Sparse Matrix Techniques in Scientific Computing

Aurelian Nicola, Constantin Popa

Ovidius University, Blvd. Mamaia 124, 900527 Constanta, Romania

{anicola, cpopa}@univ-ovidius.ro

Abstract. Although the most important and relevant part of the scientific activity of Dr. Neculai Andrei is related to the design of efficient algorithms and software products for optimization problems, his first book, written in 1983 was devoted to sparse matrices and some of their applications in scientific computing. This is why we decided to present in our contribution some developments that we made in this direction, in the context of Matlab software, and 25 years after Neculai Andrei's book. The paper presents the design and efficient implementation of some sparse matrix codes for numerical solution of a 2D convection-diffusion-reaction problem, by a preconditioned CG algorithm.

Keywords: sparse matrices, scientific computing, convection-diffusion-reaction problems, multigrid algorithms, preconditioned CG algorithm

Aurelian Nicola was born on August 17, 1973 in Constanta, Romania. He graduated from University of Constanta - Faculty of Mathematics and Computer Science in 1996 and obtained his PhD in 2005. Since 2006 he has held a lecturer position in Applied Mathematics at Ovidius University of Constanta, Romania - Faculty of Mathematics and Computer Science. His current research interests are: preconditioning techniques for finite element and finite differences discretizations of boundary value problems and iterative methods for solving sparse linear systems. He published 1 book, 1 paper in ISI quoted journal and more than 6 papers in other refereed journals or international conferences proceedings.

Constantin Popa was born on October 10, 1956 in Bucharest, Romania. He graduated the Faculty of Mathematics from University of Bucharest in 1981 and obtained his PhD in 1995. Since 2000 he has been a full professor in Applied Mathematics at the Faculty of Mathematics and Computer Science from Ovidius University of Constanta, Romania, where he is also heading the Department on Computer Science and Numerical Methods. His current research interests are: algebraic reconstruction techniques in Computerized Tomography, preconditioning techniques for finite element and finite differences discretizations of boundary value problems, iterative methods for least-squares formulations of linear systems of equalities and inequalities (projection algorithms), inverse problems - regularization techniques and methods for approximating the minimal norm solution of first kind integral equations. He published 4 books, 25 papers in ISI quoted journals and more than 40 papers in other refereed journals or international conferences proceedings. Some of his results have been cited in more than 25 papers.

1. Introduction

We shall consider in this paper the (stationary) 2D convection-diffusion-reaction problem

$$\begin{cases} -\Delta u + \gamma \dfrac{\partial u}{\partial x} + \delta u = f, \; in \; \Omega = (0,1)^2 \\ u = 0, \; on \; \partial\Omega \end{cases}, \quad (1)$$

where $\gamma > 0, \delta \geq 0$ and f are such that a unique solution exists. For $k \geq 2$ a fixed integer, we discretized the domain Ω,

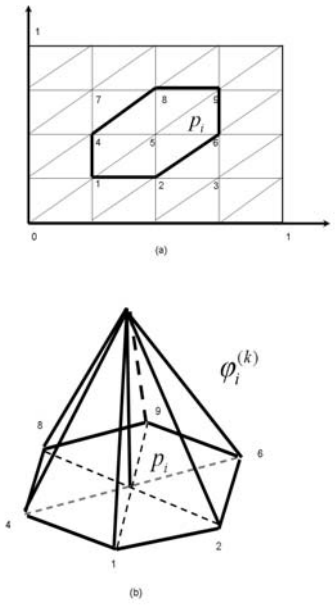

Figure1. Domain discretization.

with $n_k = (2^k - 1)^2$ uniformly distributed grid points and $h = \frac{1}{2^k}$ the mesh size of the discretization (see Figure 1(a) for *k=2*). We discretized the problem (1) by the Finite Element Method, using standard piecewise linear functions $\varphi_i^{(k)}$ as in Figure 1 (b) (see also for details [5, 6]). The $n_k \times n_k$ associated linear system

$$A_k x_k = b_k \tag{2}$$

is nonsymmetric, positive definite, big (because the approximation error is of order $O(h^2)$, which means that we need an enough big value of k - and thus of n_k, for an enough good approximation) and sparse (because of the special discretization of the problem domain in Figure 1(a) the matrix A_k has at most 7 nonzero entries on each row). From these view points, direct methods are not recommended for solving (2). And, unfortunately, the ''ill-conditioning'' of the system matrix A_k produces a (very) slow convergence for classical iterative solvers (as e.g. CG; see for details [2, 3, 4]). One way to overcome this difficulty is to consider preconditioning techniques for (2). Such a method will be described in the next section of the paper.

2. The Multilevel Preconditioning Method

The multilevel based preconditioning technique that we shall describe in this section is based on the results first proposed for symmetric elliptic boundary value problems by M. Griebel in [3]. In the paper [7] we extended them to the general nonsymmetric case for (1). In what follows we shall use the presentation and notations from this paper. The main idea comes from the field of multigrid methods (see e.g. [2, 4]). In this respect, we consider coarser discretization levels for Ω, with n_q nodes

$$n_q = (2^q - 1)^2, q = 1,2,\ldots,k. \tag{3}$$

For each q, $B_q = \{\varphi_1^{(q)},\ldots,\varphi_{n_q}^{(q)}\}$ is a basis of standard finite element functions, similar with $\varphi_i^{(k)}$ from Figure 1(b), but corresponding to the discretization level q. Let $V_q, q = 1,\ldots,k$ be the vector space generated by B_q, and $\hat{B}_k$, m_k given by

$$\hat{B}_k = B_1 \cup B_2 \cup \ldots \cup B_k,\ m_k = n_1 + n_2 + \ldots + n_k \tag{4}$$

The functions from $\hat{B}_k$ are linearly dependent and generate the subspace V_k. Each function $\varphi_j^{(q)} \in V_q \subset V_{q+1}$ has a unique representation as an element of V_{q+1}, of the form (see [2, 5])

$$\varphi_j^{(q)} = \sum_{i=1}^{n_q+1} c_{ij} \varphi_i^{(q+1)},\ j = 1,\ldots,n_q\ . \tag{5}$$

With the coefficients c_{ij} from (5) we can construct the $n_{q+1} \times n_q$ grid transfer matrices I_q^{q+1} by

$$(I_q^{q+1})_{ij} = c_{ij} \tag{6}$$

and define the $n_k \times n_q$ matrices S_q by

$$S_q^k = I_{k-1}^k I_{k-2}^{k-1} \ldots I_q^{q+1},\ q = 1,2,\ldots,k-1. \tag{7}$$

Then, the preconditioning $n_k \times m_k$ matrix S_k for the system (2) is given by (in block notation)

$$S_k = \left[\begin{array}{c|c|c|c|ccc} & & & & 1 & & \\ S_1^k & S_2^k & \ldots & S_{k-1}^k & & 1 & \\ & & & & & \ldots & \\ & & & & & & 1 \end{array}\right] \tag{8}$$

in which the right last block is the $n_k \times n_k$ unit matrix. The preconditioned version of the system (2) will be defined as

$$\hat{A}_k \hat{x}_k = b_k, \text{where } \hat{A}_k = S_k^T A_k S_k, b_k = S_k^T b \tag{9}$$

It results that the $m_k \times m_k$ matrix $\hat{A}_k$ is still nonsymmetric and positive semidefinite, but singular.

Remark 1. *The system (9) is compatible, with an infinity of solutions. But, for any of its solutions* $\hat{x}_k$ *we can recover the unique exact solution from (2) by the formula*

$$S_k \hat{x}_k = x_k . \tag{10}$$

In [7] we proved that the generalized spectral condition number of $\hat{A}_k$ from (10), defined by

$$cond(\hat{A}_k) = \left(\frac{\max\{\lambda, \lambda = nonzero\ eigenvalue\ of\ \hat{A}_k^T \hat{A}_k\}}{\min\{\lambda, \lambda = nonzero\ eigenvalue\ of\ \hat{A}_k^T \hat{A}_k\}} \right)^{1/2} \tag{11}$$

is bounded independently on the mesh size $h = \frac{1}{2^k}$. This allowed us in [7] to efficiently solve (9) by the CGLS algorithm from [2]. But all these results may become usefull in practical applications only with an efficient sparse matrix implementation of the construction of S_k and $\hat{A}_k$ in (8) and (9). This will be described in the next section of the paper.

3. Sparse Matrix Implementation of the Preconditioning Technique

In the following we shall present the construction of the preconditioned linear system. In this respect we designed a class of subroutines for organizing our problem data according to the sparse matrix commands of Matlab. The command to store a matrix in a Compressed Sparse Row format (CSR) in Matlab is *S = sparse(i, j, s, m, n, nzmax)*, where *i* and *j* are vectors indices which provide the positions and *s* is the vector of real entries which contains the stored values. Remaining quantities *m, n* and *nzmax* are scalar values which contain the number of rows, columns and maximum values for the nonzero entries. Matlab provides sparse matrix operations as concatenation, addition and matrix-vector multiplication which we will use to form the preconditioned matrix.

Starting from our data, i.e., discretization type which yields entries for $(A_k)_{ij}$ and $(b_k)_{ij}$ and also the structure of the intergrid transfer matrices I_q^{q+1}, we will describe how to determine parameters for the above sparse matrix command in order to store the preconditioned matrix $\hat{A}_k$ in CSR format. For a better understanding, we shall illustrate the presentation with an example in the 1D case of the problem (1). In this case, the intergrid operators from (8) are created as indicated in Figure 2 below.

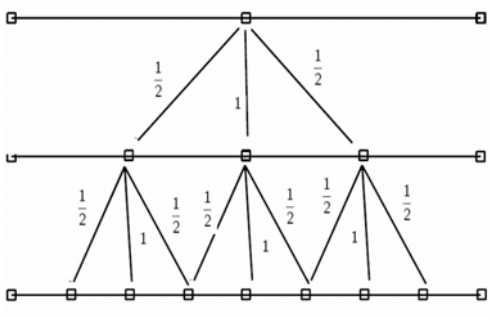

Figure 2. Three levels of discretization.

The finest level has 7 interior points, the next coarse level has 3 interior points and the coarsest level has 1 interior point. Thus, the structure of the preconditioning matrix S_k will be the one from Figure 3 below.

$$\begin{bmatrix} 1/4 \\ 1/2 \\ 3/4 \\ 1 \\ 3/4 \\ 1/2 \\ 1/4 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 1 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 3. Structure of the matrix S_k for two grids in 1D case.

where the first matrix from the right is the identity matrix, next is I_3^7 , and the last matrix is $I_3^7 I_1^3$. In the two dimensional case the construction of intergrid operators is more complex. Again, we will explain the construction of S_k using a particular case with three grids for a better understanding. Using discretization as in Figure (1), the preconditioning matrix S_k will be produced by repeating the 9-stencil shape

$$\begin{pmatrix} 0 & 1/2 & 1/2 \\ 1/2 & 1 & 1/2 \\ 1/2 & 1/2 & 0 \end{pmatrix} \quad (12)$$

in suitable positions corresponding to the discretization from Figure 1(a). We generate the matrix transfer operators defined in (6) in a recursive fashion and taking in account the simplified structure. Let *ia* and *ja* be the vectors of indices, that we used to create the sparse matrix I_q^{q+1} for an arbitrary fixed $q \in \{1,\ldots,k-1\}$, and *ma, na* its dimensions. The following routines generate the vectors *ia* and *ja* respectively.

1. Generating the *ia* vector.

```
for k=1:na
  iaux = k*ones(1,9);
  ia(9*(k-1)+1: 9*(k-1)+9) = iaux
endfor
```

where *iaux* is an auxiliary vector of 9 components used to fill *ia.*

2. Generating the *ja* vector.

```
t=1;
for k=1:sna
     vaux=[1+2*(n-1)*sma,2+2*(k-1)*sma,3+2*(k-1)*sma];
     jaux =[vaux, vaux+sma,vaux+2*sma];
    for k1 = 2:sna ;
      t = t+ j ;
      jac(t, :) = jac(t-1, :)+2 ;
    endfor
    t = t+1;
endfor
```

where *jaux* is the matrix in which we keep the values for *ja*. Furthermore, in the algorithm described for the computation of *ja* we have the three component vector *vaux* which is used to generate the the consecutive values in the *jaux*. When we increment the value of *t* we begin to build the corresponding *ja* for another matrix transfer operator. Furthermore, at this point we are ready to completely put together the vectors of indices by,

```
for k=1:na,
    ja(9*(k-1)+1: 9*(k-1)+9)=jac(k,:);
endfor
```

where for the entries of *ja* we exploit from *jac* each column stored in the previous sequence. As an example, in the particular case of three grid levels, with the finest one with 49×49 mesh points, in the Table 1 below we present the result provided by the two above presented subroutines for the intergrid matrix I_2^3 of dimensions 49×9.

Table 1. The vectors *ia* and *ja* for the 2D integrid operator I_2^3 of dimensions 49×9.

ia	1	1	1	1	1	1	1	1	1
ja	1	2	3	8	9	10	15	16	17
ia	2	2	2	2	2	2	2	2	2
ja	3	4	5	10	11	12	17	18	19
ia	3	3	3	3	3	3	3	3	3
ja	5	6	7	12	13	14	19	20	21

Finally, we are prepared to build the sparse matrix from the vector of entries *sa* which is formed using Matlab command *repmat,* which repeats the 9-stencil defined in (12) .

The sparse matrix *mS* is build using *sparse* command and then is added to the S_k in a recursive fashion using the following sequence

```
mS = sparse(ia,ja,sa,ma,na);
Sk = [mOld*mS',Sk];
mOld = mOld*mS';
```

4. Numerical Experiments

The computations were done using a Pentium 4 at 3Ghz processor, with 2 GB of RAM. In our first numerical experiment we have measured the time spent to assemble S_k in the case it was considered as a "noncompressed" matrix and compared this with the time spent to construct S_k as a sparse matrix for a fixed value *nlevel=6* which yields a matrix of size 5214×3969 (see Table 2). The time spent to form the uncompressed matrix S_k is 9.58e+02 seconds, compared with 4.21e-01 seconds corresponding to the sparse matrix construction. In Table 2 we present for different values of *nlevel* (the number of multigrid levels used), the size of S_k, the time (in seconds) t_S^F used to build S_k as a noncompressed matrix, and the time (in seconds) t_S^S used to build S_k as a sparse matrix. We can see the very big differences between the two values t_S^F and t_S^S. Moreover, starting from the value *nlevel=7* the matrix S_k cannot anymore be built as a noncompressed matrix in the computer memory because it is too large.

Table 2. Times computed for uncompressed (t_S^F) and sparse (t_S^S) allocation of S_k.

nlevel	Size of S_k	t_S^F	t_S^S
3	59x49	7.81e-02	6.25e-02
4	284x225	2.81e-01	3.21e-02
5	1245x961	1.38e+01	1.09e-01
6	5214x3969	9.58e+02	4.21e-01
7	21343x16129	Out of memory	3.01e+00
8	86368x65025	Out of memory	2.93e+01

In the second numerical experiment we did the same comparisons, but with respect to the construction of the preconditioned matrix $\hat{A}_k$ from (9). In Table 3 t_A^F and t_A^S represent the time (in seconds) for an "uncompressed" construction of $\hat{A}_k$ and a sparse one, respectively. Again we can see the big differences between t_A^F and t_A^S. Moreover, starting with the value *nlevel=7* we were "out of memory", as in the first numerical experiment (see Table 2, for the construction of S_k).

Table 3. Times computed for uncompressed (t_A^F) and sparse (t_A^S) allocation of $\hat{A}_k$.

nlevel	Size of $\hat{A}_k$	t_A^F	t_A^S
3	59x59	1.25e-01	8.25e-02
4	284x284	3.59e-01	1.21e-01
5	1245x1245	2.38e+01	4.09e-01
6	5214x5214	10.64e+02	5.34e-01
7	21343x21343	Out of memory	3.91e+00
8	86368x86368	Out of memory	4.12e+01

With $\hat{A}_k$ and $\hat{b}_k$ constructed as before in CSR format, we were able to write the CGLS code to solve the preconditioned system (9) up to the value nlevel=9 (which is enough for a good approximation of the solution of (1); see for details [7]). The CGLS algorithm uses in each iteration computations of type $\hat{A}_k z$ or $\hat{A}_k{}^T z$ which were implemented in terms of CSR Matlab's format.

REFERENCES

1. ANDREI, N., C. RASTURNOIU, **Sparse matrices and applications** (in romanian), Editura Tehnica, Bucuresti, 1983.
2. BJORCK, A., **Numerical Methods for Least Squares Problems,** SIAM Philadelphia, 1996.

3. BRIGGS, L. W. et al., **A Multigrid Tutorial,** SIAM Philadelphia, 1987.

4. GRIEBEL, M., **Multilevel algorithms considered as iterative methods on semidefinite Systems**, SIAM J. Sci. Comput., 15(3) (1994), pp. 547-565.

5. HACKBUSCH, W., E**lliptic Differential Equations. Theory and Numerical Treatment,** Springer-Verlag, Berlin, 1987.

6. MARCHOUK, G., V. AGOCHKOV, **Introduction aux Methodes des Elements Finis**, Editions MIR, Moscou, 1985.

7. ODEN, J. T., J.N. REDDY, **An Introduction to the Mathematical Theory of Finite Elements**, John Wiley and Sons, Inc., 1976.

8. NICOLA, A., C. POPA, **Preconditioning by an extended matrix technique for convection-diffusion-reaction equations**, to appear in Rev. Roum. d'Analyse Numer. et Theorie d'Approx.