy and desirable speed. In [11], a new AMC was proposed for a parallel solution of large SLAE with exponential convergence. Also, a new MC algorithm for linear algebra problems relying on a non-discounted sum of an absorbed random walk was proposed in [10]. Some improvements on the hybrid MC method for solving SLAE were presented in [13]. In addition, a new MC method is introduced to solve the real and complex fuzzy SLAE in [14].

In this paper, we present a generalized adaptive Monte Carlo algorithm using the DOS iteration method for solving SLAE. The rest of the paper is organized as follows: The conventional MC and AMC algorithms are described in section 2. The DOS algorithm and the convergence properties are briefly discussed in section 3.The generalized AMC algorithm and its convergence and properties are discussed in section 4. Numerical experiments are presented in section 5. To demonstrate the computational efficiency of the generalized algorithm, we compare our results to standard numerical and stochastic algorithms. Furthermore, we used the algorithm to solve the SLAE obtained from finite difference method for option pricing. Finally, our conclusions are given in section 6.

2. Monte Carlo algorithms

Assuming matrix B in SLAE (1.1) is a nonsingular matrix. We solve this SLAE using MC algorithms. Let $D = diag(B)$ is a diagonal matrix and I is an identity matrix. Introducing $A = \{A_{ij}\}_{i,j=1}^{n} = I - D^{-1}B$ and $f = D^{-1}b$, we have $x = Ax + f$. Under the assumption $max_i \sum_{j=1}^{n} |A_{ij}| < 1$, the Jacobi iterative method,

$$x^{(k+1)} = Ax^{(k)} + f, \tag{2.1}$$

converges. Also, under this assumption, the following MC algorithms converge where the independent random paths is simulated using initial distribution $p = (p_1, \cdots, p_n) \in \mathbb{R}^n$ and transition matrix $P \in \mathbb{R}^{n\times n}$. while we consider the nonzero vector $h \in \mathbb{R}^n$ to evaluate the inner product $\langle h; x\rangle$ and the transition matrix P as follows:

$$P_{ij} = \frac{|A_{ij}|}{\sum_{j=1}^{n} |A_{ij}|}, i, j = 1, \ldots, n,$$

the following conditions also should be satisfied:

$$\begin{aligned} &p_{ij} \geq 0, \sum_{j=1}^{n} p_{ij} = 1, \text{if } a_{ij} \neq 0 \text{ then } p_{ij} \neq 0, \\ &p_i \geq 0, \sum_{i=1}^{n} p_i = 1, \text{if } h_i \neq 0 \text{ then } p_i \neq 0. \end{aligned} \tag{2.2}$$

2.1. **Conventional Monte Carlo algorithm.** The conventional MC method described in [23] expresses each component of the solution vector as the expectation of random variable. We generate Z random paths $i_0^{(s)} \to i_1^{(s)} \to \cdots \to i_k^{(s)}$

to estimate the inner product $\langle h, x^{(k+1)}\rangle$. Then θ_k via $\theta_k(h) = \frac{1}{Z}\sum_{s=1}^{Z}\eta_k^{(s)}(h)$ should be calculated, where

$$\eta_k^{(s)}(h) = \frac{h_{i_0^{(s)}}}{p_{i_0^{(s)}}}\sum_{m=0}^{k} w_m^{(s)} f_{i_m^{(s)}},$$

and

$$w_m^{(s)} = w_{m-1}^{(s)}\frac{A_{i_{m-1}^{(s)}i_m^{(s)}}}{P_{i_{m-1}^{(s)}i_m^{(s)}}}, w_0^{(s)} = 1.$$

2.2. **Adaptive Monte Carlo algorithm.** To consider AMC algorithm described in [15], assume $f^{(0)} = b, \theta_k^{(0)} = 0, f^{(d)} = f^{(d-1)} - B\theta_k^{(d-1)}$, for $d = 1, \cdots, r$. It should be noted that r is the number of stages and $\theta_k^{(d)}$ is the approximate solution of

$$B\Delta^d x = f^{(d)}, \tag{2.3}$$

obtained by conventional MC method. It is shown in [15] that the approximated solution of SLAE (1.1) can be calculated by

$$\varphi_k^{(d)}(h) = \varphi_k^{(d-1)}(h) + \theta_k^{(d)}(h).$$

Algorithm 1. Adaptive Monte Carlo (AMC).

1. Input matrix B, vector b, numbers N, k and r.
2. Set $f^{(0)} = b, \theta_k^{(0)} = 0, \varphi_k^{(0)} = 0$.
3. Compute vector $f = Db$, matrix $A = I - DB$ and $B_1 = I - A$ where $D = diag(\frac{1}{B_{11}}, \frac{1}{B_{22}}, \cdots, \frac{1}{B_{nn}})$
4. Compute transition probability matrix P.
5. **for** $t = 1$ to n
6. **for** $s = 1$ to Z
7. Generate random paths $t \to i_1^{(s)} \to \cdots \to i_k^{(s)}$ using P.
8. Set $w_0^{(t,s)} = 1$.
9. **for** $m = 1$ to k
10. Compute $w_m^{(t,s)} = w_{m-1}^{(t,s)}\frac{A_{i_{m-1}^{(s)}i_m^{(s)}}}{P_{i_{m-1}^{(s)}i_m^{(s)}}}$.
11. **end for**
12. **end for**
13. **end for**
14. **for** $d = 1$ to r
15. Compute $f^{(d)} = f^{(d-1)} - B_1\theta_k^{(d-1)}$.
16. **for** $t = 1$ to n
17. Compute $\theta_k^{(d)}(h) = \frac{1}{Z}\sum_{s=1}^{Z}\sum_{m=0}^{k} w_m^{(t,s)} f_{i_m^{(s)}}^{(d)}$.
18. **end for**
19. Compute $\varphi_k^{(d)} = \varphi_k^{(d-1)} + \theta_k^{(d)}$ where $\theta_k^{(d)} = \{\theta_k^{(d)}(h)\}_{t=1}^{n}$.
20. **end for**

As you can see from Eq. (2.1), the Jacobi method is often used for MC methods. In this article, we will use a generalized iteration method called DOS instead of Jacobi. The DOS iteration method has some known iterative methods such as Jacobi and Gauss-Seidel as its special cases.

3. The DOS iteration method

Consider we are going to solve the SLAE (1.1). Iterative methods for solving this SLAE require efficient splittings of the coefficient matrix B. The DOS iteration method presented in [9], which our proposed AMC algorithm is based

on, is considered the following splitting of B

$$B = D + L + U, \tag{3.1}$$

where $D = diag(B)$ is a diagonal matrix, L is a strictly lower triangular matrix, and U is a general matrix. Computing the following equations

$$\begin{cases} Dx^{(k+\frac{1}{2})} = [\omega_1 D + (\omega_1 - 1)L + (\omega_1 - 1)U]x^{(k)} + (1 - \omega_1)b, \\ (D + \omega_2 L)x^{(k+1)} = [(1 - \omega_2)D - \omega_2 U]x^{(k+\frac{1}{2})} + \omega_2 b. \end{cases} \tag{3.2}$$

then $x^{(k)}$ converges to vector x for an arbitrary initial guess $x^{(0)}$.Considering L and U as strictly lower and upper triangular matrices, respectively, the DOS iteration method has some iterative methods as its special case as follows:

- For $\omega_1 = 0$,, $\omega_2 = 0$, we have the Jacobi method,
- For $\omega_1 = 1$, $\omega_2 = 1$, we have the Gauss-Seidel method,
- For $\omega_1 = 1$, free ω_2, we have the Successive Overrelaxation method.

Eliminating of $x^{(k+\frac{1}{2})}$ from the second step of (3.2), we will have

$$x^{(k+1)} = M(\omega_1, \omega_2)x^{(k)} + G(\omega_1, \omega_2)b, k = 0, 1, 2, ..., \tag{3.3}$$

where

$$\begin{aligned} M(\omega_1, \omega_2) = (D + \omega_2 L)^{-1} & \\ \times [(1 - \omega_2)D - \omega_2 U]D^{-1} & \\ \times [\omega_1 D + (\omega_1 - 1)L + (\omega_1 - 1)U], & \end{aligned} \tag{3.4}$$

and

$$G(\omega_1, \omega_2) = (D + \omega_2 L)^{-1}[(1 - \omega_1)[(1 - \omega_2)D - \omega_2 U]D^{-1} + \omega_2 I]. \tag{3.5}$$

The following theorems which have been proved in [9] show that the DOS iteration method converges unconditionally for $0 \leq \omega_1 \leq 1$ and $0 < \omega_2 \leq 1$.

Theorem 3.1. *Let $B_{n\times n} = (B_{ij}) \in \mathbb{C}^{n\times n}$ be diagonally dominant and*

$$\sum_{j=2}^{n} B_{1j} < |B_{11}|.$$

If $L = (l_{ij})$ and $U = (u_{ij})$, and $l_{ij}u_{ij} \geq 0, 0 \leq \omega_1 \leq 1$ and $0 < \omega_2 \leq 1$, then for an arbitrary initial guess, the DOS method converges to the unique solution of SLAE (1.1).

Corollary 3.2. *If $B \in \mathbb{C}^{n\times n}$ is strictly diagonally dominant, then the DOS method converges for all $0 \leq \omega_1 \leq 1$ and $0 < \omega_2 \leq 1$ where L and U matrices should be choose suitable and satisfy in conditions of Theorem 3.1.*

4. Generalized adaptive Monte Carlo algorithm

We propose a generalized AMC algorithm for solving SLAE which it is represented in Algorithm 2. In fact, we implement the DOS iteration method instead of Jacobi method to modify AMC algorithm. Furthermore, instead of generating random paths in each stage, we use the same random paths for all stages. In the other words, we use the following transition matrix P for all stages:

$$P_{ij} = \frac{|M_{ij}|}{\sum_{j=1}^{n} |M_{ij}|}, i, j = 1, 2, \ldots, n.$$

Algorithm 2. Generalized Adaptive Monte Carlo (AMC + DOS).

1. Input matrix B, vector b, numbers N, k, r, ω_1 and ω_2.
2. Set $f^{(0)} = b, \theta_k^{(0)} = 0, \varphi_k^{(0)} = 0$.
3. Compute matrices M and G using Equations (3.4) and (3.5), the vector $e = Gb$ and the matrix $C = I - M$.

4. Compute transition probability matrix P.
5. **for** $t = 1$ to n
6. **for** $s = 1$ to Z
7. Generate random paths $t \to i_1^{(s)} \to \cdots \to i_k^{(s)}$ using P.
8. Set $w_0^{(t,s)} = 1$.
9. **for** $m = 1$ to k
10. Compute $w_m^{(t,s)} = w_{m-1}^{(t,s)} \frac{M_{i_{m-1}^{(s)} i_m^{(s)}}}{P_{i_{m-1}^{(s)} i_m^{(s)}}}$.
11. **end for**
12. **end for**
13. **end for**
14. **for** $d = 1$ to r
15. Compute $e^{(d)} = e^{(d-1)} - C\theta_k^{(d-1)}$.
16. **for** $t = 1$ to n
17. Compute $\theta_k^{(d)}(h) = \frac{1}{Z} \sum_{s=1}^{Z} \sum_{m=0}^{k} w_m^{(t,s)} e_{i_m^{(s)}}^{(d)}$.
18. **end for**
19. Compute $\varphi_k^{(d)} = \varphi_k^{(d-1)} + \theta_k^{(d)}$ where $\theta_k^{(d)} = \{\theta_k^{(d)}(h)\}_{t=1}^{n}$.
20. **end for**

Then the generalized algorithm needs Z random paths with length k to estimate each component of the solution vector and therefore nkZ random variables totally. Also, the AMC algorithm in [18] needs at least rnZ random variables, because it generates the same random paths for all components of the solution vector. Therefore, the comparison of the total number of random variables shows that if $k < r$, the generalized AMC algorithm needs less random variables than AMC algorithm in [18].
The difference between Algorithms 1 and 2 is that in Algorithm 2, we use iterative Equation (3.3)) instead of Equation (2.1). It should be noted that in algorithm 2, we use iterative Equation (3.3) instead of Equation (2.1) and this is the key point of difference between Algorithms 1 and 2. The convergence of the generalized AMC algorithm will be analyzed in the following subsection and the algorithm properties will be discussed in the numerical results.

4.1. **Convergence analysis.** First of all, we define the following notations:
Consider $e^{(0)} = Gb, \Delta^0 x = x$ and Eq. (3.3) for stage r as

$$C\Delta^r x = e^{(r)}, \tag{4.1}$$

where $\Delta^r x$ and $e^{(r)}$ are calculated by the following recursive equations

$$\Delta^r x = \Delta^{r-1} x - \Delta_k^{r-1} x,$$
$$e^{(r)} = e^{(r-1)} - C\Delta_k^{r-1} x,$$

and $\Delta_k^r x$ is the approximate solution of SLAE (4.1) which we obtain by using Eq. (3.3), k times. The following theorem will be proven.

Theorem 4.1. *Under the assumptions of Theorem 3.1, $\|M\| < 1$.*

Proof. The iteration matrix $M(\omega_1, \omega_2)$ is given by Eq. (3.4). Now, we assume

$$L_{\omega_1} = D^{-1}[\omega_1 D + (\omega_1 - 1)L + (\omega_1 - 1)U],$$

and

$$L_{\omega_2} = (D + \omega_2 L)^{-1}[(1 - \omega_2)D - \omega_2 U],$$

as defined in article [9]. It is proved in [9] that $\|L_{\omega_1}\|_\infty \leq 1$, because the matrix of A is diagonally dominant. Furthermore, using the condition $\sum_{j=2}^{n} a_{1j} < |a_{11}|$, we can prove that $\|L_{\omega_2}\|_\infty < 1$. Therefore, we have

$$\|M(\omega_1, \omega_2)\| \leq \|L_{\omega_1}\|_\infty \|L_{\omega_2}\|_\infty < 1.$$

□

Considering $S_0 = \Delta_k^0 x$ and $S_r = S_{r-1} + \Delta_k^r x$. Based on Theorem 4.1, $\|M\| < 1$ and clearly we have

$$x = S_r + \Delta^{r+1} x, \tag{4.2}$$

Theorem 4.2. *Under the assumptions of Theorem 3.1 and $\Delta_0^r x = 0$, $\lim_{r\to\infty} \Delta^r x = 0$.*

Proof. From Eq. (2.1) and (2.3), we have

$$\Delta^r x = M\Delta^r x + e^{(r)},$$
$$\Delta_k^r x = M\Delta_{k-1}^r x + e^{(r)}.$$

Then we can obtain

$$\Delta^r x = \Delta^{r-1} x - \Delta_{k-1}^{r-1} x = M(\Delta^{r-1} x - \Delta_{k-1}^{r-1} x) = \cdots = M^k \Delta^{r-1} x,$$

and

$$\Delta^r x = M^k \Delta^{r-1} x = M^{2k} \Delta^{r-2} x = M^{3k} \Delta^{r-3} x = \cdots = M^{(r-1)k} x.$$

Therefore

$$\|\Delta^r x\| \le \|M^{(r-1)k}\|.\|x\|. \tag{4.3}$$

We assume that matrix B is nonsingular. Therefore the SLAE has a unique and finite solution $\|x\|$. Furthermore, based on Theorem 4.1, $\|M\| < 1$. So, taking the limit of Eq. (4.3), the proof is completed. □

Theorem 4.3. *Under the assumptions of Theorem 3.1 and $\Delta_0^r x = 0$, S_r converges to x, geometrically, as r tends to infinity, [11].*

It can be concluded from Theorem 4.3 that the numerical method which the generalized adaptive Monte Carlo method is based on, is converged to the solution vector x.

Theorem 4.4. *As Z tends to infinity, $\theta_k^{(r)}$ converges to $\Delta_k^r x$.*

Proof. Since we define the random variable $\eta_k^{(r,s)}(h)$ along the path $t \to i_1^{(s)} \to \ldots \to i_k^{(s)}$, the expectation will be

$$E[\eta_k^{(r,s)}(h)] = \sum_{i_k^{(s)}}^{n} \cdots \sum_{t}^{n} \eta_t^{(r,s)}(h) P_{t i_1^{(s)}} \cdots P_{i_{k-1}^{(s)} i_k^{(s)}},$$

which, together with the formulas in Algorithm 2, gives

$$E[\eta_k^{(r,s)}(h)] = E[\sum_{m=t}^{k} w_m^{(s)} f_{i_m^{(s)}}^{(r)}] \tag{4.4}$$

$$= \sum_{i_0=t}^{n} \cdots \sum_{i_k^{(s)}=1}^{n} M_{t i_1^{(s)}} \ldots M_{i_{m-1}^{(s)} i_m^{(s)}} f_{i_m^{(s)}}^{(r)} P_{i_m^{(s)} i_{m+1}^{(s)}} \ldots P_{i_{k-1}^{(s)} i_k^{(s)}} \tag{4.5}$$

$$= \sum_{m=t}^{k} \sum_{i_1^{(s)}=1}^{n} \cdots \sum_{i_m^{(s)}=1}^{n} M_{i_0^{(s)} i_1^{(s)}} \ldots M_{i_{m-1}^{(s)} i_m^{(s)}} e_{i_m^{(s)}}^{(r)}. \tag{4.6}$$

We obtain the last equation using the property $\sum_{j=1}^{n} P_{ij} = 1$ and immediately obtain

$$E[\eta_k^{(r,s)}(h)] = \langle h, \sum_{m=0}^{k} M^m f^{(r)} \rangle = \langle h, \Delta_t^r x \rangle.$$

Therefore as Z tends to infinity, $\theta_k^{(r)} = \frac{1}{Z} \sum_{s=1}^{Z} \eta_k^{(r,s)}$ converges to $\Delta_k^r x$. □

Theorem 4.5. *As k and r tend to infinity, $\varphi_k^{(r)}$ converges to x.*

Proof. Let $S_r = \sum_{d=1}^{r} \Delta_k^d x$. From Equation (4.2) we have

$$x = \sum_{d=1}^{r} \Delta_k^d x + \Delta^{r+1} x.$$

We have $\lim_{Z\to\infty} \theta_k^{(d)} = \Delta_k^d x$, for $d = 1, \ldots, r$ from Theorem 4.4. Also from Theorem 4.2, as k and r tend to infinity, we can conclude that $\varphi_k^{(r)} = \sum_{d=1}^{r} \theta_k^{(d)}$ converges to x. □

5. NUMERICAL RESULTS

In this section, we report numerical results using generalized AMC algorithm to solve the SLAE and European put options. It should be noted that we consider $\theta = \frac{1}{2}$ for all examples. Also, the first stage of generalized AMC algorithm is CMC algorithm exactly. Therefore, we can consider the comparision of the results obtaind by these two algorithms in all examples.

5.1. **The solution of SLAE.** The solutions of linear equations is obtained using generalized AMC algorithm and the results is compared with [11].

Consider x is the exact solution of SLAE and $x^{(r)}$ is the approximate solution using the generalized AMC method at stage r. we will use the L_2 absolute estimate

$$\|x - x^{(r)}\| = (\sum_{i=1}^{n} (x_i - {x_i}^{(r)})^2)^{\frac{1}{2}}.$$

When the exact solution x is unknown, then the following formula will be used as the absolute error of estimation,

$$(\sum_{i=1}^{n} ({x_i}^{(r)} - \sum_{j=1}^{n} A_{ij} {x_j}^{(r)} - f_i)^2)^{\frac{1}{2}}.$$

Example 5.1. Consider linear system (1.1) where

$$B_{ij} = \begin{cases} 1 + 2q, & \text{if } j = i; \\ -q, & \text{if } j = i-1, i+1; \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Considering $b_i = \frac{i}{n}$, $q = 0.5$ and $n = 100, 300$, we implement DOS iteration method for different values of ω_1 and ω_2 and the results are shown via Table 1. Furthermore, we present a covergence comparison of the generalized AMC (AMC+DOS), AMC algorithm in [11] (AMC+Jacobi) and conjugate gradient method (CG) in the Figure 1 and 2. We assume the maximum allowable number of iterations 50, the relative tolerance for the residual error 10^{-20} and the starting point ${x_i}^0 = 10$ for CG method.

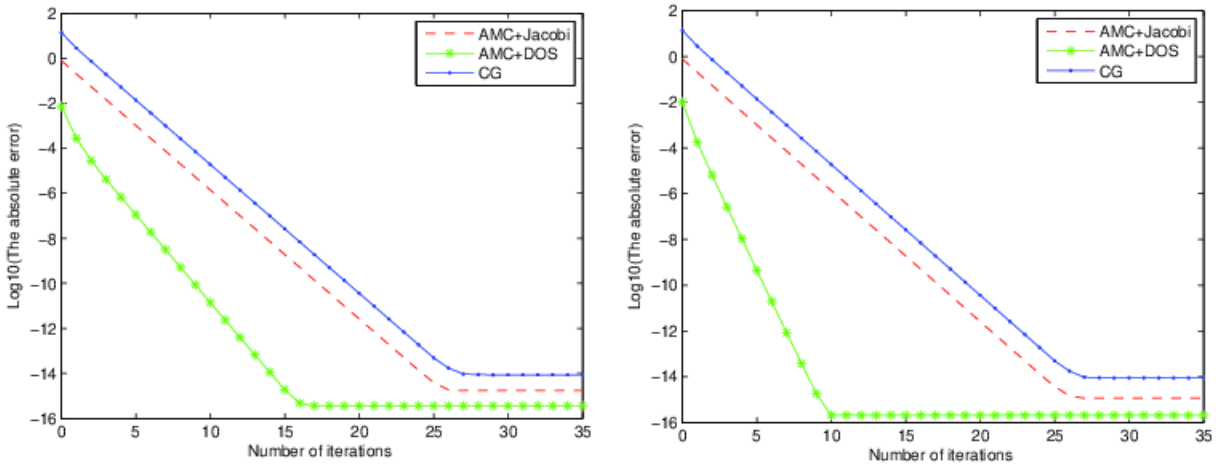
The matrix B is arised when the fully implicit finite difference scheme is used for discretization of the following parabolic equation with known initial and boundary conditions

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}.$$

The results show that for matrices with dimension $n = 100, 300$, the convergence of generalized AMC is much faster than CG and AMC algorithms; Since, it is clearly observable that the error of generalized AMC algorithm is significantly less than the error of CG and AMC algorithm for the same number of iterations. For $n = 100$ and $\omega_1, \omega_2 = 0.5$, after 10 iterations, the absolute error of generalized AMC algorithm is about 10^{-16}. At the same time, after 10 iterations, the absolute errors are about 10^{-5} and 10^{-6}, respectively. Therefore, the convergence of CG and AMC algorithms are very slower. In this case, we do expect that generalized AMC give better results because the norm of matrix A in Jacobi iteration from 0.2500 is reduced to 0.0040 using DOS iteration. It should be noted that the generalized AMC algorithm has an acceptable performance when the norm of matrix A is increased to 0.5525 in case $\omega_1, \omega_2 = 0.25$.

TABLE 1. Results of comparison for different ω_1, ω_2 and r for Example 5.1.

n	ω_1	ω_2	$\|M\|$	$\|A\|$	r	AMC+DOS	AMC+Jacobi	CG
100	0.25	0.25	0.5525	0.2500	5	1.4365×10^{-7}	7.2446×10^{-3}	5.1628×10^{-2}
					10	2.2654×10^{-12}	7.2446×10^{-6}	7.1143×10^{-5}
					35	3.6395×10^{-16}	1.6863×10^{-15}	8.6452×10^{-15}
	0.5	0.5	0.0040	0.2500	5	3.2479×10^{-10}	4.0578×10^{-3}	5.1628×10^{-2}
					10	3.4784×10^{-16}	5.8306×10^{-6}	7.1143×10^{-5}
					35	2.0444×10^{-16}	1.2331×10^{-15}	8.6452×10^{-15}
	0.75	0.75	0.0980	0.2500	5	1.6114×10^{-8}	3.9918×10^{-3}	5.1628×10^{-2}
					10	2.4401×10^{-15}	5.6511×10^{-6}	7.1143×10^{-5}
					35	2.7840×10^{-16}	1.2643×10^{-15}	8.6452×10^{-15}
300	0.25	0.25	0.5525	0.2500	5	1.5355×10^{-7}	7.0457×10^{-3}	5.2301×10^{-2}
					10	9.3167×10^{-12}	9.9960×10^{-6}	7.2216×10^{-5}
					35	6.4793×10^{-16}	2.1210×10^{-15}	2.2538×10^{-14}
	0.5	0.5	0.0040	0.2500	5	1.5771×10^{-9}	7.0145×10^{-3}	5.2301×10^{-2}
					10	4.8149×10^{-15}	9.9072×10^{-6}	7.2216×10^{-5}
					35	3.8862×10^{-16}	2.3269×10^{-15}	2.2538×10^{-14}
	0.75	0.75	0.0980	0.2500	5	7.3426×10^{-9}	6.9373×10^{-3}	5.2301×10^{-2}
					10	1.2781×10^{-15}	9.7014×10^{-6}	7.2216×10^{-5}
					35	4.4448×10^{-16}	2.4899×10^{-15}	2.2538×10^{-14}



(A) $\omega_1, \omega_2 = 0.25$.

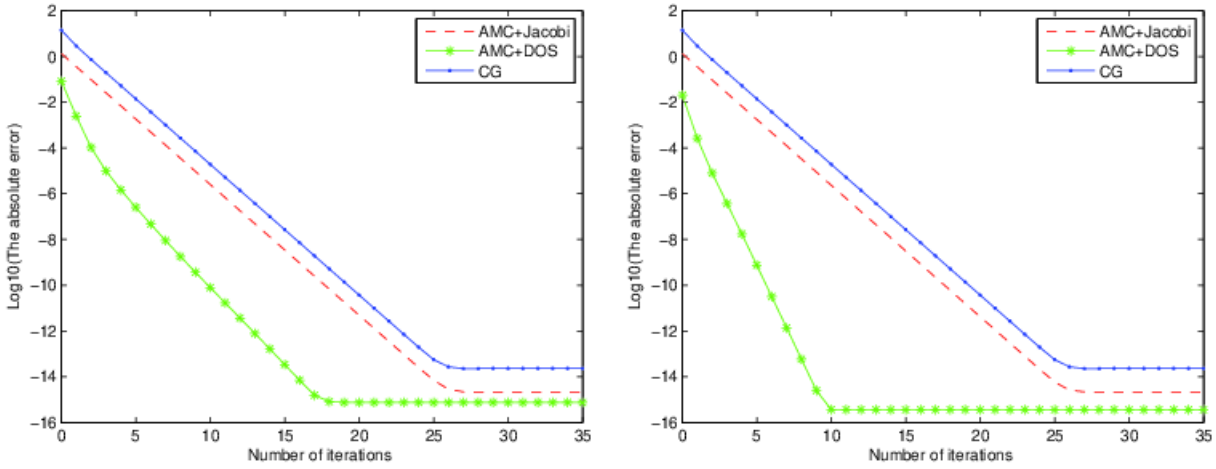
(B) $\omega_1, \omega_2 = 0.5$.

FIGURE 1. Some algorithms for solving Example 5.1 with matrix size 100×100.

After 35 iterations, the absolute error of generalized AMC algorithm for $\omega_1, \omega_2 = 0.5$ is about 10^{-16}, while it is about 10^{-15} for both CG and AMC algorithms. The same analysis can be implied for the case $n = 300$.

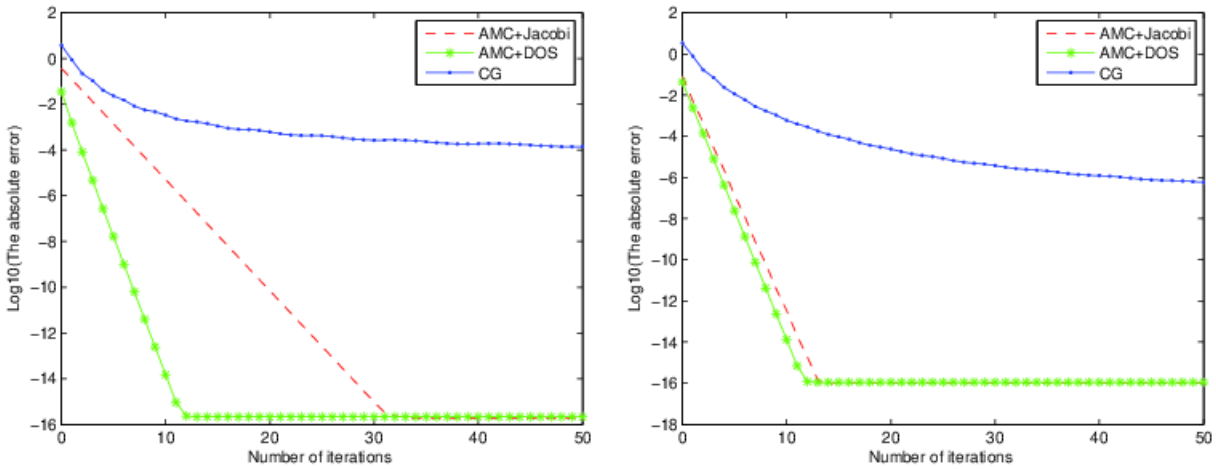
Example 5.2. Consider a dense diagonally dominant SLAE with random components

$$B_{ij} = \begin{cases} \rho_{ij}, & \text{if } i \neq j, \\ \sum_{j=1, j\neq i}^{n} \rho_{ij} + 2nr_i, & \text{if } i = j. \end{cases}$$



(A) $\omega_1, \omega_2 = 0.25$. (B) $\omega_1, \omega_2 = 0.5$.

FIGURE 2. Some algorithms for solving Example 5.1 with matrix size 300×300.



(A) $\omega_1, \omega_2 = 0.25$. (B) $\omega_1, \omega_2 = 0.5$.

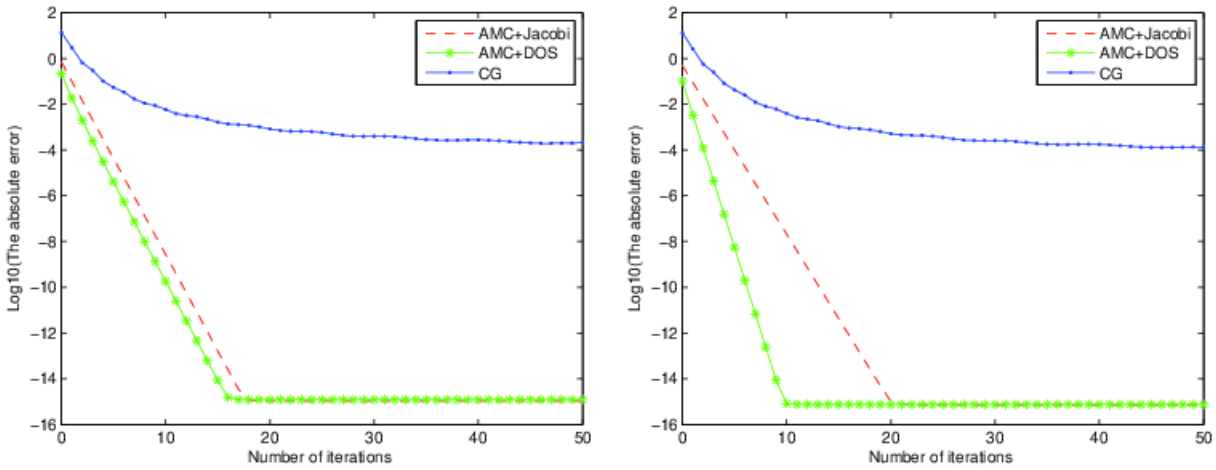
FIGURE 3. Some algorithms for solving Example 5.2 with matrix size 10×10.

where ρ_{ij} and r_i are random numbers uniformly distributed in $(0, 1)$ and $x_i = \frac{i}{n}$, [2].

We consider $k = 10, N = 100$. We calculate the errors for matrices with dimensions $n = 10, 100$ using the generalized AMC algorithm with different parameters ω_1 and ω_2 and compare the results with CG and AMC methods. The results are reported in Table 2 and on Figures 3 and 4. It is observable that the convergence of generalized AMC is much faster than CG and AMC in [11] for the smaller number of iterations. For matrix with dimension $n = 10$, we use $\omega_1, \omega_2 = 0.25$. After 50 iterations, the errors of generalized AMC and AMC algorithms are about 10^{-16}. While the error of CG mehtod is about 10^{-4}.

TABLE 2. Results of comparison for different ω_1, ω_2 and r for Example 5.2.

n	ω_1	ω_2	$\|M\|$	$\|A\|$	r	AMC+DOS	AMC+Jacobi	CG
10	0.25	0.25	0.5573	0.2961	5	7.2457×10^{-7}	6.6624×10^{-3}	5.2576×10^{-2}
					10	8.4751×10^{-15}	6.9147×10^{-5}	3.4578×10^{-2}
					50	2.1138×10^{-16}	1.7987×10^{-16}	1.3864×10^{-4}
	0.5	0.5	0.0026	0.1933	5	4.8572×10^{-7}	5.5142×10^{-6}	3.4657×10^{-2}
					10	9.2134×10^{-13}	2.1475×10^{-13}	1.8743×10^{-3}
					50	1.1102×10^{-16}	9.2315×10^{-17}	5.7105×10^{-7}
	0.75	0.75	0.1560	0.8759	5	8.2471×10^{-8}	2.5468×10^{-3}	1.1824×10^{-1}
					10	7.3228×10^{-16}	7.7762×10^{-5}	7.8836×10^{-2}
					50	2.3232×10^{-16}	1.4555×10^{-16}	1.1871×10^{-3}
100	0.25	0.25	0.6915	0.9935	5	5.3478×10^{-5}	7.2415×10^{-4}	5.2301×10^{-1}
					10	1.0241×10^{-10}	4.3874×10^{-9}	8.1495×10^{-2}
					50	1.1915×10^{-15}	1.0378×10^{-15}	1.9386×10^{-4}
	0.5	0.5	0.0037	0.3127	5	3.1474×10^{-9}	1.1469×10^{-5}	5.3641×10^{-2}
					10	3.6472×10^{-15}	5.3954×10^{-8}	4.3357×10^{-3}
					50	7.4153×10^{-16}	7.1191×10^{-16}	1.2737×10^{-4}
	0.75	0.75	0.1207	0.5275	5	3.3426×10^{-10}	5.1575×10^{-5}	2.6642×10^{-2}
					10	5.4725×10^{-15}	3.8892×10^{-8}	1.1354×10^{-3}
					50	7.1264×10^{-16}	8.2286×10^{-16}	5.7543×10^{-6}



(A) $\omega_1, \omega_2 = 0.25$. (B) $\omega_1, \omega_2 = 0.5$.

FIGURE 4. Some algorithms for solving Example 5.2 with matrix size 100×100.

Example 5.3. A linear system with

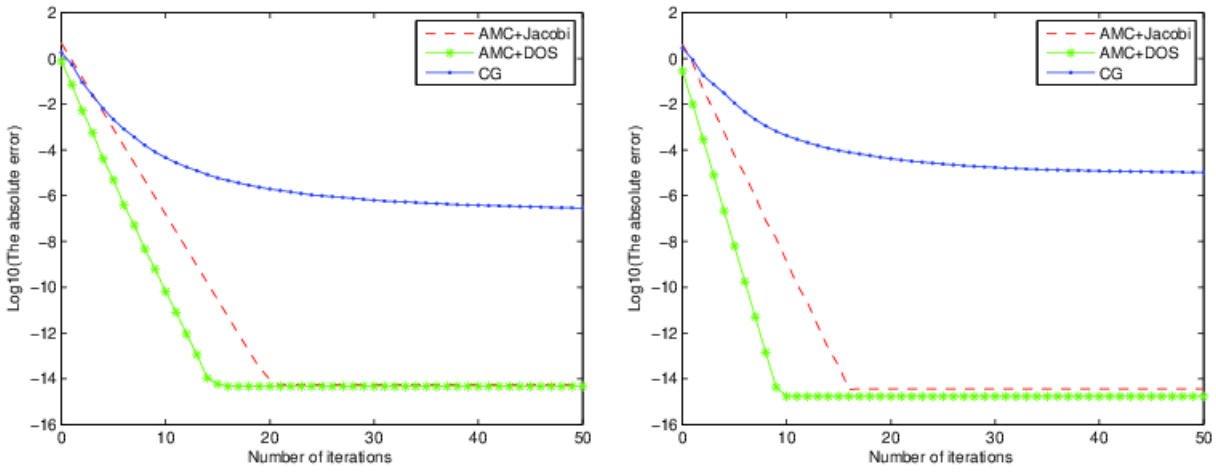
$$A_{ij} = \frac{\rho_{ij} r_i}{\sum_{k=1}^{n} \rho_{ik}},$$

is considered where $r_i = c + \rho_i(d - c)$, $c = min_i \sum_{j=1}^{n} A_{ij}$and $d = max_i \sum_{j=1}^{n} A_{ij} = \|A\|$. Furthermore ρ_i and ρ_{ij} are pseudo-random numbers uniformly distributed in (0, 1) and $F_i = i$, [18]. Let c = 0.2, d = 0.75, $k = 10$ and $N = 20$.

We present the results in Table 3 and on Figure 5 and 6 which show the convergence of CG and AMC algorithms are slower than generalized AMC. In the other words, For the same number of iterations, the error of generalized

TABLE 3. Results of comparison for different ω_1, ω_2 and r for Example 5.3.

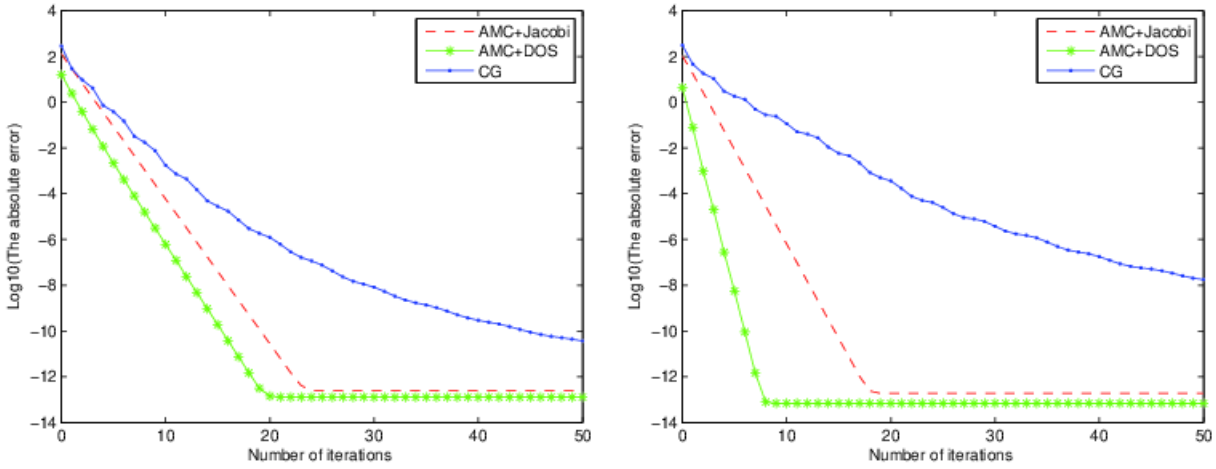
n	ω_1	ω_2	$\|M\|$	$\|A\|$	r	AMC+DOS	AMC+Jacobi	CG
10	0.25	0.25	0.6612	0.7586	5	5.3247×10^{-5}	3.5574×10^{-3}	7.1435×10^{-3}
					10	8.8877×10^{-11}	5.4428×10^{-7}	7.1425×10^{-5}
					50	4.5990×10^{-15}	5.1226×10^{-15}	2.8851×10^{-7}
	0.5	0.5	0.0071	0.3885	5	3.3685×10^{-7}	6.5465×10^{-4}	9.2546×10^{-3}
					10	3.5527×10^{-15}	5.9923×10^{-10}	7.1252×10^{-4}
					50	1.6616×10^{-15}	3.3083×10^{-15}	1.0401×10^{-5}
	0.75	0.75	0.0931	0.2150	5	7.2864×10^{-5}	2.4521×10^{-2}	1.1024×10^{-2}
					10	3.8866×10^{-11}	5.0065×10^{-5}	2.9341×10^{-3}
					50	8.1584×10^{-16}	5.4536×10^{-15}	3.9074×10^{-4}
100	0.25	0.25	0.4329	0.5956	5	3.8876×10^{-3}	9.3467×10^{-2}	1.1542×10^{-1}
					10	2.4672×10^{-7}	5.6743×10^{-5}	8.6478×10^{-3}
					50	1.2628×10^{-13}	2.5005×10^{-13}	3.6443×10^{-11}
	0.5	0.5	0.0059	0.5567	5	2.1872×10^{-7}	3.5461×10^{-2}	5.5371×10^{0}
					10	5.2487×10^{-15}	4.3589×10^{-7}	6.2574×10^{-1}
					50	6.8274×10^{-14}	1.8639×10^{-13}	1.7514×10^{-8}
	0.75	0.75	0.1506	0.7759	5	1.5455×10^{-5}	2.3576×10^{-2}	3.2156×10^{0}
					10	3.6624×10^{-11}	1.1769×10^{-5}	4.6474×10^{-2}
					50	9.0295×10^{-14}	1.7962×10^{-13}	5.0129×10^{-9}



(A) $\omega_1, \omega_2 = 0.25$. (B) $\omega_1, \omega_2 = 0.5$.

FIGURE 5. Some algorithms for solving Example 5.3 with matrix size 10×10.

AMC algorithm is significantly less than CG method and AMC algorithm in [11]. For example, for matrix dimension $n = 100$, in case $\omega_1, \omega_2 = 0.75$, after 50 iterations, the errors of generalized AMC, AMC and CG algorithms are about 10^{-14}, 10^{-13} and 10^{-9}, respectively.



(A) $\omega_1, \omega_2 = 0.25$. (B) $\omega_1, \omega_2 = 0.5$.

FIGURE 6. Some algorithms for solving Example 5.3 with matrix size 100×100.

5.2. **European option pricing.** The problem of European put option pricing can be described by the Black Scholes model [1, 11]:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r-q)S\frac{\partial V}{\partial S} - rV = 0, \tag{5.2}$$

with final condition $V(S,T) = max(E-S,0)$ and boundary conditions $V(0,t) = Ee^{-r(T-t)}$ and $V(S,t) \approx 0$ as $S \to \infty$, where S, E, T, r are the current price of asset, the strike price, the expiry time and the risk free interest rate, respectively. Also, S is assumed to behave $dS = (r-q)Sdt + \sigma S dW$, where dW is a Wiener process, r and σ are the drift rate and the volatility of the asset, respectively.

There is the closed form solution for the problem of European option pricing. But, we are unable to find the closed form solutions for more styles of options. To estimate the solution, we can use stochastic methods to price these options. In this regard, we can use the generalized AMC algorithm to value European options and check obtained formulas for these options.

We use the finite difference (FD) method with the parameter of discretization $\theta \in (0,1)$ to approximate the solution of (5.2). Considering $V_{ij} = V(i\Delta_S, j\Delta_t), 0 < i < N, 0 \le j \le M$, we can formulate the Black Scholes model as the following SLAE:

$$CV^{j+1} = DV^j + b^j, \tag{5.3}$$

where

$$C = \begin{bmatrix} 1-\theta m_1 & -\theta u_1 & 0 & \cdots & 0 \\ -\theta d_2 & 1-\theta m_2 & -\theta u_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\theta d_{N-1} & 1-\theta m_{N-1} \end{bmatrix},$$

TABLE 4. Camparisons of AMC algorithms for European Options.

S	Black Scholes	AMC in [11]	Error	AMC + DOS	Error
2	7.753099120	7.753099119	1.2699×10^{-9}	7.753099120	5.0534×10^{-11}
3	6.753099120	6.753099119	1.2370×10^{-9}	6.753099120	5.0635×10^{-11}
4	5.753099120	5.753099369	2.4941×10^{-7}	5.753099120	2.3317×10^{-11}
5	4.753099342	4.753102615	3.2730×10^{-6}	4.753099402	5.9078×10^{-8}
6	3.753180620	3.753302476	1.2185×10^{-4}	3.753185327	4.7073×10^{-6}
7	2.756835269	2.757692583	8.5731×10^{-4}	2.756871749	3.6479×10^{-5}
8	1.798714599	1.798755250	4.0650×10^{-5}	1.798710805	3.7938×10^{-6}

$$b^j = \begin{bmatrix} \theta d_1 V_{0j} + (1-\theta) d_1 V_{0j+1} \\ 0 \\ \vdots \\ 0 \\ \theta u_{N-1} V_{Nj} + (1-\theta) u_{N-1} V_{Nj+1} \end{bmatrix},$$

$$D = \begin{bmatrix} 1+(1-\theta)m_1 & (1-\theta)u_1 & 0 & \cdots & 0 \\ (1-\theta)d_2 & 1+(1-\theta)m_2 & (1-\theta)u_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & (1-\theta)d_{N-1} & 1+(1-\theta)m_{N-1} \end{bmatrix},$$

where $d_i = \frac{\Delta_t \sigma^2 S_i^2}{2\Delta_S^2} - \frac{\Delta_t (r-q) S_i}{2\Delta_S}, m_i = -\frac{\Delta_t \sigma^2 S_i{}^2}{\Delta_S^2} - \Delta_t r, u_i = \frac{\Delta_t{}^2 S_i{}^2}{2\Delta_S^2} + \frac{\Delta_t (r-q) S_i}{2\Delta_S}$. The generalized AMC algorithm is used to approximate the linear system obtained in each time-step and the result in the final step will be the price of the European option.

Example 5.4. Consider an European put option with $E = 10, T = 0.5, r = 0.05, \sigma = 0.2, q = 0, S_{min} = 0, S_{max} = 20, N = 200, M = 500$, where $k = 15, Z = 20$, $r = 20$, $\omega_1 = 0.5$ and $\omega_2 = 0.5$.

The results are presented in Table 4 which show the solution vector using the Black Scholes formula and generalized AMC algorithm and the difference between them as the absolute error. Furthermore, we have compared the generalized AMC algorithm results with AMC algorithm in [11], which demonstrate generalized AMC is better estimator than AMC.

5.3. **American option pricing.** For American option pricing, the SLAE (5.3) should be solved in each time step and the solution vector should be compared with the final condition and the result will be the solution vector in that time step. The solution vector will be calculated for $j = M, \ldots, 0$ and at the end, the solution vector will be the price of the American option. The SLAE (5.3) will be solved using the generalized AMC algorithm.

Example 5.5. Consider an American put option with $E = 100, T = 3, r = 0.08, \sigma = 0.2, S_{min} = 0, S_{max} = 2 \times S, N = 300, M = 100$, where $k = 15, Z = 20$, $r = 20$, $\omega_1 = 0.5$ and $\omega_2 = 0.5$.
The results are shown in Table 5 Which RMSE denotes the root mean squared absolute error and the true value is computed by the binomial tree method where the length of each time step is 0.0001 years, [6]. Furthermore, PAMC denotes the proposed algorithm in [2]. The results show that the generalized AMC is estimating the price of American options as well as new Monte Carlo algorithm proposed in [2].

TABLE 5. Results of comparison for American put option.

q	S	True	PAMC M=1000	AMC + DOS M=1000	PAMC M=2000	AMC + DOS M=2000
	80	20.000	20.000	20.000	20.000	20.000
	90	11.697	11.692	11.695	11.697	11.697
0.00	100	6.932	6.925	6.930	6.931	6.932
	110	4.155	4.151	4.153	4.155	4.155
	120	2.510	2.506	2.508	2.510	2.510
	80	20.350	20.347	20.347	20.350	20.350
	90	13.497	13.494	13.495	13.497	13.497
0.04	100	8.944	8.941	8.942	8.944	8.944
	110	5.912	5.909	5.910	5.912	5.912
	120	3.898	3.895	3.896	3.898	3.898
	80	22.205	22.200	22.202	22.205	22.205
	90	16.207	16.203	16.205	16.207	16.207
0.08	100	11.704	11.701	11.702	11.704	11.704
	110	8.367	8364	8.365	8.367	8.367
	120	5.930	5.927	5.928	5.930	5.930
	80	25.658	25.655	25.656	25.658	25.658
	90	20.083	20.079	20.080	20.083	20.083
0.12	100	15.498	15.496	15.497	15.498	15.498
	110	11.803	11.800	11.802	11.803	11.803
	120	8.886	8.882	8.883	8.886	8.886
RMSE			0.003	0.002	0.000	0.000

6. CONCLUSIONS

In this paper, we proposed a generalized adaptive Monte Carlo algorithm to solve SLAE. The convergence and efficiency of the generalized algorithm was discussed. The numerical results were presented for tridiagonal sparse matrices and random matrices. From numerical results, the convergence of the generalized adaptive Monte Carlo algorithm was faster than the CG method and Adaptive Monte Carlo algorithm presented in [11]. Furthermore, changing the matrix A in Jacobi method to matrix M in DOS iteration method often reduces the norm of this matrix and therefore results in faster convergence of generalized AMC algorithm than previous AMC algorithms and the CG method.

Furthermore, the generalized AMC algorithm was implemented to solve the option pricing problem. The aim of this article was to show how this algorithm works and how it is implemented to approximate financial problems. Even though the European options have closed form solution, we were going to approximate their solution to test our proposed algorithm. In this regard, parabolic partial differential equation discretized and sparse matrices obtained and solved using generalized AMC algorithm. Furthermore, we applied this algorithm to American options pricing which do not have a closed-form solution and compare to a new Monte Carlo algorithm. The results showed the efficiency and accuracy of the generalized AMC algorithm for solving option pricing problems.

Numerical results showed that the generalized algorithm is a stable and efficient way for valuing European and American options. Moreover, the results motivated us to implement the generalized AMC algorithm to several other financial models such as multi-asset options (see e.g. [21]) or implement it in conjunction with some new discretization techniques to price options (see e.g. [20]) in the forthcoming works.

We should note that the contribution of this research was the implementation of the DOS method instead of the Jacobi method which transforms the Adaptive Monte Carlo algorithm into the generalized Adaptive Monte Carlo algorithm. Therefore, this research will be the beginning of new researches about optimal generalized AMC algorithms by choosing

the best parameters and also using variance reduction techniques along with this generalized Monte Carlo algorithm. Therefore, this article will be the base of future researches in the theoretical and practical phases.

References

[1] M. Aalaei, *New Adaptive Monte Carlo algorithm to solve financial option pricing problems*, Journal of Data Science and Modeling, Journal of Data Science and Modeling, *1*(2) (2023), 139–151.

[2] M. Aalaei and M. Manteqipour, *An adaptive Monte Carlo algorithm for European and American options*, Computational Methods for Differential Equations, *10*(2) (2022), 489–501.

[3] A. R. Bacinello, *Regression-based algorithms for life insurance contracts with surrender guarantees*, Quantitative Finance, *10*(9) (2010), 1077–1090.

[4] A. R. Bacinello, P. Millossovic, and F. Viviano, *Monte Carlo Valuation of Future Annuity Contracts*, Mathematical and Statistical Methods for Actuarial Sciences and Finance, (2021), 57–62.

[5] D. Bauer, A. Kling, and J. Rub, *A universal pricing framework for guaranteed minimum benets in variable annuities*, ASTIN Bulletin, *38* (2008), 621–651.

[6] M. H. Chiang, H. H. Fu, Y. T. Huang, C. L. Lo, and P. T. Shih, *Analytical Approximations for American Options: The Binary Power Option Approach*, Journal of Financial Studies, *26*(3) (2018).

[7] J. J. Climent, C. Perea, L. Tortosa, and A. Zamora, *A BSP recursive divide and conquer algorithm to solve a tridiagonal linear system*, Applied Mathematics and Computation, *159* (2004), 459–484.

[8] J. J. Climent, L. Tortosa, and A. Zamora, *A note on the recursive decoupling method for solving tridiagonal linear systems*, Applied Mathematics and Computation, *140* (2003), 159–164.

[9] M. Dehghan, M. Dehghani-Madiseh, and M. Hajarian, *A two-step iterative method based on diagonal and off-diagonal splitting for solving linear systems*, Filomat, *31*(5) (2017), 1441–1452.

[10] I. Dimov, S. Maire, and J.M. Sellier, *A new walk on equations Monte Carlo method for solving systems of linear algebraic equations*, Applied Mathematical Modelling, *39* (2015), 4494–4510.

[11] R. Farnoosh and M. Aalaei, *New adaptive Monte Carlo algorithm for parallel solution of large linear systems with applications*, Proceedings of the Romanian Academy Series A, *16*(1) (2015), 11–19.

[12] R. Farnoosh, M. Aalaei, and M. Ebrahimi, *Combined probabilistic algorithm for solving high dimensional problems*, Stochastics An International Journal of Probability and Stochastic Processes, *87*(1) (2015), 30–47.

[13] B. Fathi-Vajargah and Z. Hassanzadeh, *Improvements on the hybrid Monte Carlo algorithms for matrix computations*, Sadhana, *44* (2018), 1–13.

[14] B. Fathi-Vajargah and Z. Hassanzadeh, *Monte Carlo method for the real and complex fuzzy system of linear algebraic equations*, Soft Computing, *24*(2) (2020), 1255–1270.

[15] J. Halton, *Sequential Monte Carlo*, Proceedings of the Cambridge Philosophical Society, *58* (1962), 57–78.

[16] C. H. Han and Y. Lai, *A smooth estimator for MC/QMC methods in finance*, Mathematics and Computers in simulation, *81*(3) (2010), 536–550.

[17] A. Jasra and P. D. Moral, *Sequential Monte Carlo methods for option pricing*, Stochastic analysis and applications, *29* (2011), 292–316.

[18] Y. Lai, *Adaptive Monte Carlo methods for matrix equations with applications*, Journal of Computational and Applied Mathematics, *231* (2009), 705–714.

[19] Y. Lai and J. Spanier, *Applications of Monte Carlo/Quasi-Monte Carlo methods in finance: option pricing*, Proceedings of a Conference at the Claremont Graduate University, Claremont, California, USA, June 22-26, 1998.

[20] H. Mesgarani, A. Adl, and Y. Esmaeelzade Aghdam, *Approximate price of the option under discretization by applying fractional quadratic interpolation*, Computational Methods for Differential Equations, In press.

[21] H. Mesgarani, S. Ahanj, and Y. Esmaeelzade Aghdam, *A novel local meshless scheme based on the radial basis function for pricing multi-asset options*, Computational Methods for Differential Equations, In press.

[22] F. Oliveira, C. S. Santos, F. A. Castro, and J.C. Alves, *A custom processor for TDMA solver on CFD application*, inARC 08, Berlin, Heidelberg, Springer-Verlag, (2008), 63–74.

[23] R. Y. Rubinstein, *Simulation and the Monte Carlo method*, Wiley, New York, 1981.

[24] C. J. K. Tan, *Solving systems of linear equations with relaxed Monte Carlo method*, The Journal of Supercomputing, *22* (2002), 113–123.
[25] C. Zhang, X. Wang, and Z. He, *Efficient Importance Sampling in Quasi-Monte Carlo Methods for Computational Finance*, SIAM Journal on Scientific Computing, *43*(1) (2021), B1-B29.