Numerical Linear Algebra2020-2021
Extra exercise session: preconditioners

This extra exercise session is intended to aid in studying the first two lectures on preconditioned iterative methods ('splitting methods' and 'Incomplete LU')

Part 1: Theory on linear methods

Any simple iterative method for the solution of a linear equation $A\mathbf{x} = b$, $A \in \mathbb{R}^{n\times n}$ can be written as a map

$$\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$

which we interpret as the procedure

$$\mathbf{x}^{(k+1)} := \Phi(\mathbf{x}^{(k)}, \mathbf{b}).$$

Note that we have assumed that the procedure for the generation of iterates does not change between iterations. Here and throughout the rest of this text it is assumed that A is invertible. A *linear* iterative method is one such that Φ is linear in $\mathbf{x}$ and $\mathbf{b}$. **Show that any linear iterative method is of the form $\Phi(\mathbf{x}, \mathbf{b}) = G\mathbf{x} + K\mathbf{b}$, with $G, K \in \mathbb{R}^{n\times n}$.** Next, we introduce the concept of a *consistent* method:

Definition 1. *An iterative method Φ for the equation $A\mathbf{x} = \mathbf{b}$ is called* consistent *if its solution $\mathbf{x}$ (i.e. $A\mathbf{x} = \mathbf{b}$) is a fixed point of $\Phi(., \mathbf{b})$. A linear method is called completely consistent if in addition $\mathbb{1} - G$ is non-singular.*

Argue that consistency is a vital property. Why is the final property above called 'complete consistency'?

Now prove the following, which shows that any linear completely consistent method is in fact derived from a splitting:

Theorem 1. *Any linear completely consitent iterative can be written as $G = M^{-1}N$, $K = M^{-1}$, with*

$$A = M - N$$

and M non-singular*.*

In short, a splitting method, given some splitting $A = M - N$ is the iterative procedure

$$\mathbf{x}^{(\mathbf{k+1})} := M^{-1}N\mathbf{x}^{(\mathbf{k})} + M^{-1}\mathbf{b}$$

Given a splitting $A = M - N$, how can you find the convergence rate of the associated linear method?

Preconditioned Conjugate Gradient

Suppose A is symmetric and positive-definite, and the unique solution to linear system is denoted to be $\boldsymbol{x}^*$ as before. Let's consider the quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top A\boldsymbol{x} - \boldsymbol{x}^\top \boldsymbol{b}.$$

The first derivative of the quadratic function is

$$\nabla f(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b},$$

so the unique solution $\boldsymbol{x}^*$ is a minimiser of $f(\boldsymbol{x})$. The derivation of the conjugate gradient algorithm are as follows. Let the initial guess be $\boldsymbol{x}_0$, the residual vector $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, and the search direction $\boldsymbol{p}_0 = \boldsymbol{r}_0$. The approximation at $j+1$ step can be expressed as

$$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j,$$

which leads to the residual vector at $j+1$ step satisfy

$$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j.$$

If $\boldsymbol{r}_j$'s are to be orthogonal, i.e.,

$$(\boldsymbol{r}_{j+1}, \boldsymbol{r}_j) = (\boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j, \boldsymbol{r}_j) = 0,$$

this leads to

$$\alpha_j = \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(A\boldsymbol{p}_j, \boldsymbol{r}_j)}.$$

As the next search direction $\boldsymbol{p}_{j+1}$ is a linear combination of $\boldsymbol{r}_{j+1}$ and $\boldsymbol{p}_j$, $\boldsymbol{p}_{j+1}$ can be expressed as

$$\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j.$$

Since $\boldsymbol{p}_j$'s are conjugate w.r.t A, i.e., $(\boldsymbol{p}_{j+1}, \boldsymbol{p}_j)_A = 0$, β_j can be obtained as

$$\beta_j = -\frac{(\boldsymbol{r}_{j+1}, A\boldsymbol{p}_j)}{(\boldsymbol{p}_j, A\boldsymbol{p}_j)},$$

in addition,

$$\alpha_j = \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(A\boldsymbol{p}_j, \boldsymbol{p}_j)}.$$

Utilising the recurrence relation of residual vectors $\boldsymbol{r}_j$, we further obtain

$$\beta_j = \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}.$$

These relations give us the conjugate gradient algorithm below.

Algorithm 1: Conjugate Gradient

```
1 Compute $\boldsymbol{r}_0 := \boldsymbol{b} - A\boldsymbol{x}_0$, $\boldsymbol{p}_0 := \boldsymbol{r}_0$.
2 for $j = 0, 1, \ldots,$ until convergence do
3     $\alpha_j := \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(A\boldsymbol{p}_j, \boldsymbol{p}_j)}$;
4     $\boldsymbol{x}_{j+1} := \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$;
5     $\boldsymbol{r}_{j+1} := \boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j$;
6     $\beta_j := \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}$;
7     $\boldsymbol{p}_{j+1} := \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$;
8 end for
```

Q1: Derive the following formulas,

$$\alpha_j := \frac{(\boldsymbol{r}_j, \boldsymbol{r}_j)}{(A\boldsymbol{p}_j, \boldsymbol{p}_j)}, \quad \beta_j = \frac{(\boldsymbol{r}_{j+1}, \boldsymbol{r}_{j+1})}{(\boldsymbol{r}_j, \boldsymbol{r}_j)}.$$

Q2: Given the convergence of CG

$$\|\boldsymbol{x}_j - \boldsymbol{x}^*\|_A \leq 2 \left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right]^j \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_A,$$

where κ is the spectral condition number $\kappa := \lambda_{\max}/\lambda_{\min}$, if we wish to perform enough iterations to reduce the norm of the error by a factor of ξ, what is the maximum number of iterations CG requires?

Q3: Suppose the preconditioner M is available in the form of incomplete Cholesky factorisation, i.e.,

$$M = LL^\top,$$

how can you split the preconditioning to preserve symmetry? How will the Split Preconditioner Conjugate Gradient look like?

Part 2: Some common splitting methods

The following definition is useful:

Definition 2. *Let Φ be a given iterative method. The weighted iterative method is defined by*

$$\Phi_\omega(\mathbf{x}, \mathbf{b}) := \omega\Phi(\mathbf{x}, \mathbf{b}) + (1-\omega)\mathbf{x}$$

Let L, D, U denote the strictly lower triangular, diagonal and strictly upper triangular part of the matrix A. **Implement the following common splitting methods in Matlab:**

(1) Jacobi, in which $M = D$

(2) Weighted Jacobi (WJ)

(3) Gauss-Seidel, in which $M = (D - L)$

(4) Succesive over-relaxation (SOR) defined by

$$\mathbf{x}^{(k+1)} = (D + \omega L)^{-1}\big(\omega\mathbf{b} - [\omega U + (\omega - 1)D]\mathbf{x}^{(k)}\big).$$

Test your routines on the given sparse matrices (make sure you have '`mmread`' in your folder) and Poisson matrix of size 100 (use `Matlab`'s `gallery`). Make sure that your routines maximally exploit sparsity of the matrices involved. Is SOR as formulated here the weighted version of Gauss-Seidel?

For the weighted methods, you can use the following heuristics for the optimal parameters:

(1) $\omega_{WJ} := \frac{2}{\lambda_n(D^{-1}A)+\lambda_n(D\text{[illegible]}}$

(2) $\omega_{SOR} := \frac{2d}{d+\sqrt{\lambda_1(A)\lambda_n(A)}}$

with $d := \max_i A_{ii}$ and eigenvalues ordered smallest to largest. In practice, these heuristics are slight overestimates of the actual optimal parameter. You should take this into account.

Compare the performance of the four methods outlined above using Matlab. Make plots of the convergence, in terms of the residues $\|A\mathbf{x}^{(k)} - \mathbf{b}\|$. Add the theoretical convergence rates to your figure. What do you observe?

Part 3: Incomplete factorizations

Another powerful technique for the preconditioning of linear systems is *incomplete factorization.* This section explores the simplest: $ILU(0)$, $ILU(p)$ and modified ILU $(MILU)$.

The basic strategy of ILU methods is to produce an approximate LU decomposition of a given matrix, where some restriction on the sparsity structure of the factors L, U is imposed. The prototypical example is $ILU(0)$, or incomplete LU-factorization without fill-in. The basic algorithm is algorithm 1.

Algorithm 1: Incomplete LU-factorization without fill-in (ILU(0))

```
input  : Matrix $A$ with sparsity pattern $S(A)$
output: Approximate $LU$-decomposition such that $S(L), S(U) \subset S(A)$
for $i = 2 : n$ do
    for $k = 1 : i-1 \wedge (i,k) \in S(A)$ do
        $A(i,k) := A(i,k)/A(k,k)$;
        for $j = k+1 : n \wedge (i,j) \in S(A)$ do
            $A(i,j) := A(i,j) - A(i,k) * A(k,j)$;
```

The above in fact produces a matrix whose upper triangular part corresponds to U, and the strictly lower triangular part is sufficient to represent L, because it is assumed that L is unit lower triangular. As you can see, the sparsity pattern of A is imposed on L and U.

Suppose now that we say that an element that has magnitude $\mathcal{O}(\epsilon^p)$ is assigned 'level' p. If we say that the level of A_{ij} is l_{ij}, we have, by the update formula

$$A_{ij} = A_{ij} - A_{ik}A_{kj}$$

that the level of the updated elment A_{ij} should be of the order of $\min\{l_{ij}, l_{ik} + l_{kj} + 1\}$. This is the basic principle behind $ILU(p)$, the incomplete LU-factorization with level-of-fill k.

Algorithm 2: Incomplete LU-factorization with fill-in (ILU(p))

```
input  : Matrix $A$ with sparsity $S(A)$, maximal level $p$
output: Approximate $LU$-decomposition of $A$
begin
    set $l_{S(A)} := 0$ and $l_{S(A)^c} := \infty$;
for $i = 2 : n$ do
    for $k = 1 : i-1 \wedge l_{ik} \leq p$ do
        $A(i,k) := A(i,k)/A(k,k)$;
        for $j = k+1 : n \wedge (i,j) \in S(A)$ do
            $A(i,j) := A(i,j) - A(i,k) * A(k,j)$;
            $l_{ij} := \min\{l_{ij}, l_{ik} + l_{kj} + 1\}$;
    for $j = 1 : n \wedge l_{ij} > p$ do
        $A(i,j) := 0$;
```

Implement both ILU(0) and ILU(p) in Matlab. Be sure to make maximal use of Matlab's colon operator. Use the resulting factors (looking only at ILU(p), $p = 0, 1$ will suffice) as preconditioners in a preconditioned Conjugated Gradient (CG). Also use your splitting methods as preconditioners to CG. For Gauss-Seidel you have to use so-called symmetric Gauss-Seidel. Use MAtlab's built-in `pcg`. As before, plot the residuals. Note: your Matlab ILU code will probably be too slow to be practical for large matrices. Instead, you can use Matlab's built-in ilu and the openly available 'ILUk' at `https://nl.mathworks.com/matlabcentral/fileexchange/48320-ilu-k-preconditioner`. Also compare with modified LU ($MILU$) and standard conjugate gradient. Plot the residuals. What do you observe? Why is the order of convergence better than for the splitting methods?

You have seen a sufficient condition for ILU to work, namely that A is an M-matrix (not to be confused with the M-matrix of a linear iterative scheme as we defined it above!). An M-matrix is defined as a matrix whose leading principle minors are all positive. **Implement a routine that checks recursively whether a matrix is an M-matrix. Are the matrices that you were given M-matrices?**