REORDERING ALGORITHM FOR PRECONDITIONING NONLINEAR PROBLEMS

C. BONCHIS, ST. MARUSTER
INSTITUTE E-AUSTRIA TIMISOARA

Abstract. The paper deals with preconditioning nonlinear system of equations on the base of reordering equations and unknowns. The goal of these permutations is to produce Jacobians with dense diagonal blocks and hence to improve the characteristics of certain block Jacobi methods, like block Newton Jacobi, block NCG Jacobi, etc. An algorithm for computing the preconditioners is proposed and some numerical experiment are also given. In the annex of the paper, the MathCad cod for the algorithm is presented.

Keywords: nonlinear systems, preconditioning, reordering.

1. Introduction

To precondition a nonlinear system involves, in the large, to compute preconditioners at every step of iteration. A global perconditioner, which is appropriate for entire iteration process, is of course also desirable, but in the nonlinear case the computational cost for such a preconditioner is highly expensive (for example, the approximation of the inverse of the Jacobian in the solution).

The general idea of preconditioning nonlinear systems consists in substitute the original function $F : D \subseteq R^n \to R^n$ with an equivalent one $G : D' \subseteq R^n \to R^n$ which have better properties concerning the computation of a solution (generally by a certain iterative method). The two functions are equivalent in the sense that they have the same solution. Other than having the same solution, the two functions may have completely different forms.

A very simple preconditioners of this type are the left and the right matrix preconditioners. The left preconditioner is a nonsingular matrix M and the preconditioned function is defined by $G := MF$. If Newton-like methods are used for solving the equation $G(x) = 0$, then it would be reasonable that $M = G(x^*)^{-1}$, where x^* is the solution of the equation $F(x) = 0$. The right preconditioned function is defined by $G(y) = F(Mx)$ and the proper unknown x is recovered from $x = My$. In the both cases the matrix M works as a global preconditioner.

More sophisticated preconditioners of this type are the single-level and multilevel nonlinear additive Schwarz preconditioners, [1], [8], [10]. For constructing single-level preconditioner, let $S = (1, 2, ..., n)$ be an set of index; i.e., one integer for each unknown x_i and F_i. Consider $S_1, S_2, ..., S_N$ a partition of S such that $\cup_{i=1}^{N} = S$ and $s_i \subset S$. It is allowed the subsets to have overlap, that is, if n_i is the

dimension of S_i, then, in general, $\sum_{i=1}^{N} \geq n$. For each S_i it is defined $V_i \subset R^n$ as

$$V_i = \{v | v(v_1, ..., v_n)^T \in R^n, \quad v_k = 0, \quad if \quad k \notin S_i\}$$

and an $n \times n$ matrix I_{S_i} whose kth column is either the kth column of the identity $n \times n$ matrix if $k \in S_i$ or zero if $k \notin S_i$. Let $F_{S_i} = I_{S_i}$ and define the function $T_i : R^n \to V_i$ as the solution of the subspace nonlinear equation $F_{s_i}(v - T_i(v)) = 0$, for $i = 1, ..., N$. The single-level nonlinear additive Schwarz preconditioner is defined by

$$G(x) = \sum_{I=1}^{n} T_i(x).$$

The common way for preconditioning nonlinear systems is to use some linearization of a nonlinear system and preconditioning the resulting linear system at each nonlinear (outer) iteration. In fact, such a preconditioning attempts to improve the iterative process (inner iteration) for solving the corresponding linear systems. The inexact Newton method is probably the most popular algorithm in this class. The outer iteration involves the solution of the linear system $J(x_k)s_k = -F(x_k)$ and sets $x_{k+1} = x_k + s_k$; $J(x_k)$ is the Jacobian of F computed in the current iteration x_k. When n is large and the nonzero structure of $J(x_k)$ does not help, the classical direct methods (Gauss elimination, LU factorization, etc.) produce a large amount of fill-in and, so, they cannot be used for practical computation. It is used an iterative method (inner iteration) for solve the linear system at each step of outer iteration. Obviously, the linear systems must not be solved with the same accuracy during the whole process of outer iteration. The following problem arise: what must be the accuracy in solving linear systems? A natural idea is that this must be smaller as the outer iteration progress. In [7] the following stopping criteria is suggested: an increment s_k is accepted as an approximate solution if

$$\|J(x_k)s_k + F(x_k)\| \leq \theta_k \|F(x_k)\|,$$

where $0 < \theta_k \leq \theta < 1$ for all $k \in N$. Under suitable conditions, this algorithm has local linear convergence. If $\theta_k \to 0$ the convergence is superlinear.

In [4], some approximation of the Jacobian matrix is used in the linear iteration and GMRES method is applied for solve the linear system. Then Klylov subspace information generated by GMRES is recycled in order to compute step by step preconditioners. The general idea is either to deflate or to shift to one the eigenvalues of the matrix. The successive preconditioners are given by recursive formula of the form $M_k^{-1} := M_{k-1}^{-1} M_{(k-1)}^{-1}$ and by certain explicit formula for $M_{(k-1)}^{-1}$.

The Conjugate gradient (CG) method for inner iteration in inexact Newton method is proposed in [5], [6]. A least-change secant-update procedures is proposed to generate the matrix preconditioners B_k at each iteration of the inexact Newton method and then CG method is applied to the equivalent preconditioned system $B_k^{-1} J(x_k)s = -B_k^{-1} F(x_k)$. The preconditioners are generate using Broyden "good formula" so that they are easy to compute. An other type of preconditioning are incomplete LL^T factorizations for positive definite matrices, developed in [9].

The effect of the permutation algorithms on the performance of certain methods for linear systems is explored in some papers [11], [12], [13]. For example, in [11], a symmetric permutation of the matrix A of a system $Ax = b$ of the form P^TAP is considered and then it is solved the equivalent system $P^TAPy = P^Tb$ and $x = Py$. The permutation algorithm produces a permuted matrix with dense diagonal blocks, while the entries outside the diagonal have magnitude below a prescribed threshold. The algorithm works with the graph of the matrix, choosing one node at a time, adding it to a group of nodes which would form the blocks along the diagonal, if the new node satisfies certain criteria. The amount of work is controlled by not searching through all the available nodes, but through a subset of eligible nodes, those which are adjacent to some nodes in the current set, i.e., in the group of nodes of the block along the diagonal being formed.

The structure of the paper is as follow. A brief description of our algorithm is given in section 2. The performance of this algorithm is illustrated in section 3, using spars matrices of moderate dimensions. In the last section the effective MathCad code is given together with necessary comments. Note that the additional analysis and experiments concerning the effect of this preconditioning on the certain Jacobi block method, will be reported in a forthcoming paper.

2. The algorithm

Let A be the "structure matrix" of a nonlinear system $F(x) = 0$, where $F : D \subseteq R^n \to R^n$, i.e., a matrix which shows what unknowns appear in every equation. A is a boolean matrix $A = (a_{ij}), i, j = 1, ..., n$, where $a_{ij} = 1$ if and only if the component x_j appears in the ith equation. The main idea of the algorithm is to process the matrix A in such a way that the nonzero elements of A be gather around the main diagonal.

Let i_l, j_l and i_u, j_u be index in A which satisfy the following properties:

(1)$a_{i_l j_l} \neq 0$, $a_{i_u j_u} \neq 0$;
(2)$i_l \geq j_l$, $j_u \geq i_u$;
(3)$a_{i+k,k+1} = 0$, $k = 0, 1, ..., n - i + 1$, $\forall i > i_l - j_l + 1$;
$a_{k+1,j+k} = 0$, $k = 0, 1, ..., n - j + 1$, $\forall j > j_u - i_u + 1$.

These index define the diagonal band of the matrix, a_{i_l,j_l} being a nonzero element on the lowest diagonal and $a_{i_u j_u}$ being a nonzero element of the most up diagonal.

Remark. The condition (2) stipulates that the lowest diagonal is under the main diagonal and that the most up diagonal is over the main diagonal. If no, then a special case arise; for instance, if $i_l < j_l$ then either the solution of the system is trivial if $a_{ii} = 1$ for $i = 1, ..., n$, or some un-determinism are taking place.

Definition 1. The width of the diagonal band is defined by

$$w(A) = i_l - j_l + j_u - i_u + 1.$$

Remark. $w(A)$ represents the number of diagonals which compose the band of A.

Definition 2. A permutation matrix P is a boolean $n \times n$ matrix with the property that in every row and in every column of P there is exact one element hose value is one.

It is easy to see that the left product of the matrix A with P produces a permutation of rows, while the right product of the matrix A with P produces a permutation of columns. We will denote by $\mathbf{P}_{n\times n}$ the set of all the $n \times n$ permutation matrices.

The problem which we are interested in, consists in find a $P_l, P_r \in \mathbf{P}_{n\times n}$ for which

$$w(P_l A P_r) = \min_{P_L, P_R \in \mathbf{P}_{n\times n}} w(P_L A P_R) \tag{1}$$

The goal of these permutations is to produce matrices with dense diagonal band (hence, a nonlinear system with dense diagonal band of the Jacobian) and presumable with improved convergence properties for parallel algorithms which uses the Jacobian, like block Newton-Jacobi, block CG-Jacobi, etc. The eigenvalue of the Jacobian (usually computed in solution), rest unchanged and so a global iterative method couldn't be improved. Nevertheless, the diagonal blocks of the Jacobian could have essential diminished condition numbers and therefore the above mentioned methods should works better. We also mention that the matrices P_l, P_r may be interpreted as left and right preconditioners. So, in Newton method, the preconditioning linear system will have the form

$$P_l J(x) P_r y = P_l F(x).$$

Note that the solution is preserved.

The solution of a nonlinear system involves the following steps:

(1) Computing the structure matrix A;
(2) Computing the preconditioners P_l and P_r satisfying (1);
(3) Reordering the equations and the unknowns;
(4) Splitting of the systems in blocks with or without overlap;
(5) Applying a block Jacobi method.

In fact, the preconditioning is achieved in the step (2). In the following, we will give a simple algorithm for computing the two matrices P_l, P_r.

Let $c_i = (a_{1i}, ..., a_{ni})$ be the ith column of A. Let $l(c_i)$ denote the first element of c_i hose value is equal to one (i.e., if $l(c_i) = a_{j_l i} = 1$, then $a_{ji} = 0$ for all $j < j_l$, and similar, let $r(c_i)$ denote the last element of c_i hose value is equal to one, that is $r(c_i) = a_{j_r i} = 1$. Now, define the *average index of the column* c_i of the possible nonzero elements of c_i, by

$$mc_i = \left[\frac{j_l + j_r}{2}\right],$$

where [.] means the integer part.

Similar definition have the average index of the row j of the matrix, which we will denote by mr_j.

The algorithm.

Repeat until convergence
Step 1: Compute mc_i for $i = 1, ..., n$;
Step 2: Transform the sequence $(mc_1, ..., mc_n)$ in a permutation of $(1, ..., n)$; p_c denote this permutation;
Step 3: Preform the permutation p_c of columns in the matrix A;
Step 4: Compute mr_j for $j = 1, ..., n$;
Step 5: Transform the sequence $(mr_1, ..., mr_n)$ in a permutation of $(1, ..., n)$; p_r denote this permutation;
Step 6: Preform the permutation p_r of rows in the matrix A.

Remarks.
The step 3 involves the following processing: the maximum value of mc_i is forced on n (if more elements have this maximum, say k elements, then these elements are forced on $n, n-1, ..., n-k+1$) respectively; next, the second element of mc_i in magnitude is forced on $n-1$ (or on $n-k$); etc. Analogous for step 5. Obviously, the product of permutation matrices obtained successively in the steps 3 and respectively 5, give the searching preconditioners. More precisely, if $l_1, ..., l_m$ denote the permutation matrices obtained in the step 3 an if $r_1, ..., r_m$ denote the permutation matrices obtained in the step 5, then $P_l = l_m l_{m-1} ... l_1$ and $P_r = r_1 r_2 ... r_m$.

The algorithm will stops when no change turn up after step 3 or 6 or if a cycle arises during of the process. The convergence analysis and the complexity of this algorithm will be done in a future work. In the annex of this paper a detailed MathCad programm is given.

In a practical application, it is not necessarily to effective computing the preconditioners (the matrices P_l and P_r); the reordering of equations and the unknowns result directly from permutation matrices or even from the permutations computed in routines vr (see annex).

3. Numerical experiments

The reordering algorithm was tested on a significant number of spars matrices, from 8×8 until 20×20 dimension, and a special experiment on a 80×80 matrix. The results were encouraging, for all considered cases the algorithm works well, the reordered matrices having dense diagonal bands and the computational effort being reasonable, the number of iteration having the average value 18 in the cases of small matrices and the value 74 for 80×80 matrix.

To illustrate the effectiveness of the approach described in section 2, we show below the results for a spars 8×8 matrix. The nonzero elements of

the matrix were taking as 1,2,...,17, in order to see the new positions of the elements after reordering.

$$A := \begin{pmatrix} 0 & 0 & 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 & 5 & 0 & 0 & 6 \\ 7 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9 & 10 \\ 0 & 11 & 12 & 0 & 0 & 0 & 0 & 0 \\ 13 & 0 & 0 & 14 & 0 & 0 & 15 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 \end{pmatrix} \quad B := \begin{pmatrix} 8 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 15 & 14 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 6 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 17 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 12 & 11 \end{pmatrix}$$

It can see that the width of the diagonal band was reduced from 12 to 3, that is a 75% reduction in percents.

4. Annex

```
minc(a, j) := | i ← 0
              | while a_{i,j} = 0
              |   i ← i + 1
              | return i
```

```
maxc(a, j) := | i ← n
              | while a_{i,j} = 0
              |   i ← i − 1
              | return i
```

```
vc(a) := | for j ∈ 0..n
         |   vc_{0,j} ← j
         |   m ← minc(a, j) + maxc(a, j)
         |   mi ← m/2 − mod(m, 2)/2
         |   vc_{1,j} ← mi
         | return vc
```

```
maxv(vc) := | max ← −2
            | for j ∈ 0..n
            |   if vc_{1,j} > max
            |     max ← vc_{1,j}
            |     j0 ← j
            | return j0
```

```
vr(vv) :=
  for k ∈ 0..n
    j ← maxv(vv)
    vr[0,k] ← k
    vr[1,j] ← n − k
    vv[1,j] ← −1
  return vr
```

```
sc(a) :=
  vf ← vr(vc(a))
  for j ∈ 0..n
    jn ← vf[1,j]
    for i ∈ 0..n
      a1[i,jn] ← a[i,j]
  return a1
```

```
minl(a,i) :=
  j ← 0
  while a[i,j] = 0
    j ← j + 1
  return j
```

```
maxl(a,i) :=
  j ← n
  while a[i,j] = 0
    j ← j − 1
  return j
```

```
vl(a) :=
  for j ∈ 0..n
    vl[0,j] ← j
    m ← minl(a,j) + maxl(a,j)
    mi ← m/2 − mod(m,2)/2
    vl[1,j] ← mi
  return vl
```

```
maxv(vl) :=
  max ← −2
  for j ∈ 0..n
    if vl[1,j] > max
      max ← vl[1,j]
      j0 ← j
  return j0
```

```
sl(a) :=
  vf ← vr(vl(a))
  for i ∈ 0..n
    jn ← vf[1,i]
    for j ∈ 0..n
      a1[jn,j] ← a[i,j]
  return a1
```

```
vr(vv) :=
  for k ∈ 0..n
    j ← maxv(vv)
    vr[0,k] ← k
    vr[1,j] ← n − k
    vv[1,j] ← −1
  return vr
```

$$s := \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad d := \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$s \cdot A \cdot d = \begin{pmatrix} 12 & 11 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 6 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 14 & 15 & 13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 8 \end{pmatrix}$$

$$p(b, itm) := \left| \begin{array}{l} \text{for } it \in 0 .. itm \\ \quad \left| \begin{array}{l} b \leftarrow sc(b) \\ b \leftarrow sl(b) \end{array} \right. \\ \text{return } b \end{array} \right. \qquad p(A, 30) = \begin{pmatrix} 8 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 15 & 14 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 6 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 17 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 12 & 11 \end{pmatrix}$$

Comments.

Routine **minc(a,j)** and **max(a,j)**: Compute the first and the last nonzero elements of the column j of the matrix a;

Routine **vc(a)**: Compute *average index of the columns* of the matrix a. For example:

$$vc(A) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 6 & 2 & 4 & 1 & 3 & 3 & 4 \end{pmatrix}$$

Routine **maxv(vc)**: Compute the maximum of the second row of the matrix vc (usually, vc=vc(A)). For example, maxv(vc(A))=1.

Routine **vr(vv)**: Transforms the matrix vv (this matrix must have two rows, the first being 0,1,...,n) into a proper permutation. For example:

$$vc(A) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 6 & 7 & 1 & 5 & 0 & 3 & 2 & 4 \end{pmatrix}$$

Routine **vc(a)**: Performs the permutation given in routine **vr** on the columns of the matrix a.

The routines **minl, maxl, vl, sl** are similar with **minc, maxc, vc, sc** but they work on the rows (lines) of a matrix.

The matrices s and d are the preconditioners computed after 7 iterations, that is $s = s_7 s_6 ... s_1$ and $d = d_1 d_2 ... d_7$. It can see that sAd is the preconditioned matrix and that $w(sAd) = 3$.

p(b,itm) is the general program; itm represent the iteration number which must be given by the user. For example, after 30 iteration the predonditioned matrix is given above.

Remark. The precondition matrix is not unique, in other words, the solution of (2) is not unique. In our example, $sAd \neq p(A, 30$, but $w(sAd) = w(p(A, 30))$.

References

[1] Xiao C.Cai, D.E.Keyes, L.Marcinkowski, Nonlinear Additive Schwarz Preconditioners and Applications in Computational Fluid Dynamics,?

[2] Xiao C.Cai, D.E.Keyes, L.Marcinkowski, Two-level nonlinear Schwarz preconditioned inexact Newton methods, 2001 (in preparation)

[3] Xiao C.Cai, D.E.Keyes, Nonlinearly preconditioned inexact Newton algorithms, SIAM J. Sci. Comput. 2000 (submited).

[4] L.Loghin, D.Ruiz, A.Touhami, Adaptive preconditioners for nonlinear systems of equations, CERFACS Technical Report TR/PA/04/02

[5] J.M.Martinez, An extension of the theory of secant preconditioners, J. Comput. Appl. Math., Vol. 60 (1995), pp. 115-125.

[6] J.M.Martinez, A Theory of Secant Preconditioners, Math. Comput., Vol. 60 (1993), pp. 681-698.

[7] Dembo, R.S., Eisenstat, S.C., Steihaug, T., Inexact Newton methods, SIAM J. Num. Anal., Vol. 19 (1982), pp. 400-408.

[8] Jenkins, E.W., et al, A Newton-Krylov-Schwrtz methods for Ricard's equations, Technical report, North Carolina State University, Center for researche in Scientific Computational, Oct., 1999

[9] Van Der Vorst, H.A., Dekker, K., Conjugate gradient type methods and preconditioning, J. Comput. Appl. Math., Vol. 24 (1988), pp. 73-87.

[10] Kees, C.E. at al.,Versatile Multilevel Schwartz Preconditioners for Multiphase Flow,

[11] Benzi, M., Choi, H., Szyld, D.B.,Threshold ordering for Preconditionong Nonsimetric Problems,
[12] Duff, I., Meurant, G.A., The effect of ordering on preconditioned conjugate gradient , BIT, Vol. 29 (1989), pp. 635-657.
[13] O'NeilJ., Szyld, D.B., A block ordering method for sparse matrices, SIAM J. Sci. Stat. Comput. Vol. 11 (1990), pp. 811-823.