Pesquisa e Ensino em Ciências Exatas e da Natureza
06: e1773 (2022)
Research and Teaching in Exact and Natural Sciences

ARTICLE

ISNN 2526-8236 (edição online)
Pesquisa e Ensino em Ciências Exatas e da Natureza
©2022 UFCG / CFP / UACEN

The *m*-order Jacobi, Gauss–Seidel and symmetric Gauss–Seidel methods

Gustavo Benitez Alvarez[1,2], Diomar Cesar Lobão[1,2] & Welton Alves de Menezes[3]

(1) Master's Programme in Computational Modeling in Science and Technology, Federal Fluminense University, Av. dos Trabalhadores,420, Vila Santa Cecília, CEP 27255-125, Volta Redonda, RJ, Brazil. E-mail: benitez.gustavo@gmail.com

(2) Department of Exact Sciences, Federal Fluminense University, Brazil. E-mail: lobaodiomarcesar@yahoo.ca

(3) Department of Exact Sciences, Federal Fluminense University, Brazil. E-mail: wamenezes@id.uff.br

Alvarez G.B., Lobão D.C. & Menezes W.A (2022) The *m*-order Jacobi, Gauss–Seidel and symmetric Gauss–Seidel methods. *Pesquisa e Ensino em Ciências Exatas e da Natureza*, 06:e1773. http://dx.doi.org/10.29215/pecen.v6i0.1773

Academic editor: Paulo Xavier Pamplona. **Received:** 30 August 2021. **Accepted:** 29 January 2022. **Published:** 29 March 2022.

Metodos de ordem *m* de Jacobi e Gauss–Seidel e métodos simétricos de Gauss–Seidel

Resumo: Aqui, são desenvolvidos métodos de ordem *m* que conservam a forma dos métodos de primeira ordem. Métodos de ordem *m* têm uma taxa de convergência maior que sua versão de primeira ordem. Esses métodos de ordem *m* são subsequências de seu método precursor, onde alguns benefícios do uso de processadores vetoriais e paralelos podem ser explorados. Os resultados numéricos obtidos com as implementações vetoriais mostram vantagens computacionais quando comparadas às versões de primeira ordem.
Palavras-chave: Métodos iterativos de ordem *m*, Taxa media de convergência, Sistema linear de equações, Aceleradores.

Abstract: Here, *m*-order methods are developed that conserve the form of the first-order methods. The *m*-order methods have a higher rate of convergence than their first-order version. These *m*-order methods are subsequences of its precursor method, where some benefits of using vector and parallel processors can be explored. The numerical results obtained with vector implementations show computational advantages when compared to the first-order versions.
Key words: *m*-order iterative methods, Average rate of convergence, Linear system of equations, Accelerators.

Introduction

Iterative methods for solving linear systems emerged in the 19th century (Young, 1989; Saad, 2019). Since then, many researchers have contributed to the development, improvement and understanding of these methods until today. It would be impossible to mention all the articles and books dedicated to this topic, and in this article we would like to pay tribute to all these researchers. For example, a historical review with the state of the art in different decades can be found in Young (1989) and Saad (2019). These methods are used in several areas of science and technology, among which we highlight: mathematics, physics, chemistry, engineering and scientific computing. As an example, when a linear partial differential equation is solved by finite difference method, finite element method or finite volume method, the problem is transformed into a system of linear algebraic equations (Young, 1989).

Consider the square system of linear algebraic equations of order n and its equivalent defined as

$$Ax = b \text{ or } GAx = Gb, \tag{1}$$

where the matrix A and the vector b are known data for the problem. For simplicity, assume that the entries of all matrices and vectors are real numbers. If the matrix G is non-singular, each matrix G chosen determines an equivalent system where a different iterative method can be deduced. If the matrix A is non-singular, its inverse A^{-1} exists, and the matrix equation (1) has a unique solution $x = A^{-1}b$ or $x = [AG]^{-1}b$ (Forsythe and Moler, 1967; Varga, 2000; Golub and Van Loan, 2013). We are interested in solving equation (1) with iterative methods that can be represented in the form

$$x^{k+1} = Tx^k + d, \tag{2}$$

where $T = I - GA$ is called the iteration matrix, $d = Gb$ is a vector, and x^{k+1} denotes the approximate solution in the $k+1$ iteration (Forsythe and Moler, 1967; Varga, 2000; Young, 1971; Saad, 2003; Quarteroni et al., 2006). It is well known that the iterative process given by equation (2) is convergent if $\rho(T) < 1$, where $\rho(T)$ denotes the spectral radius of the iteration matrix T (Young, 1989; Varga, 2000; Saad, 2003; Quarteroni et al., 2006). The spectral radius of T is defined as the largest eigenvalue λ of T in module. That is, $\rho(T) = \max_{1\leq i\leq n}\{|\lambda_i|\}$.

The Jacobi, Gauss–Seidel and symmetric Gauss–Seidel methods are iterative methods described by the equation (2). These methods are still being researched as can be founded in the following references: Saad (2019), Varga (2000), Golub and Van Loan (2013), Saad (2003), Quarteroni et al. (2006), Bertaccini and Durastante (2018), Nägel et al. (2015), Antuono and Colicchio (2016), Bai and Miao (2017), Kong et al. (2019) and Mazza et al. (2019). The main goal of the present work is to increase the rate of convergence of these methods without modifying the form of a first-order method. This is done by generating the *m*-order versions of these methods. The *m*-order methods generate subsequences of their precursor method, where some benefits of using vector and parallel processors can be explored.

This paper is organized as follows. In next Section the Jacobi, Gauss–Seidel, symmetric Gauss–Seidel iterative methods and an alternative symmetric Gauss–Seidel will be presented. In the following Section are deduced *m*-order methods and a new method of type symmetric Gauss–Seidel. Later, the new methods are confronted with their precursors in numerical experiments. Finally, some conclusions are presented in the last Section.

The Jacobi, Gauss–Seidel and symmetric Gauss–Seidel methods

The matrix A of the equation (1) is splitted as $A = D + L + U$, where D is the diagonal matrix of the diagonal entries of A, L is the strict lower triangular matrix of A, and U is the strict upper triangular matrix of A. The well-known Jacobi method (J) is defined as

$$x^{k+1} = (I - D^{-1}A)x^k + D^{-1}b, \tag{3}$$

whose $T_J = I - D^{-1}A$. It is known that if A is diagonally dominant the method will be convergent (Forsythe and Moler, 1967; Varga, 2000; Young, 1971; Saad, 2003). This is a sufficient condition but not necessary.

The Gauss–Seidel method or forward Gauss–Seidel method (fGS) is defined as

$$x^{k+1} = -(D+L)^{-1}Ux^k + (D+L)^{-1}b, \tag{4}$$

whose $T_{fGS} = -(D+L)^{-1}U$ and $d_{fGS} = (D+L)^{-1}b$. Also, it is known that convergence is guaranteed if the matrix A is diagonally dominant. There is a method similar to the equation (4) known as backward Gauss–Seidel method (bGS) (Quarteroni et al., 2006; Bertaccini and Durastante, 2018). This method is defined by the equation (5).

$$x^{k+1} = -(D+U)^{-1}Lx^k + (D+U)^{-1}b, \tag{5}$$

whose $T_{bGS} = -(D+U)^{-1}L$ and $d_{bGS} = (D+U)^{-1}b$. Note that to obtain the solution in equation (4) the unknowns are ordered from 1 to n, and in equation (5) the unknowns are ordered from n to 1. Generally, $T_{fGS} \neq T_{bGS}$, and consequently the approximate solutions of both methods are different. Therefore, this motivated the development of a method that takes into account the two forms of ordering the unknowns. This method was called the symmetric Gauss–Seidel method (Quarteroni et al., 2006; Bertaccini and Durastante, 2018).

The well-known symmetric Gauss–Seidel method (sGS) is obtained by combining the forward and backward variants of the Gauss–Seidel method (Young, 1989; Quarteroni et al., 2006; Bertaccini and Durastante, 2018).

$$\text{forward} \qquad (D+L)x_f^{k+1/2} = -Ux_f^k + b, \tag{6}$$

$$\text{backward} \qquad (D+U)x_b^{k+1} = -Lx_b^{k+1/2} + b. \tag{7}$$

Frequently, the method is obtained if in the above equations $x_f^{k+1/2}$ and $x_b^{k+1/2}$ are eliminated assuming that $x_f^{k+1/2} = x_b^{k+1/2}$. Furthermore, it is assumed that $x^{k+1} = x_b^{k+1}$ and $x^k = x_f^k$ to obtain

$$\begin{aligned} x^{k+1} =& (D+U)^{-1}L(D+L)^{-1}Ux^k + (D+U)^{-1}[I - L(D+L)^{-1}]b \\ =& T_{bGS}T_{fGS}x^k + d_{bGS} + T_{bGS}d_{fGS}. \end{aligned} \tag{8}$$

On the other hand, starting from the alternative system (9-10) it is possible to deduce a new symmetric Gauss–Seidel method (nsGS).

$$\text{forward} \qquad (D+L)x_f^{k+1} = -Ux_f^{k+1/2} + b, \tag{9}$$

$$\text{backward} \qquad (D+U)x_b^{k+1/2} = -Lx_b^k + b. \tag{10}$$

Analogously it is possible to assume that $x_f^{k+1/2} = x_b^{k+1/2}$, and solving the system is obtained

$$\begin{aligned} x^{k+1} =& (D+L)^{-1}U(D+U)^{-1}Lx^k + (D+L)^{-1}[I-U(D+U)^{-1}]b \\ =& T_{fGS}T_{bGS}x^k + d_{fGS} + T_{fGS}d_{bGS}. \end{aligned} \tag{11}$$

Note that $T_{sGS} = (D+U)^{-1}L(D+L)^{-1}U$ and $T_{nsGS} = (D+L)^{-1}U(D+U)^{-1}L$. Furthermore, both methods have an equivalent computational cost. In general, $T_{sGS} \neq T_{nsGS}$ because the product of matrices is not commutative.

However, in general $x_f^{k+1/2} \neq x_b^{k+1/2}$, and this allows to deduce new methods of the symmetric type. In fact, in the two combinations of forward and backward Gauss–Seidel methods performed above, an intermediate step $x^{k+1/2}$ is introduced, which is later eliminated. For this reason, the rate of convergence of the methods generated with these combinations should be equal to or greater than each Gauss–Seidel separately. Based on this idea of combining forward and backward Gauss–Seidel methods, *m*-order methods can be deduced, which have a higher rate of convergence.

The *m*-order methods and new symmetric Gauss–Seidel methods

Consider a new way to combine forward and backward Gauss–Seidel methods like

$$\text{forward} \qquad (D+L)x_f^{k+1} = -Ux_f^{k+1/2} + b, \tag{12}$$

$$\text{forward} \qquad (D+L)x_f^{k+1/2} = -Ux_f^{k} + b, \tag{13}$$

$$\text{backward} \qquad (D+U)x_b^{k+1} = -Lx_b^{k+1/2} + b, \tag{14}$$

$$\text{backward} \qquad (D+U)x_b^{k+1/2} = -Lx_b^{k} + b. \tag{15}$$

Thus, three new methods can be obtained. The first method, described by equation (16), is obtained by eliminating $x_f^{k+1/2}$ in the equations (12) and (13). The second method, described by the equation (17), is obtained by eliminating $x_b^{k+1/2}$ in the equations (14) and (15).

$$\text{forward} \qquad x_f^{k+1} = [(D+L)^{-1}U]^2x_f^k + [-(D+L)^{-1}U + I](D+L)^{-1}b. \tag{16}$$

$$\text{backward} \qquad x_b^{k+1} = [(D+U)^{-1}L]^2x_b^k + [-(D+U)^{-1}L + I](D+U)^{-1}b. \tag{17}$$

These two new methods can be called 2-order forward and backward Gauss–Seidel, because their iteration matrices are defined by $T_{fGSo2} = [(D+L)^{-1}U]^2 = T_{fGS}T_{fGS}$ and $T_{bGSo2} = [(D+U)^{-1}L]^2 = T_{bGS}T_{bGS}$. Note that the number of steps which the current iteration depends on is one, as the contribution of the intermediate step is transferred to the iteration matrix and vector d. Thus, these methods retain the form of a first-order method, but with a rate of convergence greater than its precursor methods given equations (4) and

(5). The third method is obtained as a linear combination of the equations (16) and (17).

$$\begin{aligned} x^{k+1} =& \mu x_f^{k+1} + (1-\mu) x_b^{k+1} \\ =& \{\mu[(D+L)^{-1}U]^2 + (1-\mu)[(D+U)^{-1}L]^2\} x^k \\ &+ \{\mu[-(D+L)^{-1}U + I](D+L)^{-1} + (1-\mu)[-(D+U)^{-1}L + I](D+U)^{-1}\} b, \end{aligned} \tag{18}$$

where $x_f^k = x_b^k$ was assumed, and the free parameter $\mu \in [0,1]$.

Considering all of the above, it is possible to deduce another method such as

$$\begin{aligned} x^{k+1} =& \mu x_f^{k+1} + (1-\mu) x_b^{k+1} \\ =& -[\mu(D+L)^{-1}U + (1-\mu)(D+U)^{-1}L] x^k \\ &+ [\mu(D+L)^{-1} + (1-\mu)(D+U)^{-1}] b, \end{aligned} \tag{19}$$

where x_f^{k+1} and x_b^{k+1} are given by the equations (4) and (5) respectively. This new symmetrical method has as its particular case the methods determined by the equations (4) ($\mu = 1$) and (5) ($\mu = 0$). However, this method should not be called an 2-order method because its iteration matrix is not the product of two iteration matrices.

This procedure can also be used to obtain higher order Jacobi methods, and with this it is possible to increase the rate of convergence. Consider Jacobi method with *m* intermediate steps that will be eliminated

$$\begin{aligned} x^{k+1} =& (I - D^{-1}A) x^{k+(m-1)/m} + D^{-1}b \\ x^{k+(m-1)/m} =& (I - D^{-1}A) x^{k+(m-2)/m} + D^{-1}b \\ \vdots \quad =& \quad \vdots \\ x^{k+1/m} =& +(I - D^{-1}A) x^k + D^{-1}b. \end{aligned} \tag{20}$$

After eliminating all the intermediate steps, the *m*-order Jacobi method is obtained and given by

$$x^{k+1} = [I - D^{-1}A]^m x^k + \left[I + \sum_{l=1}^{m-1} (I - D^{-1}A)^{m-l}\right] D^{-1}b. \tag{21}$$

Note that *m*-order Jacobi methods have practical utility, since they have a higher rate of convergence than the Jacobi method and retain the property of being parallelizable.

In general, from a generic method defined by the equation (2) its *m*-order method is determined by

$$x^{k+1} = [T]^m x^k + \left[I + \sum_{l=1}^{m-1} [T]^{m-l}\right] d. \tag{22}$$

In the particular case of the Gauss-Seidel methods, we obtain its forward and backward *m*-order methods determined by the equations (23) and (24).

$$x^{k+1} = [-(D+L)^{-1}U]^m x^k + \left[I + \sum_{l=1}^{m-1} [-(D+L)^{-1}U]^{m-l}\right] (D+L)^{-1}b. \tag{23}$$

$$x^{k+1} = [-(D+U)^{-1}L]^m x^k + \left[I + \sum_{l=1}^{m-1} [-(D+U)^{-1}L]^{m-l}\right] (D+U)^{-1}b. \tag{24}$$

Even though the forward and backward Gauss–Seidel methods are not easily parallelizable, their m-order methods have some benefits for parallelization. This is because the iteration matrix of the m-order methods is the product of m matrices of the precursor method, and the matrix product can be parallelized.

In addition, m-order methods can be interpreted as methods that generate subsequences of their precursors. For this reason a faster convergence can be proved for the m-order methods if their precursors are convergent. Before, it is necessary to define the rate of convergence for the precursor iterative method described by the equation (2). Let $\varepsilon^k := x - x^k$ be the error vector after k iterations. Thus, $\varepsilon^k = T\varepsilon^{k-1} = \cdots = T^k\varepsilon^0$. Consider the average rate of convergence for k iterations defined by Kahan as $R_K := -\frac{\ln(\frac{\|\varepsilon^k\|}{\|\varepsilon^0\|})}{k}$ (Kahan, 1958), and by Varga as $R_V := -\frac{\ln(\|T^k\|)}{k}$ (Varga, 2000).

Theorem 1 *Let an iterative method defined by the equation (2) and its corresponding m-order method determined by the equation (22). Let x^0 be the same initial guess for both iterative methods. If the precursor method (2) is convergent with rate of convergence R_K and R_V, then the m-order method is convergent with rate of convergence $\bar{R}_K \geq (m-1)R_V + R_K$.*

Proof 1 *The proof uses the following theorem from mathematical analysis.* A sequence $\{x^k\}$ is convergent if, and only if, every its subsequences $\{x^{l(k)}\}$ are convergent *(Apostol, 1981). If x^0 is the same initial guess for both iterative methods, then the convergent precursor method defined by equation (2) generates the sequence $\{x^k\}$, and its m-order method defined by the equation (22) generates a subsequence $\{\bar{x}^l = x^{l(k)}\}$ such that $\bar{x}^l = x^k$ if $k = ml$ with $l = 1, 2, \ldots$. Therefore, the subsequence $\{\bar{x}^l\}$ is convergent, and represents an acceleration in the rate of convergence.*

Note that for the m-order method we have $\bar{\varepsilon}^k := x - \bar{x}^k$, $\bar{\varepsilon}^k = [T^m]^k\bar{\varepsilon}^0$. If $\bar{x}^0 = x^0$ it follows that $\bar{\varepsilon}^0 = \varepsilon^0$ and $\bar{\varepsilon}^k = [T^k]^{m-1}\varepsilon^k$. Therefore,

$$\|\bar{\varepsilon}^k\| \leq \|[T^k]^{m-1}\|\|\varepsilon^k\| \leq \|T^k\|^{m-1}\|\varepsilon^k\| \leq \|T\|^{k(m-1)}\|\varepsilon^k\|. \tag{25}$$

Analogously to the definition of Kahan it is possible to define $\bar{R}_K := -\frac{\ln(\frac{\|\bar{\varepsilon}^k\|}{\|\bar{\varepsilon}^0\|})}{k}$ for the m-order method. Hence, from inequality (25) follows

$$\bar{R}_K \geq -\frac{ln(\|[T^k]^{m-1}\|)}{k} + R_K \geq (m-1)R_V + R_K \geq -k(m-1)\frac{ln(\|T\|)}{k} + R_K, \tag{26}$$

which completes the proof. □

The previous theorem shows that the m-order method is always iteratively faster than its precursor method for k iterations. It must be said that this process of acceleration of convergence via subsequence may be more difficult to build in Krylov subspace methods, among which we highlight Conjugate Gradient (CG), Conjugate Gradient Squared (CGS), and Generalized Minimum Residual (GMRES). In addition, the m-order methods presented here are different from the so-called Chebyshev semi-iterative methods, second order Richardson method and second-degree methods (Golub and Varga, 1961; Young, 1972), but the possibility that there is some relationship between them should not be ruled out. Furthermore, all the ideas developed here can be applied to other methods such as successive overrelaxation method (SOR) and symmetrical successive overrelaxation method (SSOR).

On the other hand, it is possible to build a linear combination of different methods with a parameter μ such that $\mu \in [0,1]$ (Traub, 1977). For example, combining the Jacobi and Gauss–Seidel methods as

$$\begin{aligned} x^{k+1} =& \mu[(I-D^{-1}A)x^k + D^{-1}b] + (1-\mu)[-(D+L)^{-1}Ux^k + (D+L)^{-1}b] \\ =& [\mu(I-D^{-1}A) - (1-\mu)(D+L)^{-1}U]x^k + \mu D^{-1}b + (1-\mu)(D+L)^{-1}b, \end{aligned} \tag{27}$$

whose iteration matrix is $T_{JfGS} = \mu T_J + (1-\mu)T_{fGS}$. In addition, it is possible to generate new methods using the composition of different methods (Traub, 1977).

Numerical experiments

Seven matrices were chosen to perform numerical experiments and compare the performance of the methods. The first matrix $A_1 = A_{4\times4}$ is the well-known Hilbert matrix defined as $a_{i,j} = (i+j-1)^{-1}$ (Forsythe and Moler, 1967). In the next four experiments we use the matrices defined in Example 4.2 of Quarteroni et al. (2006), which are defined as

$$A_2 = \begin{bmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{bmatrix}, \quad A_3 = \begin{bmatrix} -3 & 3 & -6 \\ -4 & 7 & -8 \\ 5 & 7 & -9 \end{bmatrix},$$
$$A_4 = \begin{bmatrix} 4 & 1 & 1 \\ 2 & -9 & 0 \\ 0 & -8 & -6 \end{bmatrix}, \quad A_5 = \begin{bmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{bmatrix}. \tag{28}$$

Matrices $A_6 = A_{48\times48}$ and $A_7 = A_{153\times153}$ are sparse, and were obtained in the SuiteSparse Matrix Collection (Kolodziej et al., 2019). The matrix A_6 is the file ‘bcsstk01’ from BCSSTRUC1 set. The matrix A_7 is the file ‘bcsstk05’ from BCSSTRUC1 set. For all experiments the vector b was chosen such that the exact solution is $x_i = i \ \forall i = 1,\dots,n$, and the same initial guess was $x^0 = 0$. However, there are better initial guess for each method, which are determined by $x^0 = d$. This last choice allows to reduce the number of iterations necessary to verify the stopping criterion. The stopping criterion used was $\max_{1\le i\le n}\{|x_i^{k+1} - x_i^k|\} < 10^{-14}$.

The algorithms developed and implemented in this work make extensive use of vector structures, therefore not presenting explicit loops with the exception of the external repetition structure of the ‘while’ type used to execute the iterative convergence process. Thus, the computational performance of the Jacobi and Gauss–Seidel methods are optimized in the Matlab® language environment. With respect to the methods given by equations (21), (23) and (24) the sum of order m is implemented using a ‘for’ type repetition structure since in the simulations m was equal to 2 and 10. Therefore, it is good to remember that vectorized code does not necessarily mean faster code, due to the values of m it was decided to implement the sum in a serial way without loss of computational efficiency (Kepner, 2009). Furthermore, the implementation of the algorithm considered that matrices A_6 and A_7 are sparse. This allows you to save a significant amount of memory and reduce computation time.

In Tables 1, 2, 3, 4, 5, 6 and 7 the determinant, the condition number $\kappa(T) = \|T\|_2\|T^{-1}\|_2$, the spectral radius and the number of iterations for each method are presented. For matrix A_1 only the Jacobi methods are divergent. For matrix A_2 only the method defined by equation (19) is convergent. In the case of matrix A_3 only the forward Gauss–Seidel methods are divergent, and for the matrix A_5 only the backward Gauss–Seidel methods are divergent. The numerical results confirm the Theorem 1.

Table 1: Iteration matrix of each method for the linear system defined by the matrix A_1: determinant, condition number, spectral radius and number of iterations.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$	Iterations
J Eq. (3)	-1.5294270	8.4311823	2.5820911	divergent
2-o J Eq. (21)	2.3391472	29.409302	6.6671949	divergent
10-o J Eq. (21)	70.030586	3.417×10^6	13173.942	divergent
fGS Eq. (4)	0	∞	0.9990297	22002
2-o fGS Eq. (16)	0	∞	0.9980605	12002
10-o fGS Eq. (23)	0	∞	0.9903401	2853
bGS Eq. (5)	0	∞	0.9990297	22002
2-o bGS Eq. (17)	0	∞	0.9980605	13317
10-o bGS Eq. (24)	0	∞	0.9903401	2835
sGS Eq. (8)	0	∞	0.9985069	15002
nsGS Eq. (11)	0	∞	0.9985069	14002
npsGS $\mu=\frac{1}{2}$ Eq. (18)	-0.0297230	19.554683	0.9984568	12002
psGS $\mu=\frac{1}{2}$ Eq. (19)	-0.0984462	16.596875	0.9992367	26002

Table 2: Iteration matrix of each method for the linear system defined by the matrix A_2: determinant, condition number, spectral radius and number of iterations.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$	Iterations
J Eq. (3)	-1.1666666	4.0795444	1.1251473	divergent
2-o J Eq. (21)	1.3611111	4.8365021	1.2659565	divergent
10-o J Eq. (21)	4.6716241	14.294370	3.2515769	divergent
fGS Eq. (4)	0	∞	1.5833333	divergent
2-o fGS Eq. (16)	0	∞	2.5069444	divergent
10-o fGS Eq. (23)	0	∞	99.020142	divergent
bGS Eq. (5)	0	∞	1.0801234	divergent
2-o bGS Eq. (17)	0	∞	1.1666666	divergent
10-o bGS Eq. (24)	0	∞	2.1613940	divergent
sGS Eq. (8)	0	∞	1.5833333	divergent
nsGS Eq. (11)	0	∞	1.5833333	divergent
npsGS $\mu=\frac{1}{2}$ Eq. (18)	0.6959153	4.5638159	1.3980206	divergent
psGS $\mu=\frac{1}{2}$ Eq. (19)	-0.3767361	5.4447600	0.7842738	142

Table 3: Iteration matrix of each method for the linear system defined by the matrix A_3: determinant, condition number, spectral radius and number of iterations.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$	Iterations
J Eq. (3)	-0.2539682	28.1625119	0.8133091	167
2-o J Eq. (21)	0.0644998	37.9834491	0.6614717	18
10-o J Eq. (21)	1.1163×10^{-6}	604.18447	0.1266357	18
fGS Eq. (4)	0	∞	1.1111111	divergent
2-o fGS Eq. (16)	0	∞	1.2345679	divergent
10-o fGS Eq. (23)	0	∞	2.8679719	divergent
bGS Eq. (5)	0	∞	0.9428090	565
2-o bGS Eq. (17)	0	∞	0.8888888	288
10-o bGS Eq. (24)	0	∞	0.5549289	60
sGS Eq. (8)	0	∞	0.7126966	103
nsGS Eq. (11)	0	∞	0.7126966	101
npsGS $\mu=\frac{1}{2}$ Eq. (18)	0.1854533	12.8438001	0.8232698	165
psGS $\mu=\frac{1}{2}$ Eq. (19)	-0.2968002	6.8249624	0.6993380	96

Table 4: Iteration matrix of each method for the linear system defined by the matrix A_4: determinant, condition number, spectral radius and number of iterations.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$	Iterations
J Eq. (3)	0.0740740	6.1081965	0.4438188	45
2-o J Eq. (21)	0.0054869	6.4441038	0.1969751	7
10-o J Eq. (21)	4.973×10^{-12}	16.129493	0.0002965	7
fGS Eq. (4)	0	∞	0.0185185	12
2-o fGS Eq. (16)	0	∞	0.0003429	7
10-o fGS Eq. (23)	0	∞	0.0000000	3
bGS Eq. (5)	0	∞	0.3013571	30
2-o bGS Eq. (17)	0	∞	0.0908161	16
10-o bGS Eq. (24)	0	∞	0.0000061	5
sGS Eq. (8)	0	∞	0.0185185	11
nsGS Eq. (11)	0	∞	0.0185185	11
npsGS $\mu=\frac{1}{2}$ Eq. (18)	-1.246×10^{-5}	71.573476	0.0496594	13
psGS $\mu=\frac{1}{2}$ Eq. (19)	9.087×10^{-3}	5.5394076	0.2388210	25

Table 5: Iteration matrix of each method for the linear system defined by the matrix A_5: determinant, condition number, spectral radius and number of iterations.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$	Iterations
J Eq. (3)	-0.2142857	17.5108316	0.6411328	79
2-o J Eq. (21)	0.0459183	20.473569	0.4110512	10
10-oJ Eq. (21)	2.041×10^{-7}	79.631459	0.0117349	10
fGS Eq. (4)	0	∞	0.7745966	136
2-o fGS Eq. (16)	0	∞	0.6000000	70
10-o fGS Eq. (23)	0	∞	0.0777599	16
bGS Eq. (5)	0	∞	1.0923807	divergent
2-o bGS Eq. (17)	0	∞	1.1932958	divergent
10-o bGS Eq. (24)	0	∞	2.4195832	divergent
sGS Eq. (8)	0	∞	0.4535573	46
nsGS Eq. (11)	0	∞	0.4535573	46
npsGS $\mu=\frac{1}{2}$ Eq. (18)	0.0799683	6.1521714	0.7625609	115
psGS $\mu=\frac{1}{2}$ Eq. (19)	-0.1455420	10.015480	0.5892481	66

Table 6: Iteration matrix of each method for the linear system defined by the matrix A_6: determinant, condition number and spectral radius.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$
J Eq. (3)	3.844×10^{-18}	1.140×10^{5}	1.1014522
2-o J Eq. (21)	1.478×10^{-35}	2.149×10^{6}	1.2131969
10-o J Eq. (21)	6.958×10^{-175}	3.740×10^{11}	2.6281890
fGS Eq. (4)	0	∞	0.9969136
2-o fGS Eq. (16)	0	∞	0.9938367
10-o fGS Eq. (23)	0	∞	0.9695613
bGS Eq. (5)	0	∞	0.9969136
2-o bGS Eq. (17)	0	∞	0.9938367
10-o bGS Eq. (24)	0	∞	0.9695613
sGS Eq. (8)	0	∞	0.9968851
nsGS Eq. (11)	0	∞	0.9968851
npsGS $\mu=\frac{1}{2}$ Eq. (18)	3.367×10^{-69}	5.292×10^{5}	0.9946049
psGS $\mu=\frac{1}{2}$ Eq. (19)	-1.035×10^{-43}	1.980×10^{5}	0.9976792

Table 7: Iteration matrix of each method for the linear system defined by the matrix A_7: determinant, condition number and spectral radius.

Method	$det(T)$	$\kappa(T)$	$\rho(T)$
J Eq. (3)	2.311×10^{-51}	2.521×10^{3}	2.0149510
2-o J Eq. (21)	5.342×10^{-102}	1.760×10^{6}	4.0600279
10-o J Eq. (21)	0	∞	1103.1767
fGS Eq. (4)	0	∞	0.9985763
2-o fGS Eq. (16)	0	∞	0.9971546
10-o fGS Eq. (23)	0	∞	0.9858541
bGS Eq. (5)	0	∞	0.9985763
2-o bGS Eq. (17)	0	∞	0.9971546
10-o bGS Eq. (24)	0	∞	0.9858541
sGS Eq. (8)	0	∞	0.9977967
nsGS Eq. (11)	0	∞	0.9977967
npsGS $\mu=\frac{1}{2}$ Eq. (18)	-4.112×10^{-159}	6.055×10^{3}	0.9974169
psGS $\mu=\frac{1}{2}$ Eq. (19)	-3.445×10^{-97}	1.021×10^{4}	0.9987709

In the Tables 8 and 9 three times in seconds for each method are presented. The times t_T and t_d correspond to the time taken to calculate the iteration matrix T and the vector d respectively. Time t_W corresponds to the time spent in the iterative process of each method or ‘while’. This time was obtained considering the same initial guess $x^0 = 0$ for all methods. The total time $t_t = t_T + t_d + t_W$ should be used to compare the temporal performance of each method. This is because the larger m, the greater the time t_T and t_d, and the smaller the time t_W.

Table 8: Times to calculate the iteration matrix t_T, the vector t_d, total time t_t for A_6 and the number of iterations.

Method	t_T	t_d	t_t	Iterations
J Eq. (3)	4.0800×10^{-4}	3.4040×10^{-4}	—	divergent
2-o J Eq. (21)	4.3590×10^{-4}	6.9340×10^{-4}	—	divergent
10-o J Eq. (21)	5.4720×10^{-4}	29.6490×10^{-4}	—	divergent
fGS Eq. (4)	3.5250×10^{-4}	3.5250×10^{-4}	19.517732	9829
2-o fGS Eq. (16)	5.9830×10^{-4}	8.1300×10^{-4}	5.007476	5115
10-o fGS Eq. (23)	48.542×10^{-4}	433.543×10^{-4}	0.177706	1068
bGS Eq. (5)	3.0350×10^{-4}	3.5380×10^{-4}	17.243418	9140
2-o bGS Eq. (17)	5.9830×10^{-4}	8.7980×10^{-4}	3.950617	4703
10-o bGS Eq. (24)	17.686×10^{-4}	110.108×10^{-4}	0.157725	996
sGS Eq. (8)	6.1640×10^{-4}	6.1090×10^{-4}	18.314291	9564
nsGS Eq. (11)	5.8820×10^{-4}	7.4930×10^{-4}	16.978738	9225
npsGS $\mu=\frac{1}{2}$ Eq. (18)	12.462×10^{-4}	14.577×10^{-4}	6.007731	5685
psGS $\mu=\frac{1}{2}$ Eq. (19)	7.2730×10^{-4}	5.5010×10^{-4}	31.678465	12597

Table 9: Times to calculate the iteration matrix t_T, the vector t_d, total time t_t for A_7 and the number of iterations.

Method	t_T	t_d	t_t	Iterations
J Eq. (3)	9.3470×10^{-4}	3.1560×10^{-4}	—	divergent
2-o J Eq. (21)	18.3030×10^{-4}	7.5370×10^{-4}	—	divergent
10-o J Eq. (21)	35.0760×10^{-4}	371.2180×10^{-4}	—	divergent
fGS Eq. (4)	33.7520×10^{-4}	28.8670×10^{-4}	283.972996	21246
2-o fGS Eq. (16)	74.0830×10^{-4}	96.9050×10^{-4}	76.972961	11020
10-o fGS Eq. (23)	325.071×10^{-4}	1606.472×10^{-4}	10.532080	4002
bGS Eq. (5)	23.8040×10^{-4}	25.1510×10^{-4}	279.075221	21253
2-o bGS Eq. (17)	67.4790×10^{-4}	71.0550×10^{-4}	140.556805	15002
10-o bGS Eq. (24)	326.277×10^{-4}	1725.13×10^{-4}	4.338155	2528
sGS Eq. (8)	105.240×10^{-4}	89.1220×10^{-4}	122.439710	14011
nsGS Eq. (11)	95.7180×10^{-4}	79.4130×10^{-4}	122.298417	14002
npsGS $\mu = \frac{1}{2}$ Eq. (18)	155.998×10^{-4}	145.274×10^{-4}	88.492540	11817
psGS $\mu = \frac{1}{2}$ Eq. (19)	60.7230×10^{-4}	62.1690×10^{-4}	392.218996	24913

Conclusions

The *m*-order methods and new symmetric Gauss–Seidel methods are developed. The *m*-order methods have a higher rate of convergence than their first-order methods, but conserve the form of the first-order version. These *m*-order methods are subsequences of its precursor method, where some benefits of using vector and parallel processors can be explored. This is because the iteration matrix of the *m*-order methods is the product of *m* matrices of the precursor method. The numerical results obtained with vector implementations show computational advantages when compared to the first-order versions. The Theorem 1 shows that the *m*-order method is always iteratively faster than its precursor method for *k* iterations.

The well-known symmetric Gauss–Seidel method defined by equation (8) depends on the order used to combine the forward and backward variants of the Gauss–Seidel method, since its iteration matrix is obtained by composing these two variants. If the composition order was reversed, then a new symmetric Gauss–Seidel method defined by equation (11) is obtained. Furthermore, the computational implementation of these two symmetric methods requires the determination of the product of the iteration matrices of the two Gauss–Seidel variants.

On the other hand, the new symmetric Gauss–Seidel method defined by equation (19) does not depend on the order used to combine the forward and backward variants of the Gauss–Seidel method, since its iteration matrix is obtained with the linear combination of these two variants. In addition, it does not require the determination of the product of the iteration matrices of the two Gauss–Seidel variants, so its computational implementation is simpler than the methods defined by equations (8) and (11). This method defined by equation (19) proved to be interesting as it was the only method that converged in all the numerical experiments presented here. However, a better choice of the linear combination parameter μ should be further investigated. In all our numerical experiments we used $\mu = 1/2$. Finally, as far as the authors know, this is the first time that the methods described by equations (11), (18), (19), (21), (23) and (24) are presented.

Acknowledgements

This work was supported by the Universidade Federal Fluminense. This paper is dedicated to the memory of all researchers in the field of knowledge.

References

Antuono, M. and Colicchio, G. (2016). Delayed Over-Relaxation for iterative methods. *Journal of Computational Physics*, 321:892–907.

Apostol, T. M. (1981). *Mathematical analysis*. Addilson-Wesley Publishing Company.

Bai, Z.-Z. and Miao, C.-Q. (2017). On local quadratic convergence of inexact simplified Jacobi–Davidson method. *Linear Algebra and its Applications*, 520:215–241.

Bertaccini, D. and Durastante, F. (2018). *Iterative methods and preconditioning for large and sparse linear systems with applications*. Taylor & Francis Group.

Forsythe, G. E. and Moler, C. B. (1967). *Computer solution of linear algebraic systems*. Prentice-Hall, New Jersey.

Golub, G. H. and Van Loan, C. (2013). *Matrix computations*. The Johns Hopkins University Press, Baltimore, MD, 4th edition.

Golub, G. H. and Varga, R. S. (1961). Chebyshev semi-iterative methods, successive over-relaxation iterative methods, and second-order Richardson iterative methods, Parts I and II. *Numerische Mathematik*, 3:147–168.

Kahan, W. M. (1958). *Gauss–Seidel methods of solving large systems of linear equations*. PhD thesis, University of Toronto, Toronto.

Kepner, J. (2009). *Parallel MATLAB for Multicore and Multinode Computers*. Society for Industrial and Applied Mathematics, USA, 1st edition.

Kolodziej, S. P., Aznaveh, M., Bullock, M., David, J., Davis, T. A., Henderson, M., Hu, Y., and Sandstrom, R. (2019). The SuiteSparse Matrix Collection Website Interface. *Journal of Open Source Software*, 4(35):1244–1248.

Kong, Q., Jing, Y.-F., Huang, T.-Z., and An, H.-B. (2019). Acceleration of the Scheduled Relaxation Jacobi method: Promising strategies for solving large, sparse linear systems. *Journal of Computational Physics*, 397:108862.

Mazza, M., Manni, C., Ratnani, A., Serra-Capizzano, S., and Speleers, H. (2019). Isogeometric analysis for 2D and 3D curl–div problems: Spectral symbols and fast iterative solvers. *Computer Methods in Applied Mechanics and Engineering*, 344:970–997.

Nägel, A., Vogel, A., and Wittum, G. (2015). Evaluating linear and nonlinear solvers for density driven flow. *Computer Methods in Applied Mechanics and Engineering*, 292:3–15.

Quarteroni, A., Sacco, R., and Saleri, F. (2006). Iterative Methods for Solving Linear Systems. In *Numerical Mathematics. Texts in Applied Mathematics*, volume 37, chapter 4, pages 125–180. Springer, Berlin, Heidelberg.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

Saad, Y. (2019). Iterative methods for linear systems of equations: A brief historical journey.

Traub, J. F. (1977). *Iterative methods for the solution of equations*. Chelsea Publishing Company.

Varga, R. S. (2000). *Matrix iterative analysis*. Springer-Verlag Berlin Heidelberg.

Young, D. M. (1971). *Iterative solution of large linear systems*. Academic Press, Inc., Florida.

Young, D. M. (1972). Second-degree iterative methods for the solution of large linear systems. *Journal of Approximation Theory*, 5(2):137–148.

Young, D. M. (1989). A historical overview of iterative methods. *Computer Physics Communications*, 53(1):1–17.