

# SSOR and ASSOR preconditioners for Block–Broyden method <sup>☆</sup>

Geng Yang <sup>a,\*</sup>, Peng Jiang <sup>b</sup>

<sup>a</sup> College of Mathematics and Physics, Nanjing University of Posts and Telecommunications, Nanjing 210003, China

<sup>b</sup> College of Computer Science and Technology, Nanjing University of Posts and Telecommunications,  
P.O. Box 43, Nanjing 210003, China

---

## Abstract

Solving nonlinear equations is a problem that needs to be dealt with in the practical engineering application. This paper uses Block–Broyden method for solving large-scale nonlinear systems, and two preconditioners are applied for solving the underlying linear systems, including SSOR preconditioner as well as ASSOR method, which is based on SSOR. It discusses their implementation processes and compares the two algorithms from different aspects. Finally, it solves the nonlinear systems arising from the Bratu problem. Experimental results show that the preconditioning technique is effective for the Block–Broyden method and that the preconditioner ASSOR has better performance as a whole. Therefore, it can be used in the large-scale problems arising from scientific and engineering computing.

© 2006 Elsevier Inc. All rights reserved.

*Keywords:* Nonlinear equations; Block–Broyden method; SSOR preconditioners

---

## 1. Introduction

With the rapid development of mathematics and computer science, the research on solving nonlinear systems has gone largely noticed. In the past few years, a number of books entirely devoted to iterative methods for nonlinear systems have appeared [1–4]. The most common iterative methods include stationary methods such as Jacobi, Gause-Seide, SOR and nonstationary methods such as CG, MINRES, GMRES, BiCG and so on. However, these methods have two problems in common: One is that these algorithms usually need much storage and converge very slowly. The other is that they suffer from serious limitations, such as lack of good reliability. These problems have made the algorithms difficult to be applied to practical engineering computing. Therefore, much work has been done to overcome these limitations.

---

<sup>☆</sup> The work was supported by the Natural Science Foundation of Jiangsu Province under Grant Nos. BK2004218 and BK2003106, and also by the PanDeng Project of NUPT.

\* Corresponding author.

E-mail address: [yangg@njupt.edu.cn](mailto:yangg@njupt.edu.cn) (G. Yang).

Brown and Saad [5] proposed the nonlinear GMRES (m) method in 1990, which is a widespread algorithm at present. By storing a matrix with the dimension of  $(m + 1)$ , the algorithm only needs very little storage, whereas the nonlinear property of the question has made it difficult to devise parallel programs. A Block–Broyden (BB) algorithm was first proposed in 1997 with the proof of its local convergence [6]. The complexity, parallel performance and the storage requirement of the algorithm are discussed in [7]. It showed theoretically that this algorithm has advantages of effectiveness, high parallelism as well as low storage, and then discussed the practical use of the algorithm by applying it to the parallel machine SGI Power Challenge 12 CPU. The numerical results, in accordance with the theoretical analysis, demonstrated the unique advantage and practical prospect of the algorithm. However, as pointed out in paper [6,7], the iterative matrix in the Block–Broyden algorithm is a block diagonal matrix so that partial relevant information among the nodes is lost, affecting the convergence speed of the algorithm to some extent. Hence, seeking for proper preconditioning methods is one of the effective ways to solve this problem. Some preconditioners have been proposed and discussed in Refs. [8–10].

With the realization that preconditioning is essential for the successful use of iterative methods, research on preconditioners has moved to center stage in recent years. The first use of the term in connection with iterative methods was found by Evans in a paper on Chebyshev acceleration of SSOR in 1968. However, the concept of preconditioning as a way of improving the convergence of iterative methods is much older. As far back as 1845 Jacobi's method was known as an effective way to ensure the convergence of the simple iterative scheme. Preconditioning, as a means of reducing the condition number in order to improve convergence of an iterative process, seems to have been first considered by Cesari in 1937. A major breakthrough took place around the mid-1970s, with the introduction by Meijerink and van der Vorst of the incomplete Cholesky-conjugate gradient (ICCG) algorithm. Recently, BILUM preconditioner, multilevel preconditioner [11] as well as other preconditioning methods [12–14] were proposed and the performance of iterative methods combined with preconditioning techniques has been analyzed in Refs. [15–17]. In general, a good preconditioner should at least meet two requirements. One is that the preconditioner should be cheap to construct and apply. The other is that the preconditioned system should be easy to solve, for counteracting the cost of constructing the preconditioner.

In this article, we investigate SSOR preconditioner for the Block–Broyden method. In order to take less time to calculate the inverse of the preconditioner, we use a diagonal matrix as an approximation of the inverse. We name this method as Approximate-SSOR method (ASSOR), which is based on the ideas arising from the SSOR preconditioner combined with Block–Broyden method. Our purpose is to compare and analyze these two methods.

The rest of the paper is organized as follows. Section 2 introduces relevant knowledge on Block–Broyden Algorithm and preconditioning techniques. SSOR preconditioner is introduced in Section 3. The ASSOR preconditioning method is proposed in Section 4. Section 5 gives general remarks on preconditioning methods based on Block–Broyden algorithm, shows the implementation details and analyzes each method from different aspects. Some numerical results and interpretation of these results are included in Section 6. Section 7 contains the summary remarks.

## 2. Relevant knowledge

### 2.1. Block–Broyden algorithm

In this paper we are concerned with the problem of solving the large system of nonlinear equations

$$F(x) = 0, \quad (1)$$

where  $F(x) = (f_1, \dots, f_n)^T$  is a nonlinear operator from  $R^n$  to  $R^n$ . Suppose that it is possible to generate a new approximation  $x^k$  of  $x^*$  for  $k = 0, 1, \dots$  and the components of  $x$  and  $F$  are divided into  $q$  blocks

$$F = \begin{pmatrix} F_1 \\ \vdots \\ F_q \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_q \end{pmatrix}.$$

We define a block diagonal Broyden matrix as the following form:

$$B^k = \begin{bmatrix} B_1^k & & 0 \\ & \ddots & \\ 0 & & B_q^k \end{bmatrix} = \text{diag}(B_1^k, \dots, B_q^k).$$

The Block–Broyden algorithm applied to (1) could be summarized as follows (see Algorithm 2.1.1 below):

**Algorithm 2.1.1. BB Algorithm**

1. Let  $x^0$  be an initial guess of  $x^*$ , and  $B^0$  an initial block diagonal matrix. Calculate  $r^0 = F(x^0)$ .
2. For  $k = 0, 1$ , until convergence.
  - 2.1 Solve a linear system as follows:

$$B^k s^k = -F(x^k), \tag{2}$$

where  $B^k$  is a Block–Broyden matrix.

- 2.2 Update the approximate solution  $x^{k+1} = x^k + s^k$ .
- 2.3 Calculate  $F(x^{k+1})$ . If it is small enough, stop.
- 2.4 Update the Block–Broyden matrix according to  $B_i^{k+1} = B_i^k + \frac{F_i(x^{k+1})(s_i^k)^T}{(s_i^k)^T s_i^k}$  and repeat the loop.

As a result of the block diagonal structure of  $B^k$ , the nonzero coefficients of the Jacobian that fall outside of the block diagonal are discarded. Thus, block partition strategy plays a major role in the BB method.

In step 2.1, linear system (2) is usually solved by an iterative method. For a given LS, the corresponding BB method could be called BB–LS. For a given GMRES (m) [18], the corresponding BB method could be called BB–GMRES (m) method.

**2.2. Preconditioning methods**

We suppose that there are large sparse linear systems of the form as (3) where  $A = [a_{i,j}]$  is an  $n \times n$  matrix and  $b$  is a given right-hand-side vector, as we have know, the convergence rate of iterative methods depends on spectral properties of the coefficient matrix, hence one may attempt to transform the linear system in (3) into another one that has the same solution but more favorable properties for iterative solution. A preconditioner is a matrix that effects such a transformation. Hopefully, the transformed matrix will have a smaller spectral condition number, and/or eigenvalues clustered around 1.

$$Ax = b. \tag{3}$$

If  $M$  is a nonsingular matrix that approximates  $A$  (in some sense), then the linear system

$$M^{-1}Ax = M^{-1}b \tag{4}$$

has the same solution as (3) but may be easier to solve. Here  $M$  is the preconditioner. In cases where  $M^{-1}$  is explicitly known, the preconditioner is  $M^{-1}$  rather than  $M$ .

System (4) is preconditioned from the left, but one can also precondition from the right

$$AM^{-1}y = b, x = M^{-1}y. \tag{5}$$

In addition, split preconditioning is also possible

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, x = M_2^{-1}y, \tag{6}$$

where the preconditioner is now  $M = M_1M_2$ .

Which type of preconditioning to use depends on the choice of the iterative methods and problem characteristics. In devising a preconditioner, we are faced with a choice between finding a matrix  $M$  that approximates  $A$ , and for which solving a system is easier than solving one with  $A$ , or finding a matrix that

approximates  $A^{-1}$ , so that only multiplication by  $M$  is needed. In general, a good preconditioner  $M$  should meet the following requirements:

- The preconditioned system should be easy to solve.
- The preconditioner should be cheap to construct and apply.

The first property means that the preconditioned iteration should converge rapidly, while the second ensures that each iteration is not too expensive. Notice that these two requirements are in competition with each other. It is necessary to strike a balance between the two needs. With a good preconditioner, the computing time for the preconditioned iteration should be significantly less than that for the unpreconditioned one.

### 3. SSOR preconditioners

The SSOR preconditioner can be derived from the coefficient matrix without any work. If the original, symmetric, matrix is decomposed as

$$A = D + L + L^T$$

in its diagonal, lower and upper triangular part, the SSOR matrix is defined as

$$M = (D + L)D^{-1}(D + L)^T.$$

Then we need to calculate the inverse of the preconditioner and get the transformed linear system  $M^{-1}Ax = M^{-1}b$  that has the same solution as (3).

The SSOR matrix is given in factored form, so this preconditioner shares many properties of other factorization-based methods. For instance, its suitability for vector processors or parallel architectures depends strongly on the ordering of the variables.

### 4. Approximate-SSOR preconditioning method

In this section, we describe an approximate inverse of SSOR preconditioner and propose the Approximate-SSOR preconditioning (ASSOR) method.

#### 4.1. Approximate inverse

The inversion of the preconditioner causes two problems. First, inverting the preconditioner is likely to be a costly operation. Second, initially all diagonal blocks of the matrix may be sparse and we would like to maintain this type of structure. Those two characteristics inspire us to employ an approximation of the inverses. And the simplest approximation to  $M^{-1}$  is the diagonal matrix  $D$  whose elements are reciprocals of the diagonal of  $M$ :  $d_{i,i} = 1/m_{i,i}$ .

#### 4.2. ASSOR algorithm

This preconditioning method first calculates the preconditioner  $M$  that is the same as SSOR preconditioner. The only difference is that it finds an approximation matrix  $D$  to  $M^{-1}$ , the inverse of the preconditioner  $M$ , to transform the original system (3). ASSOR algorithm is as follows:

##### Algorithm 4.2.1. ASSOR algorithm

1. Depose the coefficient matrix  $A$  in (3) as  $A = D + L + L^T$ .
2. Calculate the preconditioner  $M$  according to  $M = (D + L)D^{-1}(D + L)^T$ .
3. Get the diagonal matrix  $D$ :  $d_{i,i} = 1/m_{i,i}$  as an approximation to  $M^{-1}$ .
4. Transform the linear system (3) as  $DAx = Db$  which has the same solution as (3).

It can be easily deduced that the time complexity for the ASSOR method is much lower than that for the SSOR method. However, it cannot be determined whether the number of iterations for the ASSOR method will reduce or even increase largely. It still needs to be further studied on the overall performance of this method considering the above two factors.

**5. Preconditioning methods for Block–Broyden algorithm**

This Section introduces Block–Broyden algorithm combined with several Preconditioning methods (BBP). As introduced in Section 2.1, Block–Broyden(BB) algorithm uses iterative methods without preconditioning for solving the underlying linear system (2), whereas BBP algorithm applies different preconditioners to solve linear systems. Section 5.1 gives the description of BBP algorithm, Section 5.2 shows the implementation details, and Section 5.3 analyzes two BBP algorithms from different aspects.

*5.1. General remarks*

In the following discussion,  $x^* \in R^n$  is an exact solution of system (1), i.e.,  $F(x^*) = 0$ . Let  $x^0$  be an initial guess of  $x^*$ , and suppose that it is possible to generate a new approximation  $x^k$  of  $x^*$  for  $k = 0, 1, \dots$ . Suppose that the components of  $x$  and  $F$  are divided into  $q$  blocks

$$F = \begin{pmatrix} F_1 \\ \vdots \\ F_q \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_q \end{pmatrix},$$

where  $x_i \in R^{n_i}$ ,  $F_i$  is a nonlinear function from  $R^{n_i}$  to  $R^{n_i}$ ,  $i = 1, \dots, q$  and  $\sum_{i=1}^q n_i = n$ . Let  $J(x)$  be the Jacobian matrix of  $F$  at  $x$ , i.e.,  $J(x) = \left. \frac{\partial F(y)}{\partial y} \right|_{y=x}$ . For  $i = 1, \dots, q$ , let  $B_i^0 \in Mat_{n_i \times n_i}(R)$  be a full nonsingular matrix.

Under the same assumptions of the Block–Broyden method described in Section 2.1, we consider generalization of BBP algorithm as follows:

**Algorithm 5.1.1. BBP Algorithm**

1. Let  $x^0$  be an initial guess of  $x^*$ , and  $B^0$  an initial block diagonal approximation of  $J(x^0)$ . Calculate  $r^0 = F(x^0)$ .
2. For  $k = 0, 1 \dots$  until convergence:
  - 2.1. Solve  $B^k s^k = -r^k$  by various preconditioning methods.
    - 2.1.1. Preconditioning this linear system by SSOR, ASSOR preconditioning techniques described in Sections 3 and 4, respectively, to get the preconditioned equation.
    - 2.1.2. Solve the preconditioned equation by Jacobi method.
  - 2.2. Update the solution  $x^{k+1} = x^k + s^k$ .
  - 2.3. Calculate  $r^{k+1} = F(x^{k+1})$ . If  $r^{k+1}$  is small enough, stop.
  - 2.4. Calculate  $(s^k)^T s^k$  and update  $B^{k+1}$  by

$$B_i^{k+1} = B_i^k + \frac{r_i^{k+1} (s_i^k)^T}{(s^k)^T s^k}. \tag{7}$$

Then set  $k = k + 1$ , and go to step 2.

*5.2. Implementation*

The first step in BBP method is the preparation stage and needs only one calculation. It is mainly used to initialize  $B^0$  and  $x^0$ , also calculate  $r^0 = F(x^0)$ .

The second step in BBP method is an iterative process, mainly solving  $q$  blocks of linear equations  $B_i^k s_i^k = -r_i^k$ , then we update the solution  $x^{k+1}$  according to step 2.2 in the method, and calculate the residual according to step 2.3. If the residual is smaller than a given value or the number of iterations is larger than an

upper bound value, the iteration process is stopped. Otherwise, the inner product of  $s^k$  is calculated and  $B^{k+1}$  is updated to repeat the iteration. Fig. 1 describes the computing flow.

5.3. Comparison of the two BBP algorithms

We have applied SSOR and ASSOR preconditioning techniques respectively for Block–Broyden method and these two BBP algorithms will be compared from four indexes, which are the number of iterations, CPU time, construction cost as well as storage need. We can get the value of the first two indexes from experimental data, and analyze the other two indexes in the following subsections.

5.3.1. Construction cost

Let us use  $\bar{M}$  to indicate the construction cost of each preconditioning method, respectively. For SSOR preconditioner described in Section 3, the first step is to calculate the preconditioner. It needs  $\bar{n}$  to calculate  $D^{-1}$ , and  $\frac{\bar{n}^2+\bar{n}}{2}$  to calculate  $(D+L)D^{-1}$ ,  $\bar{n}^2(2\bar{n}-1)$  to calculate  $(D+L)D^{-1}(D+U)$ . The second step is to calculate the inverse of the preconditioner. This is equal to calculate  $\bar{n}$  linear equations. Suppose we use Gauss elimination method to solve these equations, and the complexity of the method is  $(\frac{\bar{n}^3}{3} + \bar{n}^2 - \frac{\bar{n}}{3})$ , so we get the complexity of calculating the inverse of the preconditioner as follows:

$$I_n = \bar{n} \times \left( \frac{\bar{n}^3}{3} + \bar{n}^2 - \frac{\bar{n}}{3} \right) = \frac{\bar{n}^4}{3} + \bar{n}^3 - \frac{\bar{n}^2}{3}. \tag{8}$$

The third step is to calculate  $M^{-1}A$  whose complexity is  $\bar{n}^2(2\bar{n}-1)$ , and the last step is to calculate  $M^{-1}b$  whose complexity is  $\bar{n}(2\bar{n}-1)$ . So we get  $\bar{M}$  as follows:

$$\bar{M} = \bar{n} + \frac{\bar{n}^2 + \bar{n}}{2} + \bar{n}^2(2\bar{n}-1) + \frac{\bar{n}^4}{3} + \bar{n}^3 - \frac{\bar{n}^2}{3} + \bar{n}^2(2\bar{n}-1) + \bar{n}(2\bar{n}-1) \iff \bar{M} = \frac{\bar{n}^4}{3} + 5\bar{n}^3 + \frac{\bar{n}^2}{6} + \frac{\bar{n}}{2}. \tag{9}$$

For ASSOR Preconditioner described in Algorithm 4.2.1, we first calculate the SSOR preconditioner whose complexity is  $\bar{n} + \frac{\bar{n}^2+\bar{n}}{2} + \bar{n}^2(2\bar{n}-1)$ . The second step is to get the approximation matrix  $D$ , which needs  $\bar{n}$  divisions. The third step is to calculate  $D \times A$  whose complexity is  $\bar{n}^2$ , and the last step is to calculate  $D \times b$  whose complexity is  $\bar{n}$ . So we get  $\bar{M}$  as follows:

$$\bar{M} = \bar{n} + \frac{\bar{n}^2 + \bar{n}}{2} + \bar{n}^2(2\bar{n}-1) + \bar{n} + \bar{n}^2 + \bar{n} = 2\bar{n}^3 + \frac{\bar{n}^2}{2} + \frac{7}{2}\bar{n}. \tag{10}$$

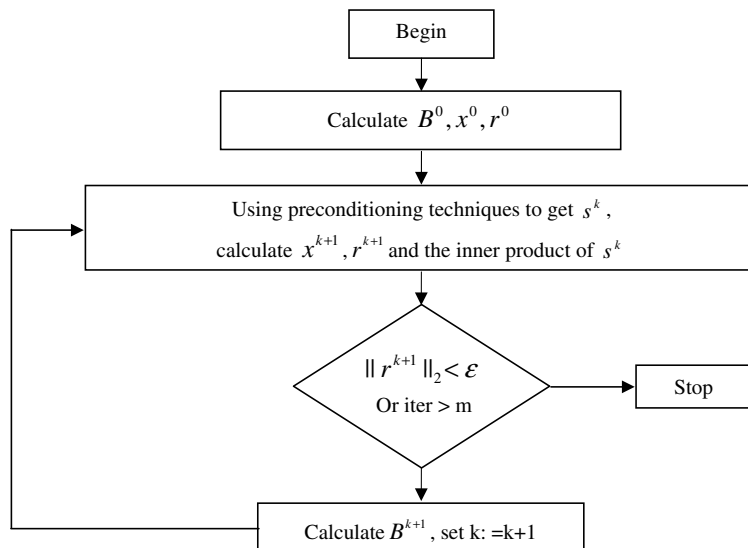


Fig. 1. The flowchart of BBP method.

Table 1  
Comparison of construction cost

	$\bar{M}_1$ (SSOR)	$\bar{M}_2$ (ASSOR)
Complexity	$\frac{\bar{n}^4}{3} + 5\bar{n}^3 + \frac{\bar{n}^2}{6} + \frac{\bar{n}}{2}$	$2\bar{n}^3 + \frac{\bar{n}^2}{2} + \frac{7}{2}\bar{n}$

From Table 1, it is clearly known that SSOR algorithm takes more time to construct preconditioners than ASSOR method.

### 5.3.2. Storage need

As the inverse of the preconditioner is a diagonal matrix when using ASSOR algorithm, it only needs to store  $\bar{n}$  elements, while SSOR algorithm needs to store  $\bar{n} \times \bar{n}$  elements. Therefore, ASSOR algorithm needs less space to store elements than SSOR.

## 6. Numerical experiments

In the numerical experiments, we compare the four indexes of SSOR and ASSOR preconditioning methods for solving a Bratu problem in computational physics. The programming language is C++, using double float variables to calculate the problem.

The nonlinear partial differential equation for this problem can be written as

$$\begin{cases} -\Delta u + u_x + \lambda e^u = f, & (x, y) \in \Omega = [0, 1] \times [0, 1] \\ u|_{\partial\Omega} = 0 \end{cases} \quad (11)$$

on the unit square  $\Omega$  of  $R^2$  with zero Dirichlet boundary conditions. It is known as the Bratu problem and has been used as a test problem by Brown and Saad [5], Yang [7] and Jiang [8–10]. The function  $f$  is chosen such that the solution of the discretized problem is unity. Each side of the unit square is divided into  $N - 1$  intervals, and each small square into two triangles. Using a linear element  $P$ , a standard finite element approximation gives a large system of nonlinear equations of size  $N^2$ . For  $\lambda \geq 0$ , the system has a unique solution. In the following tests, we suppose  $f = e$ ,  $\lambda = 1$  and  $N = 20, 30, 100$  and  $200$ , giving four grids,  $M1, M2, M3$  and  $M4$ , with 400, 900, 10000 and 40000 unknown, respectively. The original numbering of the grids is obtained layer by layer along the vertical axis. In each layer, vertices are ordered from left to right.

Let  $r$  be the nonlinear discretized function of dimension  $n = N^2$  obtained from (11), so the residual vector norm is  $\|r^k\|_2$ , where  $k$  is the number of nonlinear iterations. We also set an upper bound value  $m = 5000$ . The stopping criterion used in the following tests is either  $\|r^k\|_2 < 10^{-5}$  or  $m > 5000$ . Here we use various preconditioning methods to solve the underlying linear equations, constituting three different Block–Broyden Preconditioning (BBP) methods to solve the nonlinear system. From boundary condition we can get the initial solution  $x^0 = 0$  and the initial block jacobi matrix  $B^0$ .

### 6.1. Comparison of number of iterations under different dimensions

First we show the number of iterations under small dimensions. When the grid is  $M1$  and the block number  $q1 = 8$ , the linear system is divided into eight blocks each of which has same dimension of 50 ( $20 \times 20/8$ ). When the grid is  $M2$  and the block number  $q2 = 10$ , the linear system is divided into 10 blocks each of which has same dimension of 90 ( $30 \times 30/10$ ). Tables 2 and 3 report the number of iterations including both the number of nonlinear iterations and the number of iterations for solving each block in  $M1, q1$  and  $M2, q2$ , respectively. These methods include SSOR and ASSOR methods, as well as the unpreconditioned one, for the purpose of comparison. The value “ $R$ ” in Tables 2 and 3 indicates the number of nonlinear iterations, and the value “ $R[i]$ ” in Tables 2 and 3 indicates the number of iterations for solving each linear block during the  $i$ th nonlinear iteration.

Table 2  
Comparison of the number of iterations using various methods in M1, q1

R	SSOR	89							
	ASSOR	86							
	No preconditioner	88							
R[60]	SSOR	4	5	6	8	7	10	5	5
	ASSOR	7	18	25	35	34	45	17	9
	No preconditioner	8	21	29	41	39	53	20	11
R[60]	SSOR	3	5	4	4	5	5	5	3
	ASSOR	5	18	11	10	11	18	17	5
	No preconditioner	7	23	14	13	14	21	21	8

Table 3  
Comparison of the number of iterations using various methods in M2, q2

R	SSOR	153									
	ASSOR	160									
	No preconditioner	159									
R[17]	SSOR	4	9	15	13	22	25	13	15	9	4
	ASSOR	11	39	62	54	94	94	54	62	39	11
	No preconditioner	13	44	71	60	107	107	60	71	44	13
R[51]	SSOR	3	7	8	10	9	13	8	7	6	3
	ASSOR	10	30	45	44	31	32	46	33	31	10
	No preconditioner	12	37	54	54	39	39	56	42	38	12

From Tables 2 and 3, the following observations can be made statistically based on our experiment data:

- The number of nonlinear iteration is almost the same when using these three methods, but the number of iteration in each block is smaller for SSOR and ASSOR preconditioners than the unpreconditioned method. This is because we do not apply any preconditioning technique to solve the nonlinear system (1), but to the underlying linear system, as described in step 2.1 of BBP method in Algorithm 5.1.1. After preconditioning the linear equation  $B^k s^k = -r^k$ , the number of iteration in each block reduces a lot.
- The number of iterations in each block is larger for ASSOR method than SSOR. This conclusion can also be verified by the data from Tables 4–8, which is got under large dimensions and different numbers of block.

When the grid is M3, we set the block number as  $q3 = 300, q4 = 500$  and  $q5 = 1000$ , respectively. When the grid is M4, we set the block number as  $q6 = 1000$  and  $q7 = 2000$ . Tables 4–8 show the sum of number of iterations in each block during certain iteration process under these five circumstances, respectively. In Tables 4–8 the value “iter” denotes the number of iterations when solving the nonlinear system, and  $k_{p_1}, k_{p_2}, k_n$  refer to the sum of number of iterations for SSOR method, ASSOR method and the unpreconditioned method, respectively.

From the data shown in Tables 4–8, we can easily find that the number of iterations is larger for ASSOR method than SSOR, which is already verified by the data shown in Tables 2,3.

Table 4  
Comparison of the total number of iterations using various methods in M3, q3

	$k_{p_1}$ (SSOR)	$k_{p_2}$ (ASSOR)	$k_n$ (No preconditioner)
iter = 10	594	2357	2951
iter = 30	588	2544	3295
iter = 50	775	2758	3507

Table 5  
Comparison of the total number of iterations using various methods in  $M3$ ,  $q4$

	$k_{p_1}$ (SSOR)	$k_{p_2}$ (ASSOR)	$k_n$ (No preconditioner)
iter = 10	976	3866	4846
iter = 70	1196	4202	5414
iter = 85	1290	4946	6142

Table 6  
Comparison of the total number of iterations using various methods in  $M3$ ,  $q5$

	$k_{p_1}$ (SSOR)	$k_{p_2}$ (ASSOR)	$k_n$ (No preconditioner)
iter = 15	1938	6572	8530
iter = 50	1910	6474	8414
iter = 140	2954	12530	14684

Table 7  
Comparison of the total number of iterations using various methods in  $M4$ ,  $q6$

	$k_{p_1}$ (SSOR)	$k_{p_2}$ (ASSOR)	$k_n$ (No preconditioner)
iter = 5	1980	5930	7910
iter = 10	1970	5910	7890
iter = 12	1970	5910	7890

Table 8  
Comparison of the total number of iterations using various methods in  $M4$ ,  $q7$

	$k_{p_1}$ (SSOR)	$k_{p_2}$ (ASSOR)	$k_n$ (No preconditioner)
iter = 10	3940	11792	15752
iter = 20	3920	11712	15672
iter = 23	3920	11690	15652

## 6.2. Comparison of CPU Time under different dimensions

CPU time means the time needed for a computer to complete certain task. First we show the CPU time under small dimensions. Tables 9 and 10 report the CPU time needed for solving each block in  $M1$ ,  $q1$  and  $M2$ ,  $q2$ , respectively. The value “ $T[i]$ ” in Tables 9 and 10 indicates the CPU time for solving each linear block during the  $i$ th nonlinear iteration.

From Tables 9 and 10 we know that ASSOR algorithm needs much less CPU time to solve each linear system. Tables 11–15 show the sum of the CPU time in each block during certain iteration process under large dimensions. In Tables 11–15 the value “iter” denotes the number of iterations when solving the nonlinear system, and  $k_{i_1}$ ,  $k_{i_2}$  refer to the sum of the CPU time for SSOR method and ASSOR method, respectively.

From the data shown in Tables 11–15, we can easily find that ASSOR method always needs less CPU time than SSOR, which is already verified by the data shown in Tables 9,10.

## 6.3. Comparison of construction cost

We compare the construction cost of ASSOR method with SSOR using the data shown in Table 6 as an example. The dimension of each linear system can be calculated as follows:

$$\bar{n} = (100 \times 100)/1000 = 10.$$

Thus we can calculate  $\bar{M}_1$ (SSOR),  $\bar{M}_2$ (ASSOR) according to Table 1:

Table 9  
Comparison of the CPU time in  $M1, q1$

	SSOR	ASSOR
T[30]	0.010014	0.008852
	0.010064	0.008902
	0.010124	0.008932
	0.010154	0.008982
	0.010204	0.009032
	0.010274	0.009063
	0.010324	0.009123
	0.010354	0.009153
T[60]	0.019948	0.017735
	0.020008	0.017785
	0.020048	0.017845
	0.020098	0.017875
	0.020138	0.017915
	0.020209	0.017955
	0.020279	0.018015
	0.020389	0.018065

Table 10  
Comparison of the CPU time in  $M2, q2$

	SSOR	ASSOR
T[17]	0.039316	0.035340
	0.039596	0.035591
	0.039927	0.035821
	0.040207	0.036061
	0.040508	0.036352
	0.040818	0.036652
	0.041109	0.036903
	0.041319	0.037103
	0.041529	0.037293
	0.041730	0.037473
T[51]	0.140471	0.110038
	0.140812	0.110228
	0.141183	0.110428
	0.141603	0.110629
	0.141934	0.110849
	0.142154	0.111129
	0.142514	0.111360
	0.142885	0.111560
	0.143245	0.111760
	0.143646	0.111950

Table 11  
Comparison of the total CPU time in  $M3, q3$

	$k_{r_1}$ (SSOR)	$k_{r_2}$ (ASSOR)
iter = 10	0.039206	0.032286
iter = 30	0.120052	0.097039
iter = 50	0.198585	0.162143

$$\bar{M}_1 = \frac{\bar{n}^4}{3} + 5\bar{n}^3 + \frac{\bar{n}^2}{6} + \frac{\bar{n}}{2} = \frac{10^4}{3} + 5 \times 10^3 + \frac{10^2}{6} + \frac{10}{2} = 8355,$$

$$\bar{M}_2 = 2\bar{n}^3 + \frac{\bar{n}^2}{2} + \frac{7}{2}\bar{n} = 2 \times 10^3 + \frac{10^2}{2} + \frac{7}{2} \times 10 = 2085.$$

Table 12  
Comparison of the total CPU time in  $M3$ ,  $q4$

	$k_{t_1}$ (SSOR)	$k_{t_2}$ (ASSOR)
iter = 10	0.012908	0.011476
iter = 70	0.085943	0.073185
iter = 85	0.104039	0.088487

Table 13  
Comparison of the total CPU time in  $M3$ ,  $q5$

	$k_{t_1}$ (SSOR)	$k_{t_2}$ (ASSOR)
iter = 15	0.008282	0.006539
iter = 50	0.027749	0.022632
iter = 140	0.072283	0.060547

Table 14  
Comparison of the total CPU time in  $M4$ ,  $q6$

	$k_{t_1}$ (SSOR)	$k_{t_2}$ (ASSOR)
iter = 5	0.085182	0.068228
iter = 10	0.168181	0.138198
iter = 12	0.201239	0.165427

Table 15  
Comparison of the total CPU time in  $M4$ ,  $q7$

	$k_{t_1}$ (SSOR)	$k_{t_2}$ (ASSOR)
iter = 10	0.052815	0.041399
iter = 20	0.101716	0.081306
iter = 23	0.116377	0.093274

From calculation we can draw a conclusion that the construction cost is much smaller for ASSOR method than for SSOR method, and this conclusion can be verified by the data shown in other tables.

#### 6.4. Comparison of storage need

The storage need of ASSOR method and SSOR can be compared using the data shown in Table 7 as an example. The dimension of each linear system can be calculated as follows:

$$\bar{n} = (200 \times 200)/1000 = 40.$$

According to the analysis in Section 5.3.2 we know that when calculating the inverse of the preconditioner, SSOR algorithm needs to store  $\bar{n} \times \bar{n}$  (which is 1600) elements, while ASSOR algorithm only needs to store  $\bar{n}$  (which is 40) elements, therefore largely reducing the storage space.

## 7. Conclusions

We have proposed the SSOR and ASSOR preconditioners for Block–Broyden method to solve nonlinear systems arising from scientific and engineering computing. ASSOR is based on the ideas arising from the SSOR preconditioner. Theoretical analyses and numerical experiments show that though ASSOR method needs more times of iterations than SSOR, it needs less CPU time, smaller construction cost and lower storage need. Therefore, ASSOR algorithm has better performance than SSOR method as a whole.

On the other hand, the results demonstrated show evidently some advantages to combine Block–Broyden algorithm with the preconditioners. One of the reasons is that a proper preconditioner is used to transform the

system, so the spectral properties of the Broyden matrix is improved and quick convergence speed is gained. On the other hand, as the iterative matrix is block diagonal, the algorithm only needs to store the diagonal matrix, thus largely reduces memory space. In future work, we expect to develop more effective preconditioning methods for Block–Broyden algorithm so that the preconditioned system can converge quickly and meanwhile can be constructed as easily as possible.

Finally, we have to point out that the SSOR and ASSOR preconditioners have the same diagonal matrix form as that of BB method. It is easily to implement them in parallel environment [6,7]. This is an important benefit from the preconditioners.

## References

- [1] Igor Boglaev, Monotone iterative algorithms for a nonlinear singularly perturbed parabolic problem, *J. Comput. Appl. Math.* 172 (2004) 313–335.
- [2] S. Amat, S. Busquier, J.M. Gutiérrez, Geometric constructions of iterative functions to solve nonlinear equations, *J. Comput. Appl. Math.* 157 (2003) 197–205.
- [3] Emanuele Galligani, The Newton-arithmetic mean method for the solution of systems of nonlinear equations, *Appl. Math. Comput.* 134 (2003) 9–34.
- [4] Y. Saad, H.A. van der Vorst, Iterative solution of linear systems in the 20th century, *J. Comput. Appl. Math.* 123 (2000) 1–33.
- [5] P.N. Brown, Y. Saad, Hybrid Krylov methods for nonlinear systems of equations, *SIAM J. Sci. Stat. Comput.* 11 (1990) 450–481.
- [6] G. Yang, L. Dutto, M. Fortin, Inexact block Jacobi Broyden methods for solving nonlinear systems of equations, *SIAM J. Sci. Comput.* 18 (1997) 1367–1392.
- [7] G. Yang, Analysis of parallel algorithms for solving nonlinear systems of equations, *Chinese J. Comp.* 23 (2000) 555–777 (in Chinese).
- [8] Peng Jiang, Geng Yang, Performance analysis of preconditioners based on Broyden method, *Appl. Math. Comput.* 178 (2) (2006) 295–308.
- [9] Peng Jiang, Geng Yang, Chunming Rong, Combined method for nonlinear systems of equations, in: *LNCS – IV, Computational Science – ICCS 2006: 6th International Conference*, vol. 3994, Springer-Verlag, 2006, pp. 693–699.
- [10] Peng Jiang, Geng Yang, Chunming Rong, Performance analysis of block Jacobi preconditioning technique based on Block Broyden method, in: *LNCS – I, Computational Science – ICCS 2006: 6th International Conference*, vol. 3991, Springer-Verlag, 2006, pp. 794–797.
- [11] J. Zhang, A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices, *SIAM J. Matrix Anal. Appl.* 22 (2000) 925–947.
- [12] M. Bollhöfer, A robust ILU with pivoting based on monitoring the growth of the inverse factors, *Linear Algebra Appl.* 338 (2001) 201–218.
- [13] K. Chen, An analysis of sparse approximate inverse preconditioners for boundary integral equations, *SIAM J. Matrix Anal. Appl.* 22 (2001) 1058–1078.
- [14] I.E. Kaporin, I.N. Konshin, A parallel block overlap preconditioning with inexact submatrix inversion for linear elasticity problems, *Numer. Linear Algebra Appl.* 9 (2002) 141–162.
- [15] J. Von Hagen, W. Wiesbeck, Physics-based preconditioner for iterative algorithms in MoM-problems, *IEEE Trans. Antennas Propagation* 50 (2002) 1315–1316.
- [16] A. Padiy, O. Axelsson, B. Polman, Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems, *SIAM J. Matrix Anal. Appl.* 22 (2000) 793–818.
- [17] O. Axelsson, J. Karátson, Conditioning analysis of separate displacement preconditioners for some nonlinear elasticity systems, *Math. Comput. Simulat.* 64 (2004) 649–668.
- [18] C. Vuik, R.P. van Nooyen, P. Wesseling, Parallelism in ILU-preconditioned GMRES, *Parallel Comput.* 24 (1998) 1927–1946.