MATLAB Implementation of Element-Based Solvers

Leszek Marcinkowski[1] and Jan Valdman[2(✉)]

[1] Faculty of Mathematics, University of Warsaw, Warszawa, Poland
lmarcin@mimuw.edu.pl

[2] Faculty of Science, University of South Bohemia, České Budějovice, and The Czech Academy of Sciences, Institute of Information Theory and Automation, Prague, Czechia
jan.valdman@utia.cas.cz

Abstract. Rahman and Valdman (2013) introduced a vectorized way to assemble finite element stiffness and mass matrices in MATLAB. Local element matrices are computed all at once by array operations and stored in multi-dimensional arrays (matrices). We build some iterative solvers on available multi-dimensional structures completely avoiding the use of a sparse matrix.

Keywords: MATLAB code vectorization · Finite elements · Stiffness and mass matrices · Iterative solvers

1 Motivation Example

We solve a benchmark boundary value problem

$$-\triangle u + \nu u = f \qquad \text{on } x \in \Omega = (0,1) \times (0,1)$$

for given $f \in L^2(\Omega)$ and a parameter $\nu \geq 0$. Nonhomogeneous Dirichlet or homogeneous Neumann boundary conditions are assumed on parts of boundary $\partial\Omega$ and measure of the Dirichlet boundary has to be positive for $\nu = 0$. A finite element method is applied and leads to a linear system of equations

$$Au = (K + \nu M)u = b, \tag{1}$$

for an unknown vector $u \in \mathbb{R}^{n_n}$, where n_n denotes the number of mesh nodes (vertices). Stiffness and mass matrices $K, M \in \mathbb{R}^{n_n \times n_n}$ and the right hand side vector $b \in \mathbb{R}^{n_n}$ are defined as

$$K_{ij} = \int_\Omega \nabla\Phi_i \cdot \nabla\Phi_j \,\mathrm{d}x, \qquad M_{ij} = \int_\Omega \Phi_i \,\Phi_j \,dx, \qquad b_j = \int_\Omega f\,\Phi_j \,\mathrm{d}x \tag{2}$$

The work of the 1st author was partially supported by Polish Scientific Grant: National Science Center 2016/21/B/ST1/00350.
The work of the 2nd and the corresponding author was supported by the Czech Science Foundation (GACR), through the grant GA17-04301S.

using local basis functions Φ_i for $i = 1, \ldots, n_n$ and ∇ denotes the gradient operator. Figure 1 shows an example of a 2D discretization of Ω. Sparse matrices K, M are generated as

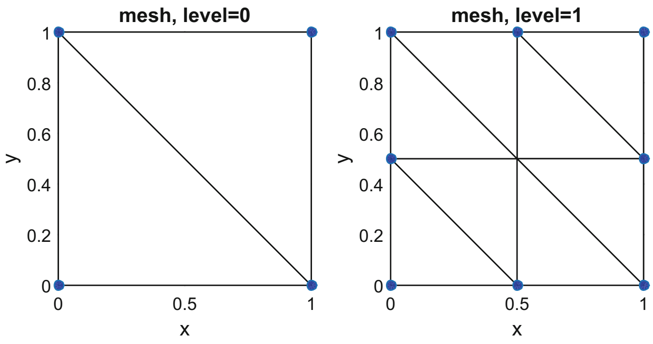

Fig. 1. Two examples of triangular meshes of a unit square domain Ω with $n_e = 2$ elements and $n_n = 4$ nodes (left) and $n_e = 8$ elements and $n_n = 9$ nodes (right).

$$K = \sum_{e=1}^{n_e} C_e^T K_e C_e, \qquad M = \sum_{e=1}^{n_e} C_e^T M_e C_e, \tag{3}$$

where n_e denotes a number of mesh elements (number of triangles in Fig. 1),

$$K_e, M_e \in \mathbb{R}^{n_b \times n_b}, \qquad e = 1, \ldots, n_e$$

are local element matrices and

$$C_e \in \mathbb{R}^{n_b \times n_n}, \qquad e = 1, \ldots, n_e$$

are Boolean connectivity matrices which distribute the terms in local element matrices to their associated global degrees of freedom. Here, n_b denotes a number of local basic functions. In the simplest case of nodal linear (P_1) finite elements:

$$\begin{aligned} n_b &= 3 \qquad \text{for triangles in 2D,} \\ n_b &= 4 \qquad \text{for tetrahedra in 3D.} \end{aligned}$$

Extensions to higher order isoparametric elements are also possible. All matrices K_e, M_e for $e = 1, \ldots, n_e$ are generated at once using vectorized routines of [3]. They are stored as 3-dimensional full matrices (see Fig. 2) of sizes

$$n_b \times n_b \times n_e.$$

The storage of 3-dimensional matrices contains certain memory overheads in comparison to sparse matrices (which can be automatically generated from

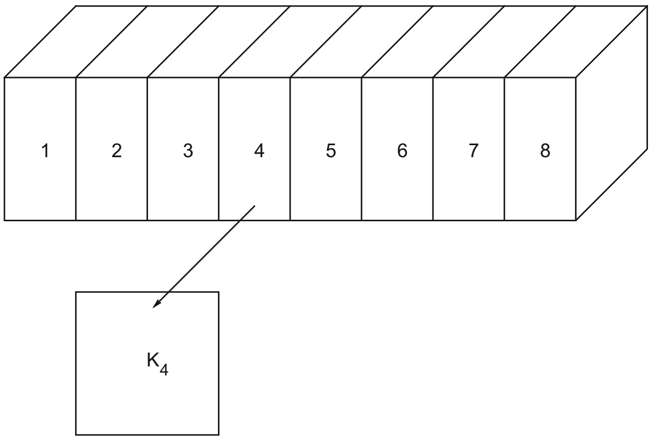

Fig. 2. Example of a 3-dimensional array storing all local stiffness matrices. The matrix corresponds to a triangular mesh with 8 elements displayed on Fig. 1 (right). A particular local stiffness matrix $K_4 \in \mathbb{R}^{3\times 3}$ is indicated.

them), since local contributions from restrictions of basis functions to shared elements are stored separately. Our aim is to build and explain in detail simple linear iterative solvers based on local element matrices K_e, M_e without assembling the sparse matrices M, K. This is our first attempt in this direction and therefore we show the possibility of this approach rather than efficient implementations and runtimes. The complementary software to this paper is available for download

https://www.mathworks.com/matlabcentral/fileexchange/70255.

2 Element-Based Solvers

Some examples of element-based iterative solvers are provided including their simple MATLAB implementations. All are based on a vectorized computation of a (column) residual vector

$$r := b - Ax \tag{4}$$

for a given approximation (column) vector $x \in \mathbb{R}^{n_n}$. The residual is computed using local matrices and local vectors

$$A_e := K_e + \nu M_e \in \mathbb{R}^{n_b \times n_b}, \quad b_e \in \mathbb{R}^{n_b}, \qquad e = 1, \ldots, n_e.$$

Matrices $R_e \in \mathbb{R}^{n_b \times n_n}, e = 1, \ldots, n_e$ are restriction matrices from global to local indices. Note that elementwise evaluations inside the loop (lines 2 and 3) operate with local matrices and local vectors only. A fully vectorized MATLAB version of Algorithm 1 follows:

Algorithm 1 residual computation - looped version

```
1: for $e = 1, \dots, n_e$ do
2:     $x_e = R_e x,$                  (restriction)
3:     $r_e = b_e - A_e x_e,$          (local residual)
4: end for
5: $r = \sum_{e=1}^{n_e} C_e^T r_e.$   (assembly)
```

```
function r=residual_e(A_e,bt_e,x,ind_e,indt)
x_e=x(ind_e);                                  %restriction - all
rt_e=bt_e-avtam(x_e,A_e);                      %residual - all
r=accumarray(indt(:),rt_e(:));                 %assembly - all
end
```

Clearly, matrices R_e and C_e of Algorithm 1 are not stored, but their operations are replaced by a convenient indexing using two index arrays:

$$\texttt{ind\_e} \in \mathbb{I}^{n_b \times 1 \times n_e}, \quad \texttt{indt} \in \mathbb{I}^{n_b \times n_e}.$$

Both arrays contain the same global nodes numbers corresponding to each element, but they are ordered differently with respect to their operations.

All objects indexed by elements are stored as full higher dimensional matrices and their names end with a symbol `_e`.

2.1 Richardson Iteration

We recall few examples of iterative methods based on a residual computation More details about them can be found eg. in [2,4]. One of the simplest iterative methods to solve (1) is the Richardson iteration for iterations $k = 0, 1, 2, \dots$ in the form

$$\begin{aligned} r^k &= b - Ax^k, \\ x^{k+1} &= x^k + \omega\, r^k \end{aligned} \tag{5}$$

with the initial column vector $x^0 \in \mathbb{R}^n$ and a given positive parameter $\omega > 0$. The optimal coefficient is equal to $\omega_{opt} = \frac{1}{\lambda_1 + \lambda_2}$ for $A = A^T > 0$, where λ_1 is the smallest and λ_2 the largest eigenvalue of A. For this ω_{opt} the convergence estimate is the fastest, i.e.

$$\|x^k - u\|_2 \leq \frac{\lambda_2 - \lambda_1}{\lambda_2 + \lambda_1} \|x^{k-1} - u\|_2.$$

Here, $u \in \mathbb{R}^n$ is the solution of (1). A MATLAB version follows:

```
function x=Richardson_e(A_e,bt_e,x0,ind_e,indt,iters,lam2,lam1,
    nd)
omega=2/(lam2+lam1);                             %optimal parameter
x=x0;                                            %iteration initial
for k=0:iters-1
    r=residual_e(A_e,bt_e,x,ind_e,indt);         %residual comput.
    r(nd)=0;                                     %dirichlet condit.
    x=x+omega*r;                                 %iteration update
end
end
```

2.2 Chebyshev Iteration

The Chebyshev polynomial (of the first kind) of degree $N \in \mathbb{N}_0$ is defined by

$$T_N(x) := \cos(N \arccos(x)), \qquad x \in [-1,1]$$

and it is known to have roots in the form

$$\alpha_k = \cos(\pi(k+1/2)/N), \qquad k = 0,\dots,N-1.$$

Consequently, a shifted and scaled polynomial

$$P_N(t) = T_N\left(\left(\frac{-2}{\lambda_2-\lambda_1}\right)\left(t - \frac{\lambda_1+\lambda_2}{2}\right)\right)/C_N, \qquad t \in [\lambda_1, \lambda_2],$$

with the scaling factor $C_N = T_N\left(\frac{\lambda_1+\lambda_2}{\lambda_2-\lambda_1}\right)$ satisfies the condition $P_N(0) = 1$. It also has N distinct roots

$$\alpha_k = \frac{\lambda_1+\lambda_2}{2} - \frac{\lambda_2-\lambda_1}{2}\cos\left(\frac{\pi(k+1/2)}{N}\right), \qquad k = 0,\dots,N-1$$

lying in (λ_1, λ_2). This polynomial has the smallest maximum norm on $[\lambda_1, \lambda_2]$ over all polynomials of degree less or equal N which are equal to one at zero.

Two-Level Chebyshev Iteration. The cyclic two-level Chebyshev iterative methods to solve (1) is in the form

$$\begin{aligned} r^k &= b - Ax^k, \\ x^{k+1} &= x^k + \alpha^{-1}_{k \pmod N} r^k. \end{aligned} \tag{6}$$

The method is convergent if all eigenvalues of A are contained in $[\lambda_1, \lambda_2] \subset (0, \infty)$. The optimal convergence is accessed where λ_1 is the minimal eigenvalue and λ_2 the maximal eigenvalue of A. Note that after N iterations we get

$$x^N - u = \Pi_{k=0}^{N-1}(I - \alpha_k^{-1}A)(x^0 - u) = P_N(A)(x^0 - u). \tag{7}$$

and then after ℓN iterations we get $x^{\ell N} - u = (P_N(A))^\ell (x^0 - u)$. Note that the Richardson iteration (5) is the special case of this method with $N = 1$. This error formula gives us,

$$\|x^N - u\|_2 \le \sum_{t \in [\lambda_1, \lambda_2]} |P_N(t)| \|x^0 - u\|_2 \le 2 \left(\frac{\sqrt{\lambda_2} - \sqrt{\lambda_1}}{\sqrt{\lambda_2} + \sqrt{\lambda_1}} \right)^N \|x^0 - u\|_2.$$

A MATLAB version follows:

```
function x=Chebyshev2Level_e(A_e,bt_e,x0,ind_e,indt,iters,lam2,
     lam1,N,nd)
d=(lam2+lam1)/2;  c=(lam2-lam1)/2;
k=0:N-1;  alphas=d+c*cos(pi*(1/2+k)/N);
x=x0;                                          %iteration initial
for k=0:iters-1
     r=residual_e(A_e,bt_e,x,ind_e,indt);      %residual comput.
     r(nd)=0;                                  %dirichlet condit.
     alpha=alphas(mod(k,N)+1);
     x=x+(1/alpha)*r;                          %iteration update
end
end
```

Three-Level Chebyshev Iteration. We now present the three-level Chebyshev iteration, cf. e.g. [2,4], The method is defined by the error equation, cf. also (7),

$$x^k - u = P_k(A)(x^0 - u), \qquad k = 0, 1, 2, \ldots \tag{8}$$

and its implementation is based on the following recurrence relation

$$T_k(t) = 2t\, T_{k-1}(t) - T_{k-2}(t), \qquad k > 1, \qquad T_1(t) = t,\; T_0(t) = 1.$$

This relation for t_k yields the recurrence formula for $k > 1$,

$$\begin{aligned} P_{k+1}(x) &= 2\frac{\lambda_1 + \lambda_2 - 2x}{\lambda_2 - \lambda_1} \frac{C_k}{C_{k+1}} P_k(x) - \frac{C_{k-1}}{C_{k+1}} P_{k-1}(x), \\ C_{k+1} &= 2\frac{\lambda_1 + \lambda_2}{\lambda_2 - \lambda_1} C_k - C_{k-1}, \end{aligned} \tag{9}$$

where

$$P_1(x) = C_1^{-1} \frac{\lambda_1 + \lambda_2 - 2x}{\lambda_2 - \lambda_1}, \qquad P_0 = 1, \qquad C_1 = \frac{\lambda_1 + \lambda_2}{\lambda_2 - \lambda_1}, \qquad C_0 = 1.$$

For $k = 1$ we get $x^1 - u = P_1(A)(x^0 - u)$ and

$$x^1 = u + \frac{\lambda_2 - \lambda_1}{\lambda_1 + \lambda_2} \frac{\lambda_1 + \lambda_2 - 2A}{\lambda_2 - \lambda_1}(x^0 - u) = x^0 + \frac{2}{\lambda_2 - \lambda_1} r_0,$$

where $r_0 = b - Ax^0$. Note that x^1 is computed as one iteration of the Richardson method applied to x^0 with the optimal coefficient, cf. (5). Our method is defined by (8), thus using the above recurrence relation we get for $k > 1$,

$$x^{k+1} - u = \frac{2C_k}{C_{k+1}}\left(\frac{\lambda_1+\lambda_2}{\lambda_2-\lambda_1}I - \frac{2}{\lambda_2-\lambda_1}A\right)(x^k - u) - \frac{C_{k-1}}{C_{k+1}}(x^{k-1} - u).$$

Since

$$1 = 2\frac{\lambda_1+\lambda_2}{\lambda_2-\lambda_1}\frac{C_k}{C_{k+1}} - \frac{C_{k-1}}{C_{k+1}}$$

we see that

$$x^{k+1} = \frac{2C_k}{C_{k+1}}\frac{\lambda_1+\lambda_2}{\lambda_2-\lambda_1}x^k + \frac{4}{\lambda_2-\lambda_1}\frac{C_k}{C_{k+1}}(b - Ax^k) - \frac{C_{k-1}}{C_{k+1}}x^{k-1}$$

and utilizing this identity once more we have the three level Chebyshev iterations

$$\begin{aligned} x^{k+1} &= x^k + \frac{C_{k-1}}{C_{k+1}}(x^k - x^{k-1}) + \frac{4}{\lambda_2-\lambda_1}\frac{C_k}{C_{k+1}}r_k, \qquad k > 1, \\ x^1 &= x^0 + \frac{2}{\lambda_2-\lambda_1}r_0 \end{aligned} \tag{10}$$

with $r_k = b - Ax^k$ $k = 0, 1, 2, \ldots$. We remind that the scaling factors C_k are defined by (9). Note that x^N in the both 2-level and 3-level iterations, cf. (6) and (10), are equal to each other what follows from (7) and (8). A MATLAB version reads:

```
function x=Chebyshev3Level_e(A_e,bt_e,x0,ind_e,indt,iters,lam2,
    lam1,nd)
d=(lam2+lam1)/2;  c=(lam2-lam1)/2;
x=x0;
r=residual_e(A_e,bt_e,x,ind_e,indt);              %residuum comput.
r(nd)=0;                                          %dirichlet condit.
for k = 0:iters-1
    z=r;
    if (k==0)
        p=z;  alpha=1/d;
    else
        beta=(c*alpha/2)^2;
        p=z+beta*p;  alpha=1/(d - beta/alpha);

    end
    x=x+alpha*p;
    r=residual_e(A_e,bt_e,x,ind_e,indt);          %residuum comput.
    r(nd)=0;                                      %dirichlet condit.
end
end
```

3 Numerical Experiments

We consider for simplicity the case of the square domain $\Omega = (0,1) \times (0,1)$, no mass matrix ($\nu = 0$) and nonhomogenous Dirichlet boundary conditions $u = 1$ for $x \in \partial\Omega$. For a uniformly refined triangular mesh (see Fig. 1) with n^2 nodes (also counting boundary nodes), there are $(n-2)^2$ eigenvalues of $A = K$ in the form

$$\lambda = 4\left(\sin^2 \frac{i\pi}{2(n-1)} + \sin^2 \frac{j\pi}{2(n-1)}\right), \quad i,j = 1,\ldots,n-2$$

and the minimal eigenvalue λ_1 is obtained for $i = j = 1$ and the maximal eigenvalue λ_2 for $i = j = n-2$. We utilize these eigenvalue bounds for all mentioned iterations methods. Furthermore, we assume a constant function $f = 1$ for $x \in \Omega$.

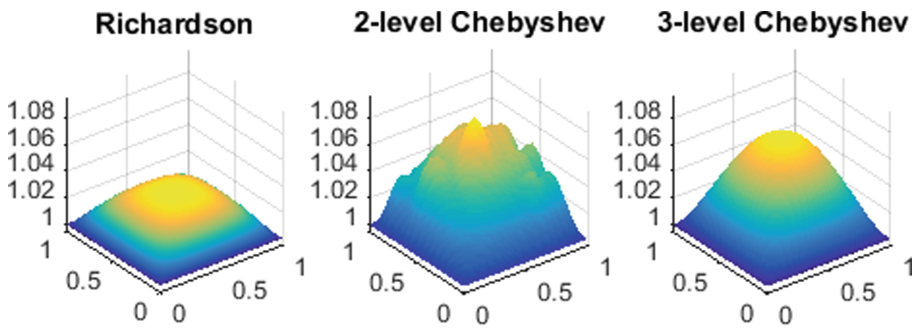

Fig. 3. Final iterates.

For a given number of iterations (we choose 124 iterations) and a (level 5) mesh with $1089 = 33^2$ nodes, final iterates are displayed in Fig. 3. Only the 3-level Chebyshev method converged optically to the exact solution. Richardson requires more steps to improve its convergence and the 2-level Chebyshev (with $N = 32$) demonstrates a known instability. The remedy to fix this instability would be to reorder values of precomputed parameters α_k to enhance the stability. Time performance is also reasonable for finer meshes. On a (level 10) mesh with $1050625 = 1025^2$ nodes, the assemblies of 3-dimensional arrays K_e, M_e take around 5 s each and 124 iterations take around 50 s for all iteration methods. The direct solver of MATLAB takes 5 s. Since number of iterations to obtain a convergence with respect to a given tolerance is known to grow as a function of condition number of finer meshes, we need to combine studied solvers with preconditioners or use several iterations of them as smoothers for instance in multigrid procedures.

Outlooks

We are interested in developing preconditioners for discussed solvers on multi-dimensional structures and extension to edge elements based on [1].

References

1. Anjam, I., Valdman, J.: Fast MATLAB assembly of FEM matrices in 2D and 3D: edge elements. Appl. Math. Computat. **267**, 252–263 (2015)
2. Hackbusch, W.: Iterative Solution of Large Sparse Systems of Equations. Applied Mathematical Sciences, vol. 95. Springer-Verlag, New York (1994). https://doi.org/10.1007/978-1-4612-4288-8. Translated and revised from the 1991 German original
3. Rahman, T., Valdman, J.: Fast MATLAB assembly of FEM matrices in 2D and 3D: nodal elements. Appl. Math. Computat. **219**, 7151–7158 (2013)
4. Samarskii, A.A., Nikolaev, E.S.: Numerical Methods for Grid Equations. Vol. II. Iterative Methods. Birkhäuser Verlag, Basel (1989). https://doi.org/10.1007/978-3-0348-9272-8. Translated from the Russian and with a note by Stephen G. Nash