Bull. Nov. Comp. Center, Num. Anal., 11 (2002), 75–86

Computation of a few smallest eigenvalues and their eigenvectors for large sparse SPD matrices*

M.R. Larin

The algorithm and the code for computing several eigenvalues and their corresponding eigenvectors of a large sparse symmetric positive definite (SPD) matrix, which arises as a result of grid approximations (FDM, FEM, or FVM) of multidimensional boundary value problems (BVPs) are described. The preconditioned inverse iteration (PINVIT) method is implemented by using the explicit incomplete factorization method with conjugate gradient acceleration for solving an auxiliary linear system of equations. The numerical results are presented and discussed.

1. Introduction

We consider the partial eigenvalue problems for computing p smallest eigenpairs $\{\lambda_i, \boldsymbol{u}_i\}$ of a very large sparse SPD matrix

$$\begin{aligned} &A\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i, \quad \|\boldsymbol{u}_i\| = 1, \quad 0 < \lambda_1 \le \lambda_2 \le \ldots \le \lambda_N, \\ &(\boldsymbol{u}_i, \boldsymbol{u}_j) = \delta_{ij}, \quad i,j = 1,\ldots,N, \end{aligned} \tag{1}$$

where δ_{ij} is the Kronecker symbol, $(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{u}^T\boldsymbol{v}$ and N is the order of the matrix corresponding to finite-difference, finite element, or finite volume (FDM, FEM, or FVM) approximations of the two- or the three-dimensional self-adjoint elliptic BVPs [1]

$$-\frac{\partial}{\partial x}\sigma_x\frac{\partial u}{\partial x} - \frac{\partial}{\partial y}\sigma_y\frac{\partial u}{\partial y}\Big(-\frac{\partial}{\partial z}\sigma_z\frac{\partial u}{\partial z}\Big) = \lambda u, \quad \|u\|_2 = 1, \quad (x,y,z) \in \Omega, \tag{2}$$

where the coefficients σ_x, σ_y (and σ_z) are positive and piecewise smooth functions in general. At different parts of the boundary $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ of the computational domain Ω, different types of the boundary conditions hold:

$$u|_{\Gamma_1} = 0, \qquad \frac{\partial u}{\partial n}\Big|_{\Gamma_2} = 0, \qquad \frac{\partial u}{\partial n} + \alpha u\Big|_{\Gamma_3} = 0. \tag{3}$$

The preconditioned inverse iteration method [2, 3] is used for solving problem (1). To solve an auxiliary linear system of equations with the matrix A, the explicit incomplete factorization method with conjugate gradient

*Supported by the Russian Foundation for Basic Research under Grant 01-07-90367.

acceleration (EXIFCG) is used (for details see [4]). The efficiency of the considered method is demonstrated by various numerical results for various model BVPs.

In Section 2, we present the description of the PINVIT algorithm and its parts. The main blocks of the code and some examples of their application are considered in Section 3. Finally, the numerical results are presented and discussed.

2. Description of algorithms

The idea of the considered approach for computing p smallest eigenvalues and their corresponding eigenvectors of the sparse SPD matrix is based on the effect of domination of the smallest eigenvalues as a result of repeated multiplications of the inverse matrix A^{-1} by m-dimensional subspaces, $p < m \ll N$. Unfortunately, the exact computation of A^{-1} by direct methods is the very memory and time consuming procedure, even by modern supercomputers. For example, in case of the two-dimensional BVPs (2), (3) the memory requirement increases as $O(N \log N)$, whereas the number of floating-point operations grows as $O(N^{3/2})$ (here we consider the nested dissection method [4] as the most efficient direct solver). To avoid the above problems, the preconditioning iterative technique is widely used.

One of the most robust preconditioned eigensolver is the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method, which has been suggested and analyzed by Knyazev [3]. The idea of the method for computing the first eigenvalue is based on the local optimization of the three-term reccurence by the Rayleigh–Ritz method on a three-dimensional subspace consisting of the previous iterate $\boldsymbol{v}^{(i-1)}$, the current iterate $\boldsymbol{v}^{(i)}$ and the preconditioned residual $\boldsymbol{w}^{(i)}$. The generalization to the block version is straightforward.

The basic scheme of the LOBPCG method can be described as follows:

Algorithm LOBPCG

Input: m starting vectors $\boldsymbol{v}_1^{(0)}, \ldots, \boldsymbol{v}_m^{(0)}$

Devices: to compute $A\boldsymbol{v}$ and $M\boldsymbol{v}$ for a given vector $\boldsymbol{v}$
to compute the scalar product $(\boldsymbol{v}, \boldsymbol{u})$ for given vectors $\boldsymbol{v}$ and $\boldsymbol{u}$

Select $\boldsymbol{v}_j^{(0)}$, $j = 1, \ldots, m$.
for $i = 0, 1, \ldots,$ until convergence
  for $j = 1, \ldots, m$
    $\mu_j^{(i)} = (A\boldsymbol{v}_j^{(i)}, \boldsymbol{v}_j^{(i)})/(\boldsymbol{v}_j^{(i)}, \boldsymbol{v}_j^{(i)})$
    $\boldsymbol{r}_j^{(i)} = \mu_j^{(i)} \boldsymbol{v}_j^{(i)} - A\boldsymbol{v}_j^{(i)}$
    $\boldsymbol{w}_j^{(i)} = M\boldsymbol{r}_j^{(i)}$

end
Use the Rayleigh-Ritz method on the trial subspace
Span$\{\boldsymbol{w}_1^{(i)}, \ldots, \boldsymbol{w}_m^{(i)}, \boldsymbol{v}_1^{(i)}, \ldots, \boldsymbol{v}_m^{(i)}, \boldsymbol{v}_1^{(i-1)}, \ldots, \boldsymbol{v}_m^{(i-1)}\}$
set $\boldsymbol{v}_j^{(i+1)}$ corresponds to j-th smallest Ritz vector,
$j = 1, \ldots, m$
end

Output: the approximations $\mu_j^{(i)}$ and $\boldsymbol{v}_j^{(i)}$ to the smallest eigenvalues λ_j and corresponding eigenvectors $\boldsymbol{u}_j$, $j = 1, \ldots, m$.

Here we have to note that in [3], a version of the LOBPCG method which is mathematically equivalent, but more numerically stable, has been proposed. The latter will be implemented in our code.

In the present paper, we offer to use an iterative method for solving the linear system of equations

$$A\boldsymbol{u}_j^{(i)} = \boldsymbol{r}_j^{(i)} \tag{4}$$

as an "implicit" preconditioner to the matrix A inside the LOBPCG method. Let the approximate solution of (4) be

$$\boldsymbol{w}_j^{(i)} = M\boldsymbol{r}_j^{(i)} = A^{-1}\boldsymbol{r}_j^{(i)} - \boldsymbol{e}_j^{(i)} = \boldsymbol{u}_j^{(i)} - \boldsymbol{e}_j^{(i)}, \tag{5}$$

where $\boldsymbol{e}_j^{(i)}$ is an error and M denotes the internal preconditioned conjugate gradient method matrix:

$$M = P_i(\tilde{A}), \quad \tilde{A} = U_B^{-1} A L_B^{-1}, \quad B = L_B U_B. \tag{6}$$

Here $L_B = U_B^t$ is the lower triangular matrix of the Cholesky factorization of matrix B, which corresponds to the explicit incomplete factorization method, see [4]. To provide a given accuracy $\varepsilon \ll 1$ for solution of (5), the number of internal iterations is proportional to $N^{5/4}$ for the 2D model BVPs and to $N^{7/6}$ for three-dimensional boundary value problems.

3. Program organization and data structure

The presented program package solves the partial eigenvalue problem (1), which arises as a result of the finite element approximations of BVPs (2), (3). The package consists of several subroutines, which generate and solve the partial eigenvalue problem by the algorithm LOBPCG. It implements a numerically stable variant of the Rayleigh–Ritz projection to find new eigenvector approximations and search directions, constructs incomplete factorization preconditioner and solves an auxiliary linear system of equations by the EXIFCG method. The programming language is Fortran-77.

The matrix A is stored in the sparse compact row-wise format [6] as follows:

`n` – the order of the matrix A,

`nnz` – the number of non-zero entries in the matrix A,

`JA` – the array of size `[1:n]`, in which the pointers to the first entry of each column are stored,

`AR` – the array of size `[1:nnz]`, in which all non-zero entries of the original matrix A are stored column by column, each column starting with its diagonal element, whereas the other off-diagonal entries follow their diagonal elements in any order,

`IA` – the array of size `[1:nnz]`, in which the corresponding row indices of all non-zero entries are stored.

For example, `JA(i)` points to the position of the diagonal entry of i-th column within the array `IA` (or `AR`), moreover, `IA(JA(i))=i`.

The eigenvalues and their corresponding eigenvectors are stored as follows:

`m` – the number of eigenvectors to be computed,

`p` – the number of eigenvectors used, `p` $\leq$ `m`,

`EigValues` – the array of size `[1:p]` for eigenvalue approximations,

`EigVectors` – the array of size `[1:n,1:p]` for eigenvector approximations.

Moreover, we use two user-defined parameters to compute the desired moment of process termination:

`stopeps` – the tolerance for accepting eigenpairs,

`maxits` – the maximum number of iterations to be performed.

Below we present an example of application of the package to compute one eigenpair (`p` = 1), using five iteration eigenvectors (`m` = 5) of the three-dimensional (`idim` = 3) BVP (2), (3) with $\sigma_x = 1.0$, $\sigma_y = 10^{-3}$ and $\sigma_z = 10^{-2}$ in a cube domain on the $(31 \times 31 \times 31)$ grid. Moreover, the accuracy of eigenpairs computed is 10^{-6} and `maxits` = 50.

```
      Program TEST
      m = 5                                  ! eigenvectors used
      p = 1                                  ! eigenvectors dèsired
```

Define mesh size and values of coefficients

```
      idim = 3                               ! dimension of BVP
      nx = 31                                ! nodes in x-direction
      ny = 31                                ! nodes in y-direction
      nz = 31                                ! nodes in z-direction
      sigmax = 1.0                           ! σx in (2)
```

```
      sigmay = 1.0d-3                          ! σ_y in (2)
      sigmaz = 1.0d-2                          ! σ_z in (2)
```

Matrix generation

```
      call GENMATR(idim, nx, ny, nz, sigmax, sigmay, sigmaz,
                   n, nnz, AR, IA, JA)   ! Output: matrix A
```

Define preconditioner

```
      call SETUP(n, nnz, AR, IA, JA, nu, D, NE, NEIB, AU)
                                         ! Output: matrix B
```

Define eigensolver parameters

```
      stopeps = 1.0d-6
      maxits = 50
```

Call eigensolver

```
      call EIGPCG(n, nnz, AR, IA, JA, nu, D, NE, NEIB, AU,
                  m, p, EigValues, EigVectors, stopeps, maxits)
                                    ! Output: p eigenpairs {μ_j, v_j}
      stop
```

The following subroutines are called here:

`GENMATR` computes the original matrix A in the sparse compact column-wise format. Note that one can simply read the original matrix from a data file instead of its generation;

`SETUP` computes the preconditioner matrix B. Note that one can use any type of the preconditioning technique, but *only* the output data structure for preconditioner depends on the preconditioner and its data structure used.

`EIGPCG` computes p smallest eigenpairs of the matrix A.

We present below the code of the subroutine `EIGPCG` with EXIFCG solver as preconditioner. Herein we use two additional arrays:

`Residuals` – the array of size `[1:n,1:p]`, in which vectors of the residual $\boldsymbol{r}_j^{(i)}$ and the pseudo-residual $\boldsymbol{w}_j^{(i)}$, are stored;

`SearchDirections` – the array of size `[1:n,1:p]`, in which vectors of search directions used instead of the old eigenvector approximations $\boldsymbol{v}_j^{(i-1)}$ during the Rayleigh–Ritz projection are stored.

```
      Subroutine EIGPCG(n, nnz, AR, IA, JA, nu, D, NE, NEIB, AU,
                        m, p, EigValues, EigVectors, stopeps,
                        maxits)
```

Initialization

```
      for i = 1, ..., m
          Residuals(:,i) = 0
```

```
        SearchDirections(:,i) = 0
        EigVectors(:,i) = random
    end
    call RITZ(n, nnz, AR, IA, JA, m, EigVectors, Residuals,
              SearchDirections)
    for i = 1, ..., m
        SearchDirections(:,i) = EigVectors(:,i)
    end
```

Apply one step of the steepest descent method

```
    nits = 1
    for i = 1, ..., m
        call MATVEC(n, nnz, AR, IA, JA, EigVectors(:,i),
                    WorkVector)
        EnergyNorm = ScalProd(n, EigVectors(:,i), WorkVector)
        EuclidNorm = ScalProd(n, EigVectors(:,i),
                              EigVectors(:,i))
        EigValues(i) = EnergyNorm / EuclidNorm
        call RESIDUAL(n, EigVectors(:,i), EigValues(i),
                      WorkVector, Residuals(:,i))
        call PRECOND(n, nu, D, NE, NEIB, AU, F, U,
                     Residuals(:,i))
    end
    call RITZ(n, nnz, AR, IA, JA, m, EigVectors, Residuals,
              SearchDirections)
```

Main cycle

```
    eps = 1.0
    while((eps > stopeps) or (nits ≥ maxits))do
        for i = 1, ..., m
            call MATVEC(n, nnz, AR, IA, JA, EigVectors(:,i),
                        WorkVector)
            EnergyNorm = ScalProd(n, EigVectors(:,i),
                                  WorkVector)
            EuclidNorm = ScalProd(n, EigVectors(:,i),
                                  EigVectors(:,i))
            EigValues(i) = EnergyNorm / EuclidNorm
            call RESIDUAL(n, EigVectors(:,i), EigValues(i),
                          WorkVector, Residuals(:,i))
            call PRECOND(n, nu, D, NE, NEIB, AU, F, U,
                         Residuals(:,i))
        end
        eps = Check(p, m, n, EigValues, Residuals)
```

```
      call RITZ(n, nnz, AR, IA, JA, m, EigVectors,
                Residuals, SearchDirections)
      nits = nits + 1
   end while
   return
```

The functions and subroutines called in EIGPCG are the following:

`Check(p, m, n, EigValues, Residuals)` computes a new value of the stopping estimate eps defined as follows

$$\texttt{eps} = \frac{\delta_{\text{actual}}}{\delta_{\text{initial}}} = \frac{\delta^{(i)}}{\delta^{(0)}}, \quad \delta^{(i)} = \max_{j_1, j_2, \ldots, j_p} |\texttt{Residuals}(j_k)|,$$

where the indices $j_1, j_2, \ldots, j_p$ correspond to p smallest values of the array `EigValues`;

`ScalProd` computes the scalar product of two vectors;

`MATVEC(n, nnz, AR, IA, JA, VectorV, VectorW)` computes the matrix-vector product `VectorW = A * VectorV`;

`RESIDUAL(n, VectorA, Value, VectorB, VectorC)` computes the *saxpy* product `VectorC = VectorA - Value * VectorB`;

`RITZ` computes `m` new eigenvector approximations and `m` new search directions by the Rayleigh–Ritz method;

`PRECOND(..., VectorInOut)` computes the pseudo-residual $w_j^{(i)}$ by the EXIFCG method. The array `VectorInOut` is the right-hand side as input, and the solution as output.

Note that a call for a subroutine `PRECOND` depends on the preconditioner used. For example, to call EXIFCG, we have to write down the following subroutine.

```
Subroutine PRECOND(n, nu, D, NE, NEIB, AU, F, U, Vector)
eps = 1.0d-12          ! accuracy of inner iterative solution
maxits = sqrt(n)       ! maximum number of inner iterations
theta = 1.0            ! compensation parameter (0 ≤ theta ≤ 1)
omega = 1.0            ! relaxation parameter (1 ≤ omega ≤ 2)
per = 1.0              ! perturbation parameter (per ≥ 1)
U = random             ! set random initial guess
F = Vector             ! set the right-hand side
call EXIFA(D, U, F, NE, n, NEIB, AU, nu, eps, maxits,
           theta, omega, per) ! solve MatrixA * U = F
Vector = U             ! put the solution in output array
return
```

A detailed description of the subroutine EXIFA and its input-output data can be found in [4].

To apply the Rayleigh–Ritz method, we have to define two standard subroutines:

ORTH(n, m, EigVectors, Residuals, SearchDirections) orthnormalizes all input vectors by the Gramm-Schmidt methods with respect to the energy scalar product;

JACOBI(m, RitzMatrix, RitzEigValues, RitzEigVectors) solves the complete eigenproblem with the Ritz matrix, stored in the array RitzMatrix of size [1:3m,1:3m], by the Jacobi method [5] and save the computed eigenpairs in the array RitzEigValues of size [1:3m] and RitzEigVectors of size [1:3m,1:3m].

Moreover, to compute new eigenvector approximations and search directions we need one extra subroutine:

SORT(m, n, Values, j_1, j_2, ..., j_m) computing m indices j_1, j_2, ..., j_m of m smallest values of the array Values[1:n], $m \leq n$.

Now the suitable Rayleigh–Ritz method for computing new eigenvectors approximations and new search directions can be described as follows:

```
    Subroutine RITZ(n, nnz, AR, IA, JA, m, EigVectors,
                    Residuals, SearchDirections)
Orthogonalization by the Gramm-Schmidt process
    call ORTH(n, m, EigVectors, Residuals, SearchDirections)
Compute Ritz matrix
    for i = 1, 2, ..., m
        call MATVEC(n, nnz, AR, IA, JA, EigVectors(:,i),
                    WorkVector)
        for j = 1, 2, ..., m
            RitzMatrix(i,j) = ScalProd(n, EigVectors(:,i),
                                       WorkVector)
            RitzMatrix(i,j+m) = ScalProd(n, Residuals(:,i),
                                         WorkVector)
            RitzMatrix(i,j+2m) = ScalProd(n,
                SearchDirections(:,i), WorkVector)
        end
        call MATVEC(n, nnz, AR, IA, JA, Residuals(:,i),
                    WorkVector)
        for j = 1, 2, ..., m
            RitzMatrix(i+m,j) = ScalProd(n, EigVectors(:,i),
                                         WorkVector)
```

```
            RitzMatrix(i+m,j+m) = ScalProd(n, Residuals(:,i),
                                           WorkVector)
            RitzMatrix(i+m,j+2m) = ScalProd(n,
                SearchDirections(:,i), WorkVector)
        end
        call MATVEC(n, nnz, AR, IA, JA, SearchDirections(:,i),
                    WorkVector)
        for j = 1, 2, ..., m
            RitzMatrix(i+2m,j) = ScalProd(n, EigVectors(:,i),
                                          WorkVector)
            RitzMatrix(i+2m,j+m) = ScalProd(n, Residuals(:,i),
                                            WorkVector)
            RitzMatrix(i+2m,j+2m) = ScalProd(n,
                SearchDirections(:,i), WorkVector)
        end
    end
```

Compute all eigenpairs of the Ritz matrix by the Jacobi method

```
    call JACOBI(m, RitzMatrix, RitzEigValues, RitzEigVectors)
```

Compute new eigenpairs and search directions

```
    call SORT(m, 3m, RitzEigValues, j1, j2, ..., jm)
    for i = j1, j2, ..., jm
        for j = 1, 2, ..., n
            sum = 0.0
            for k = 1, 2, ..., m
                sum = sum + Residuals(j, k) *
                            RitzEigVectors(k+m, i)
                          + SearchDirections(j, k) *
                            RitzEigVectors(k+2m, i)
            end
            SearchDirections(j, i) = sum
            for k = 1, 2, ..., m
                sum = sum + EigVectors(j, k) *
                            RitzEigVectors(k, i)
            end
            EigVectors(j, i) = sum
        end
    end
    return
```

4. Numerical results

Let the matrix A correspond to the piecewise-linear finite-element discretization of the two-(three)-dimensional BVPs (2), (3) in the square (cube) domain $\Omega = [0,1] \times [0,1] (\times [0,1])$ on a uniform Cartesian mesh $\mathcal{T}_h$ with stepsize $h = N^{-1}$. It is well known that this problem has an (analytical) exact solution.

In Table 1, we present the results for the eigenvalue problem for the first smallest eigenpair $\{\lambda_1, u_1\}$ by the LOBPCG method beginning with the random initial guess $v^{(0)}$ and continue the iterative process until the following stopping criterion

$$\frac{\|r_1^{(k)}\|}{\|r_1^{(0)}\|} < \varepsilon = 10^{-6}$$

is satisfied. All calculations were performed on an IBM-SP2 in the double precision.

Table 1. The number of iterations for LOBPCG method, $m = p = 1$

Coefficients	N						
	4	8	16	32	64	128	256
	(2D case)						
$\sigma_x = 1, \sigma_y = 1$	4	6	6	5	5	4	4
$\sigma_x = 1, \sigma_y = 10^{-1}$	7	10	8	8	7	7	5
$\sigma_x = 1, \sigma_y = 10^{-2}$	7	15	19	18	11	10	10
$\sigma_x = 1, \sigma_y = 10^{-3}$	7	21	29	38	26	26	26
	(3D case)						
$\sigma_x = 1, \sigma_y = 1, \sigma_z = 1,$	6	7	6	6	5	—	—
$\sigma_x = 1, \sigma_y = 1, \sigma_z = 10^{-1}$	12	12	10	8	7	—	—
$\sigma_x = 1, \sigma_y = 1, \sigma_z = 10^{-2}$	12	19	24	18	14	—	—
$\sigma_x = 1, \sigma_y = 1, \sigma_z = 10^{-3}$	11	25	32	34	32	—	—
$\sigma_x = 1, \sigma_y = 10^{-1}, \sigma_z = 10^{-1}$	11	11	9	7	7	—	—
$\sigma_x = 1, \sigma_y = 10^{-1}, \sigma_z = 10^{-2}$	18	22	20	14	12	—	—
$\sigma_x = 1, \sigma_y = 10^{-1}, \sigma_z = 10^{-3}$	22	35	44	31	29	—	—
$\sigma_x = 1, \sigma_y = 10^{-2}, \sigma_z = 10^{-2}$	14	22	22	17	15	—	—
$\sigma_x = 1, \sigma_y = 10^{-2}, \sigma_z = 10^{-3}$	25	46	50	32	29	—	—
$\sigma_x = 1, \sigma_y = 10^{-3}, \sigma_z = 10^{-3}$	13	28	43	38	35	—	—

From these numerical results one can see that the number of iterations for the LOBPCG method with the EXIF preconditioner does not depends on N or even decrease and only slightly depend on the anisotropy ratio. It happens due to the changing the ratio between the first and the second exact eigenvalues, which affects on the rate of convergence, when N is increased.

Next, we solve the same eigenproblem, but with several eigenvectors ($m = p > 1$) and with other stopping criteria ($\varepsilon = 10^{-3}$). In Table 2, we

Table 2. The number of iterations for the LOBPCG method and accuracy riched, $p = m = 1, \ldots, 5$

Parameter	m				
	1	2	3	4	5
NIters	2	10	10	10	10
$\lvert\lambda_1 - \mu_1\rvert$	$0.3921 \cdot 10^{-5}$	$0.2427 \cdot 10^{-3}$	$0.1184 \cdot 10^{-3}$	$0.9524 \cdot 10^{-4}$	$0.7795 \cdot 10^{-4}$
$\lvert\lambda_2 - \mu_2\rvert$	—	$0.4870 \cdot 10^{-12}$	$0.4984 \cdot 10^{-13}$	$0.2332 \cdot 10^{-15}$	$0.9268 \cdot 10^{-18}$
$\lvert\lambda_3 - \mu_3\rvert$	—	—	$0.1964 \cdot 10^{-11}$	$0.7409 \cdot 10^{-14}$	$0.8368 \cdot 10^{-16}$
$\lvert\lambda_4 - \mu_4\rvert$	—	—	—	$0.3129 \cdot 10^{-7}$	$0.9528 \cdot 10^{-11}$
$\lvert\lambda_5 - \mu_5\rvert$	—	—	—	—	$0.5464 \cdot 10^{-11}$
$\lVert Av_1 - \mu_1 v_1\rVert$	$0.1567 \cdot 10^{-2}$	$0.1699 \cdot 10^{-1}$	$0.1013 \cdot 10^{-1}$	$0.1091 \cdot 10^{-1}$	$0.1009 \cdot 10^{-1}$
$\lVert Av_2 - \mu_2 v_2\rVert$	—	$0.5717 \cdot 10^{-6}$	$0.1883 \cdot 10^{-6}$	$0.1380 \cdot 10^{-7}$	$0.8086 \cdot 10^{-9}$
$\lVert Av_3 - \mu_3 v_3\rVert$	—	—	$0.1281 \cdot 10^{-5}$	$0.7658 \cdot 10^{-7}$	$0.9405 \cdot 10^{-8}$
$\lVert Av_4 - \mu_4 v_4\rVert$	—	—	—	$0.1213 \cdot 10^{-3}$	$0.3360 \cdot 10^{-5}$
$\lVert Av_5 - \mu_5 v_5\rVert$	—	—	—	—	$0.1571 \cdot 10^{-5}$

present the number of iterations with respect of the number of eigenvectors used. Here by μ_i and v_i we denote the approximation to the exact eigenvalues and eigenvectors, λ_i and u_i, respectively. Correspondingly, $|\mu_i - \lambda_i|$ measures the difference between the exact and the computed eigenvalues and $\|Av_i - \mu_i v_i\|$ measures the quality of the approximated pair $\{\mu_i, v_i\}$.

Based on the results in Table 2, we conclude that the quality of eigenpair approximations is uniformly improved when m is increased. On the other hand, we have a problem with the convergence for the first eigenpair. Indeed, the number of iterations, denoted by `NIters`, is always equal to the maximum number of iterations (`maxits` = 10). The latter can be explained only by the special eigenvalue property of the preconditioned matrix under construction of which we use the equality $Ae = Be$, where $e = \{1\}$ is a unit vector.

References

[1] Il'in V.P. Finite Difference and Finite Volume Methods for Elliptic Equations. – Novosibirsk: ICM&MG, 2001 (in Russian).

[2] Knyazev A.V., Neymeyr K. A geometric theory for preconditioned inverse iteration, III: A short and sharp convergence estimate for generalized eigenvalue problems // Linear Algebra Appl. (to appear).

[3] Knyazev A.V. Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method // SIAM J. on Scientific Computing. – 2001. – Vol. 23, № 2. – P. 517–541.

[4] Il'in V.P. Iterative Incomplete Factorization Methods. – Singapore: World Scientific Publishing Co., 1992.

[5] Parlett B. The Symmetric Eigenvalue Problem. – Prentice-Hall, Inc., 1980.

[6] Pissanetzky S. Sparse Matrix Technology. – New York: Academic Press, 1984.