Assignment on iterative solution methods

1. Computational exercise: Solving the Heat equation in 2D

The major aim of this exercise is to apply iterative solution methods and preconditioners when solving linear systems of equations as arising from discretizations of a partial differential equation of parabolic type.

Introduction: The heat conduction problem

Consider the time-dependent heat equation in two dimensions

\[ u_t = \Delta u + f(u, t), \ u = u(x, t), \ x \in \Omega = [0, 1]^2, \]

where \( \Delta u = \sum_{i=1}^{2} \frac{\partial^2 u}{\partial x_i^2} \).

For the test example we let \( f = 1, \ u(x, t) \mid_{x \in \partial \Omega} = 0 \) and as an initial condition we choose the discontinuous function (See Figure 1(a))

\[ u(x, 0) = \begin{cases} 
1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} < 0.2, \\
0 & \text{elsewhere}. 
\end{cases} \]

The problem is discretized in two steps.

(i) Space discretization

The space discretization is done using triangular mesh and the Finite Element method with linear basis functions. The spatial domain is discretized first with a coarse mesh, shown Figure 1(b). That mesh is then refined adaptively a couple of times to better suit the initial condition. An example of the mesh after three steps of adaptive refinement is shown in Figure 1(c).

As a result we obtain the so-called semi-discrete problem which constitutes a system of ODEs and has the following form:

\[ M \frac{\partial \mathbf{U}(t)}{\partial t} = K \mathbf{U}(t) + \mathbf{F}(t), \quad (1) \]

where \( \mathbf{U}(t) \) is the unknown solution, discretized in space and being now a function only of time \( t \). Similarly, \( \mathbf{F}(t) \) is the discrete counterpart of \( f(u, t) \).
Figure 1: Space discretizations
The matrices $M$ and $K$ are the corresponding standard mass and stiffness finite element matrices, which are symmetric positive definite and symmetric negative definite correspondingly.

Since space discretization falls out of the scope of this project, all related matrices are readily provided.

(ii) **Time discretization**

The time discretization has to be done using the so-called $\theta$ method. We recall that briefly the $\theta$ method for scalar equations. Given the initial value problem

$$y'(t) = f(t, y), \ t > 0, \ y(0) = y_0,$$

we replace it with the discrete equation

$$y^{(k+1)} = y^{(k)} + \delta t \left( (1 - \theta) f(t_k, y^{(k)}) + \theta f(t_{k+1}, y^{(k+1)}) \right), \ 0 \leq \theta \leq 1.$$

As we can see, $\theta = 0$ corresponds to the explicit Euler method, $\theta = 1$ corresponds to the implicit Euler method and $\theta = 0.5$ corresponds to Crank-Nicolson’s scheme (also referred to as the trapezoidal method).

Regarding stability of the above time-discretization scheme, theory says that for $\theta \in [0, 0.5)$ we have a conditionally stable scheme, where the time step $\delta t$ has to be related to the space discretization parameter ($h$), and for $\theta \in [0.5, 1]$ the time discretization scheme is unconditionally stable. It is also known that in order to balance in a best way the contributions of the space and time discretization errors into the global discretization error, $\theta$ should be chosen as $\theta = 0.5 + \xi$ for some small $\xi$ (cf. i.e., [1]).

In our case, the fully discretized system reads as follows:

$$(M - \theta \delta t K) U^{k+1} = (M + (1 - \theta) \delta t K) U^k + \delta t (\theta F^{k+1} + (1 - \theta) F^k), \ 0 \leq \theta \leq 1, \ (2)$$

where $U^{k+1}$ is the solution on time level $k+1$ to be computed and it is assumed that we already have obtained $U^k$.

Note, that since the initial solution is discontinuous, the choice $\theta = 0.5$ might lead to unphysical oscillations in the computed solution, in particular in the beginning of the simulation process.

**Problem description and tasks**

The aim is to test a preconditioned iterative method to solve the so-arising linear systems of equations with matrices $A = M - \theta \delta t K$. Since $-K$ and $M$ are symmetric positive definite (spd), then $A$ is also spd.

Observe that in the case when space discretization is done by the Finite Element method, even if we use explicit time discretization schemes, we always have to solve a linear system of equations
with the mass matrix $M$. This is not a severe restriction since a good approximation of $M$ is its diagonal part ($\text{diag}(M)$), cf. [3].

Below we describe the preconditioner for the matrix $A$ that has to be implemented in Matlab.

We assume that the matrix $A$ has a two-by-two block structure

$$
A = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}.
$$

The splitting is based on two consecutive (nested) mesh refinements. It is well-known (and can be checked with straightforward computation) that $A$ admits the following exact factorization:

$$
A = \begin{bmatrix}
A_{11} & 0 \\
A_{21} & S_A
\end{bmatrix} \begin{bmatrix}
I_1 & A_{11}^{-1}A_{12} \\
0 & I_2
\end{bmatrix},
$$

where $I_1$ and $I_2$ are identity matrices of a proper order and $S_A$ is the corresponding Schur complement matrix $S_A = A_{22} - A_{21}A_{11}^{-1}A_{12}$.

In practice, even if $A$ is sparse, its Schur complement is in general dense and costly to be formed explicitly, and therefore we do not want to compute it explicitly. We assume that we are able to construct some good enough approximation $S$ of $S_A$, which is also sparse. Further, we assume that we can approximate reasonably well the product $A_{11}^{-1}A_{12}$ by another sparse matrix $Z_{12}$. Then we consider the following preconditioner to $A$, based on the exact factorization (3):

$$
C = \begin{bmatrix}
A_{11} & 0 \\
A_{21} & S
\end{bmatrix} \begin{bmatrix}
I_1 & Z_{12} \\
0 & I_2
\end{bmatrix}.
$$

1. Write a MATLAB code which implements the following procedure:

For a given $\theta, \delta, M, K, A_{11}, A_{21}, S, Z_{12}, F$ perform ten timesteps, based on the discrete problem (2).

During each timestep solve the corresponding matrix $A$ using a suitable Krylov iterative method with a preconditioner $C$. The matrices $-K, M, A$ and $S$ are symmetric and positive definite.

To solve systems with $A$, use for instance \textsc{pcg} or \textsc{gmres} as implemented in MATLAB. You may try the unpreconditioned versions of the two methods, to start with.

For the preconditioned case write your own function to implement solutions of systems with the preconditioner $C$ from (4). To give an example, in MATLAB we call \textsc{pcg} with a preconditioner, defined in a separate function, as follows

$$
[u_{\text{next}}, \text{flag}, \text{relres}, \text{iter}, \text{resvec}] = \text{pcg}(A, \text{rhs}, \text{tol}, \text{maxit}, @\text{blkprec}, \ldots)
$$

and similarly for \textsc{gmres}. The MATLAB function

$$
\text{function } w = \text{blkprec}(v,A_{11},A_{21},S,Z_{12})
$$
should implement the solution of the system $Cw = v$. You should NOT form $C$ explicitly but use its block-factorized form.

Each solution with $C$ requires one solution with $A_{11}$ and one solution with $S$. As a first option, use MATLAB’s backslash operator to solve systems with both matrices. As a second option, use AGMG to solve with $A_{11}$ (and also with $S$). Check how the outer iterations depend on how accurately we solve with the $11$-block.

2. Should you use pcg or gmres to solve the linear system with $A$? Try both and give your arguments.

3. Observe the iterations per timestep, check the convergence history `resvec`.

4. Compare the performance of the unpreconditioned and the preconditioned solution method methods. Estimate the computational complexity and give your reasonings on whether it is relevant to use an involved preconditioner in this case.

5. Check how well $S$ approximates $S_A$, respectively, how well $C$ approximates $A$, and include a comment on that. This should be done for small matrix sizes, otherwise it requires much computation and computer resources. For instance, check the spectrum (the eigenvalues) of $A$ and how it is transformed after preconditioning:

```matlab
EA = sort(eig(full(A)));
EC = sort(eig(full(A),full(C)));
plot(A,'o');
plot(EC,'o');
```

Are the eigenvalues EC clustered? Does the product $C^{-1}A$ resemble the identity matrix? Do the same analysis for $S^{-1}S_A$.

**Remark 1:** In order to run the unpreconditioned iterative method, it suffices to use only the matrices $M$ and $K$, which do not depend on the choice of $\delta_t$ and $\theta$. Therefore, such experiments can be done with $\delta_t$ that is not predetermined (and, of course $\theta$).

For the preconditioned solver, the blocks $A_{11}$, $A_{21}$, $S$, $Z_{12}$ are constructed for a particular values of $\delta_t$ and $\theta$, which are included in the corresponding data files. For the cases where the blocks of the preconditioner are precomputed, $\delta_t = h_{\text{max}}$, where $h_{\text{max}}$ measures the largest triangle in the discretization mesh.

**Remark 2:** It could be profitable to provide a Matlab function that computes the action of $A$ on a vector (in order not to form it explicitly).

**Writing a report on the results**

The report has to have the following issues covered:

1. A brief problem description
2. Numerical experiments

Describe the experiments as consistently as possible. Include some relevant information on your choice - iteration counts, plots of residual histories, condition numbers (as a function of the size) etc. Is there a noticeable difference in the quality of the solution with respect to the parameter $\theta$? How does the number of iterations per time step grow with the problem size? Add a discussion on the suggested preconditioner - is it good (robust, computationally feasible etc.) or not and why. Comment on the relation between numerical stability (robust discretization scheme) and robust linear system solver.

The MATLAB code has to be attached to the report.

3. Conclusions

Data for the numerical experiments

All the input matrices are to be loaded from MATLAB .mat files with the following name convention:

```
Heat_in_N_theta_dt
```

where $N$ is the size of the matrix $A$, $\theta$ is the corresponding value of $\theta$ and $dt$ is the timestep. The latter is needed since the corresponding matrices $A_{11}$, $A_{21}$, $S$ and $Z_{12}$ do depend on both $\theta$ and $\delta_t$.

Each data file contains the following variables:

- $M$, $K$, $A_{11}$, $A_{21}$, $S$, $Z_{12}$, $FF$, $\theta$, $dt$, $u_0$
- $\text{lvl\_total}$, $\text{Node}$, $\text{Face\_Node}$

Some of the arrays are provided in order to enable you to plot the solution, which for example could be done as follows:

```matlab
clf
eval(['u0\_nr(0\' int2str(lvl\_total) '1) = u0;']) % Re-reorder
trisurf(Face\_Node(:,2)',Node(1,:),Node(2,:),u0\_nr,'facecolor','interp')
disp('Initial condition')
```

and

```matlab
eval(['u\_nr(0\' int2str(lvl\_total) '1) = u\_next;']) % Re-reorder
clf
trisurf(Face\_Node(:,2)',Node(1,:),Node(2,:),u\_nr,'facecolor','interp')
```

In this particular case the function $f(u, t)$ is constant and is contained in the vector $FF$.

The data files are downloadable from the course web-page http://user.it.uu.se/~maya/Courses/NLA/Projects/2014.
2. Theoretical exercise: Diagonal preconditioning

Consider the class of symmetric and positive definite matrices. It is widely accepted that a (pointwise) diagonal preconditioner can help to improve the convergence of an iterative method. Indeed, in some cases it may help a lot to reduce the number of iterations. In some other cases, such as for matrices with a constant main diagonal, the diagonal preconditioner does not help at all.

Provide some reasoning when one could expect a better convergence if a pointwise diagonal preconditioner is applied, namely, for a given spd matrix $A$, let $D$ be a diagonal matrix with the same elements as on the diagonal of $A$, what are the conditions to have

$$\text{cond}(D^{-1}A) < \text{cond}(A),$$

or, rather,

$$\text{cond}(D^{-1/2}AD^{-1/2}) < \text{cond}(A),$$

This exercise is not considered to be easy. There are some results related to searching the best (pointwise or blockwise) diagonal scaling of a matrix, where 'best' refers to a minimum condition number of the scaled matrix. It might be helpful to have a look at Theorem 4.3 in [2]. The paper can be downloaded via the course web-site http://user.it.uu.se/~maya/Courses/NLA/Projects/2014/VanderSluis1963.pdf.

Success!

References

