March 26, 2018

Lecture 10 – Preconditioning, Software, Parallelisation

A Incorporating a preconditioner

We are interested in solving

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{10.1}$$

for $\mathbf{x}$. Here, $\mathbf{A}$ is an $n \times n$ non-singular matrix and $\mathbf{b}$ is a n-vector. Both $\mathbf{A}$ and $\mathbf{b}$ are given. Often convergence of iterative solvers improves dramatically by solving an alternative system as

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}. \tag{10.2}$$

Here, $\mathbf{M}$ is a carefully selected $n \times n$ non-singular matrix. Systems for $\mathbf{M}$ should be easily solvable. In Lecture 6.B, we learnt that the conditioning of the matrix $\mathbf{A}$, nor the distribution of the eigenvalues determine the convergence of an iterative method: GMRES may converge arbitrarily slowly even for unitary matrices $\mathbf{A}$ and also for matrices $\mathbf{A}$ with all eigenvalues clustered around 1. On the other hand GMRES may converge quickly for matrices with some eigenvalue (relatively) close to 0, in particular, for matrices that are ill-conditioned. However, the examples that illustrate these observations are rather academical. In practice, iterative methods often converge well for well-conditioned systems. Therefore, the traditional point of view on the idea of applying an iterative method to (10.2) rather than to (10.1) is that the multiplication by $\mathbf{M}^{-1}$ improves the conditioning of the linear matrix. Therefore, (10.2) is referred to as the **preconditioned linear system** and $\mathbf{M}$ (or $\mathbf{M}^{-1}$) is called the **preconditioner** of $\mathbf{A}$.

If the idea is the improve the condition number, then it may be useful to realise that, in case $\mathbf{A}$ is ill-conditioned, a preconditioned matrix $\mathbf{M}^{-1}\mathbf{A}$ can be well-conditioned only if $\mathbf{M}$ is ill-conditioned as well, because

$$\frac{\mathcal{C}_2(\mathbf{A})}{\mathcal{C}_2(\mathbf{M})} \leq \mathcal{C}_2(\mathbf{M}^{-1}\mathbf{A}) \leq \mathcal{C}_2(\mathbf{M})\mathcal{C}_2(\mathbf{A}).$$

(Check this and conclude that $\mathcal{C}_2(\mathbf{M})$ is large whenever $\mathcal{C}_2(\mathbf{A})$ is large and $\mathcal{C}_2(\mathbf{M}^{-1}\mathbf{A})$ is of modest size).

We observe that in practice, the distribution of the eigenvalues of the matrix is a better indication on the succes of iterative solvers than the condition number. The eigenvalues should cluster away form 0 for fast convergence. We rather like to view preconditioning as a mean to improve the distribution of eigenvalues

In (10.2), the system is obtained by multiplying (10.1) from the left by $\mathbf{M}^{-1}$. Therefore, this way of preconditioning is called **left preconditioning**. **Right preconditioning**,

$$\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b} \quad \text{and} \quad \mathbf{x} = \mathbf{M}^{-1}\mathbf{y} \tag{10.3}$$

and **two-sided preconditioning**, with $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$,

$$\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}\mathbf{y} = \mathbf{M}_1^{-1}\mathbf{b} \quad \text{and} \quad \mathbf{x} = \mathbf{M}_2^{-1}\mathbf{y} \tag{10.4}$$

is also possible and may have advantages. As we will see in the next exercise, concerning the distribution of the eigenvalues, it does not matter whether we use left, right or two-sided preconditioning. Although preconditioning is essential for the success of iterative methods, in practice, the way the preconditioner is used does not play a role in the speed of convergence: except for one or two steps, an iterative method needs the same number of steps for left, as for right and two-sided preconditioning (assuming the same initial guess).

Exercise 10.1.

(a) Show that the eigenvalues of the preconditioned matrix $\mathbf{M}^{-1}\mathbf{A}$ are solutions of the generalised eigenvalue problem $\mathbf{A}\mathbf{x} = \lambda\mathbf{M}\mathbf{x}$.

(b) Show that the preconditioned matrices $\mathbf{M}^{-1}\mathbf{A}$, $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ with $\mathbf{M} \equiv \mathbf{M}_1\mathbf{M}_2$, and $\mathbf{A}\mathbf{M}^{-1}$ have the same spectrum. Often, in applications, $\mathbf{M}_1$ is lower triangular and $\mathbf{M}_2$ is upper triangular.

Exercise 10.2. Consider the GCR method. We take the trivial vector $\mathbf{0}$ as initial guess.

The extension vector $\mathbf{u}_k$ in standard GCR is selected as $\mathbf{u}_k = \mathbf{r}_k$.[1]

(a) Show that if the extension vector $\mathbf{u}_k$ is selected to satisfy $\mathbf{A}\mathbf{u}_k = \mathbf{r}_k$, then $\mathbf{r}_{k+1} = \mathbf{0}$, that is, $\mathbf{x}_{k+1}$ solves $\mathbf{A}\mathbf{x} = \mathbf{b}$.

(b) Let $\mathbf{M}$ be a preconditioner. Let the extension vector $\mathbf{u}_k$ be the solution of $\mathbf{M}\mathbf{u}_k = \mathbf{r}_k$ (**implicit preconditioning**). Show that the kth residuals of this version of GCR (called PGCR) equals the kth residual of standard GCR (GCR) applied to the right preconditioned problem $\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$. Show also that, if $\mathbf{y}_k$ is the kth approximate solution computed in this method, then $\mathbf{x}_k = \mathbf{M}_k^{-1}\mathbf{y}_k$ is the kth approximate of PGCR: GCR for $\mathbf{A}\mathbf{M}^{-1}\mathbf{y} = \mathbf{b}$ computes "preconditioned approximates".

(c) Consider the left preconditioned problem $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$. Show that GCR applied to this preconditioned problem computes 'preconditioned residuals' and 'unpreconditioned approximates'. Discuss the relation of the kth true residual of GCR for $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$ and the kth residual of PGCR (see also (b) of Exercise 10.1).

Exercise 10.3. Let $\mathbf{A}$ be $n \times n$ Hermitian. Let $\mathbf{M}$ be an Hermitian preconditioner for the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$. CG allows implicit preconditioning (PCG); see Exercise 7.2.

(a) Is $\mathbf{A}\mathbf{M}^{-1}$ Hermitian (with respect to the standard inner product)? Disprove (give a 2×2 counter example) or prove the claim "there is a decomposition $\mathbf{M} = \mathbf{M}_1\mathbf{M}_2$ of $\mathbf{M}$ such that $\mathbf{M}_1^{-1}\mathbf{A}\mathbf{M}_2^{-1}$ is Hermitian".

Assume $\mathbf{M}$ is positive definite.

(b) Show that $\mathbf{M}^{-1}\mathbf{A}$ is self adjoint (Hermitian) with respect to the $\mathbf{M}$-inner product and $\mathbf{A}\mathbf{M}^{-1}$ is self adjoint with respect to the $\mathbf{M}^{-1}$-inner product.

(c) Prove that CG with implicit preconditioning (PCG) is standard CG for the right (or left?) preconditioned problem $\mathbf{A}\mathbf{M}^{-1}\mathbf{x} = \mathbf{b}$, but with respect to the $\mathbf{M}^{-1}$-inner product (or $\mathbf{M}$-inner product?) instead of the standard inner product.

(d) If $\mathbf{M} = (\mathbf{L}+\mathbf{D})\mathbf{D}^{-1}(\mathbf{D}+\mathbf{L}^*)$ is a preconditioner for some positive definite diagonal matrix $\mathbf{D}$, then we need $\mathbf{D}^{\frac{1}{2}}$ if we want to apply CG (with standard inner product) to an explicitly preconditioned system. Why? Do we need $\mathbf{D}^{\frac{1}{2}}$ for PCG?

(e) What form of preconditioning is used in the MATLAB code `pcg.m` for CG? (Hint: perform in MATLAB the command `type pcg` and inspect the code.)

The following exercise (Exercise 10.4(b)-(c)) explains how to incorporate a Hermitian positive definite preconditioner $\mathbf{M}$ in a symmetric solver as MINRES. The approach exploits $\mathbf{M}$, but is (mathematically) equivalent to applying standard MINRES to (7.1) (see Exercise 10.4(e). It relies on the fact that $\mathbf{M}^{-1}\mathbf{A}$ is self adjoint with respect to the $\mathbf{M}$-inner product (see Exercise 7.2 and Exercise 10.3).

Exercise 10.4. **Preconditioned MINRES.**

(a) Show that Lanczos applied to $\mathbf{M}^{-1}\mathbf{A}$ with respect to the $\mathbf{M}$-inner product and $\mathbf{v}_1$ a multiple of $\mathbf{M}^{-1}\mathbf{b}$ (and $\mathbf{v}_0 = \mathbf{0}$) produces a sequence $(\mathbf{v}_j)$ of n-vectors such that

$$\beta_{k+1}\mathbf{v}_{k+1} = \mathbf{M}^{-1}\mathbf{A}\mathbf{v}_k - \alpha_k\mathbf{v}_k - \beta_k\mathbf{v}_{k-1} \tag{10.5}$$

[1] We refer to the vector $\mathbf{u}_k$ in GCR before the orthogonalisation of $\mathbf{c}_k \equiv \mathbf{A}\mathbf{u}_k$ as the **extension vector**. This is the vector by which the search subspace is extended. We call the vector $\mathbf{u}_k$ after the orthogonalisation loop, the **update vector**. It is the vector from the search subspace that is used for updating $\mathbf{x}_k$.

with scalars α_k and β_k such that

$$\mathbf{v}_k^* \mathbf{M} \mathbf{v}_j \quad (j \neq k) \quad \text{and} \quad \mathbf{v}_k^* \mathbf{M} \mathbf{v}_k = 1 \quad (k = 0, 1, 2, \ldots). \tag{10.6}$$

(b) Generate two sequences $(\mathbf{u}_k)$ and $(\mathbf{v}_k)$ of n-vectors with $\mathbf{u}_1$ a multiple of $\mathbf{b}$ ($\mathbf{u}_0 = \mathbf{0}$) such that

$$\beta_{k+1} \mathbf{u}_{k+1} = \mathbf{A}\mathbf{v}_k - \alpha_k \mathbf{u}_k - \beta_k \mathbf{u}_{k-1} \quad \text{and} \quad \mathbf{M}\mathbf{v}_{k+1} = \mathbf{u}_{k+1},$$

with scalars α_k and β_k such that

$$\mathbf{v}_k^* \mathbf{u}_j \quad (j \neq k) \quad \text{and} \quad \mathbf{v}_k^* \mathbf{u}_k = 1 \quad (k = 0, 1, 2, \ldots).$$

Prove that the vectors $\mathbf{v}_j$ and the scalars are the same as produced in (a). Conclude that

$$\mathbf{A}\mathbf{V}_k = \mathbf{U}_{k+1} \underline{T}_k \quad \text{with} \quad \mathbf{U}_k = \mathbf{M}\mathbf{V}_k,$$

where $\mathbf{V}_k$ and $\mathbf{U}_k$ are the $n \times k$ matrices with columns the first k $\mathbf{v}_j$ and $\mathbf{u}_j$, respectively. $\underline{T}_k$ is the $(k+1) \times k$ tri-diagonal matrix with entries the α_j and β_j.

(c) Let $\underline{T}_k = \underline{Q}_k R_k$ be the QR-decomposition of $\underline{T}_k$. Take

$$\mathbf{x}_k \equiv \mathbf{x}_0 + \rho_0 (\mathbf{V}_k R_k^{-1})(\underline{Q}_k{}^* e_1) \quad \text{with} \quad \rho_0 \equiv \sqrt{\mathbf{b}^* \mathbf{M} \mathbf{b}}. \tag{10.7}$$

Show that $\mathbf{x}_k$ minimises $\|\mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\widetilde{\mathbf{x}})\|_M$ over all $\widetilde{\mathbf{x}}$ in $\mathbf{x}_0 + \text{span}(\mathbf{V}_k)$, $\text{span}(\mathbf{V}_k) = \mathcal{K}_k(\mathbf{M}^{-1}\mathbf{A}, \mathbf{M}^{-1}\mathbf{b})$. That is, the residual $\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ is minimal with respect to the $\mathbf{M}^{-1}$-norm.

(d) Modify ALG. 7.3 to include preconditioning following the suggestions in (b) and (c).

(e) Take $\mathbf{w}_1$ a multiple of $\mathbf{L}^{-1}\mathbf{b}$ (and $\mathbf{w}_0 = \mathbf{0}$). Apply Lanczos to the problem in (7.1) to find a sequence $\mathbf{w}_j$ or orthonormal vectors and a tri-diagonal matrix $\underline{S}_k$ such that

$$(\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-*})\mathbf{W}_k = \mathbf{W}_{k+1}\underline{S}_k$$

Show that

$$\underline{S}_k = \underline{T}_k \quad \text{and} \quad \mathbf{W}_k = \mathbf{L}^* \mathbf{V}_k = \mathbf{L}^{-1}\mathbf{U}.$$

Show, if $\mathbf{z}_k = \text{argmin} \|\mathbf{L}^{-1}\mathbf{b} - (\mathbf{L}^{-1}\mathbf{A}\mathbf{L}^{-*})\widetilde{\mathbf{z}}\|_2$ where we minimise over all $\widetilde{\mathbf{z}} \in \text{span}(\mathbf{W}_k)$, then $\mathbf{x}_k = \mathbf{L}^{-*}\mathbf{z}_k$.

(f) What form of preconditioning is used in the MATLAB code `minres.m`?

Note that the preconditioner here is required to be positive definite. As explained in Exercise 7.3, it suffices for preconditioned CG to have an Hermitian preconditioner. The above approach, or to be more precise, the above version of Lanczos, also may work of $\mathbf{M}$ is Hermitian, but indefinite (check that the "M-orthogonality" conditions in can be fulfilled with the three term recurrence of Exercise 10.4(a)). However, as for CG with such a general preconditioner, the process may break down ("$\mathbf{v}_k^* \mathbf{M} \mathbf{v}_k$ can be zero") and residuals are not minimized with respect to some norm.[2]

Consider the decomposition $\mathbf{A} = \mathbf{L}_A + \mathbf{D}_A + \mathbf{U}_A$ of $\mathbf{A}$ into a strictly lower triangular matrix $\mathbf{L}_A$, a diagonal matrix $\mathbf{D}_A$ and a strictly upper triangular matrix $\mathbf{U}_A$. This leads to so-called D-ILU preconditioners:

$$\mathbf{M} \equiv (\mathbf{L}_A + \mathbf{D})\mathbf{D}^{-1}(\mathbf{D} + \mathbf{U}_A). \tag{10.8}$$

Here, $\mathbf{D}$ is some suitable non-singular diagonal matrix as $\mathbf{D} = \mathbf{D}_A$. In §B.3, we will suggest other, more effective choices for $\mathbf{D}$. For the Eisenstat trick, to be discussed in the next exercise, Exercise 10.5, it suffices to know that $\mathbf{D}$ is non-singular. This trick is an efficient *implementation* of the *action* of D-ILU preconditioners. This trick combines a preconditioning step with a matrix vector multiplication essentially at the same cost as a matrix-vector multiplication only.

[2] Actually, the PCG version of ALG. 7.1 is an efficient variant of Bi-CG (the shadow system is not explicitly needed). Similarly, this MINRES version can be viewed as an efficient variant of QMR.

Exercise 10.5. **Eisenstat trick.** With $\mathbf{D}$ and $\mathbf{M}$ as in (10.8), consider the problem

$$(\mathbf{I}+\widetilde{\mathbf{L}})^{-1}\widetilde{\mathbf{A}}(\mathbf{I}+\widetilde{\mathbf{U}})^{-1}\mathbf{y}=\widetilde{\mathbf{b}}, \tag{10.9}$$

where

$$\widetilde{\mathbf{L}}\equiv\mathbf{D}^{-1}\mathbf{L}_A, \qquad \widetilde{\mathbf{A}}\equiv\mathbf{D}^{-1}\mathbf{A}, \qquad \widetilde{\mathbf{U}}\equiv\mathbf{D}^{-1}\mathbf{U}_A, \quad \text{and} \quad \widetilde{\mathbf{b}}\equiv(\mathbf{I}+\widetilde{\mathbf{L}})^{-1}\mathbf{D}^{-1}\mathbf{b}. \tag{10.10}$$

(a) Suppose $\mathbf{y}$ solves (10.9). Show that the solution $\mathbf{x}$ of $(\mathbf{I}+\widetilde{\mathbf{U}})\mathbf{x}=\mathbf{y}$ solves $\mathbf{A}\mathbf{x}=\mathbf{b}$.

(b) Show that the spectrum of $\mathbf{M}^{-1}\mathbf{A}$ and the operator $(\mathbf{I}+\widetilde{\mathbf{L}})^{-1}\widetilde{\mathbf{A}}(\mathbf{I}+\widetilde{\mathbf{U}})^{-1}$ are the same: (10.9) uses a two-sided preconditioning of $\mathbf{A}$ with $\mathbf{M}$. Discuss the consequences for convergence.

To compute $\mathbf{c}\equiv(\mathbf{I}+\widetilde{\mathbf{L}})^{-1}\widetilde{\mathbf{A}}(\mathbf{I}+\widetilde{\mathbf{U}})^{-1}\mathbf{u}$ for a given vector $\mathbf{u}$, introduce $\Delta\equiv\mathbf{D}^{-1}\mathbf{D}_A-2\mathbf{I}$ and proceed as follows:

$$\begin{aligned}&\text{Solve } \mathbf{u}' \text{ from } (\mathbf{I}+\widetilde{\mathbf{U}})\mathbf{u}'=\mathbf{u}\\ &\text{Solve } \mathbf{u}'' \text{ from } (\mathbf{I}+\widetilde{\mathbf{L}})\mathbf{u}''=\Delta\mathbf{u}'+\mathbf{u}\\ &\text{Compute } \widetilde{\mathbf{c}}=\mathbf{u}'+\mathbf{u}''\end{aligned} \tag{10.11}$$

(c) Write $\widetilde{\mathbf{A}}=\Delta+\mathbf{I}+\widetilde{\mathbf{L}}+\mathbf{I}+\widetilde{\mathbf{U}}$ and show that $\widetilde{\mathbf{c}}=\mathbf{c}$.

Approach (10.11) for computing $\mathbf{c}\equiv(\mathbf{I}+\widetilde{\mathbf{L}})^{-1}\widetilde{\mathbf{A}}(\mathbf{I}+\widetilde{\mathbf{U}})^{-1}\mathbf{u}$ is known as the **Eisenstat trick**. Note that this trick is not a preconditioner: it is a way of *implementing* the preconditioner of (10.8).

(d) Recall that we are interested in the case where $\mathbf{A}$ is sparse.
Show that solving $\mathbf{M}\widetilde{\mathbf{c}}=(\mathbf{L}_A+\mathbf{D})(\mathbf{I}+\widetilde{\mathbf{U}})\widetilde{\mathbf{c}}=\mathbf{c}$ for $\widetilde{\mathbf{c}}$ is exactly as expensive (in terms of flops) as a matrix-vector multiplication $\mathbf{c}=\mathbf{A}\mathbf{u}$.
Show that (10.11) is only $2n$ flop more expensive than a matrix-vector multiplication $\mathbf{c}=\mathbf{A}\mathbf{u}$, assuming the matrices $\widetilde{\mathbf{L}}$, $\widetilde{\mathbf{U}}$ and Δ are available.

(e) How do you incorporate the Eisenstat trick in an Krylov subspace method (How and where in the code do you compute $\widetilde{\mathbf{L}}$, $\widetilde{\mathbf{U}}$, Δ, $\widetilde{\mathbf{b}}$, $\mathbf{x}$)?

(f) We want to use the preconditioner $\mathbf{M}$ from (10.8). We consider to incorporate it in a straight forward way as $\mathbf{M}^{-1}\mathbf{A}$, or $(\mathbf{D}+\mathbf{L}_A)^{-1}\mathbf{A}(\mathbf{D}+\mathbf{U}_A)^{-1}\mathbf{D}$, or with the Eisenstat trick. Do you expect a gain in computational time by incorporating the Eisenstat trick in a long recurrence type of Krylov subspace method (as GMRES and GCR)? What about short recurrence methods (as Bi-CGSTAB)?

Assume $\mathbf{A}$ is Hermitian positive definite. Work with $\widetilde{\mathbf{A}}=\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ to obtain a symmetric variant.

(g) How much faster do you expect the CG code to be with this symmetric variant of the Eisenstat trick as compared to using $\mathbf{M}$ as an implicit preconditioner (you may use that $\mathbf{A}$ has [on average] nnz non-zeros in each row and take, for instance, $nnz=7$).

B Preconditioners

Let $\mathbf{A}=(A_{ij})$ be an $n\times n$ matrix.
Let $\mathbb{I}$ be the set of all matrix indices (i,j) $(i,j=1,\ldots,n)$.

In this subsection, we discuss some specific types of preconditioners. The general rule is that the more effective the preconditioner is in reducing the number of iterations steps of a (Krylov) subspace method, the more costly it is to construct the preconditioner (that is, the preprocess stage will be more expensive) and the more expensive it is to solve preconditioned systems (the steps of the iterative process get more expensive): less steps, but slower steps. What the most effective preconditioner is (in terms of overall computational time) depends on

the problem[3] (even on the problem size)[4] and also on the architecture of the computer that is to be used.

B.1 Incomplete LU-decompositions

An important class of preconditioners is based on sparse approximations of $\mathbf{A}$ that are easy to invert. More specifically, these preconditioners $\mathbf{M}$ are obtained in *factorized form*, $\mathbf{M} = \mathbf{LU}$, with sparse factors $\mathbf{L}$ and $\mathbf{U}$ of lower and upper triangular matrices, respectively, that approximate the corresponding factors of the complete LU-decomposition of $\mathbf{A}$. In this subsubsection, we explain the general ideas behind these so-called incomplete LU preconditioners. Then, in §§B.2-B.3, we discuss some more specific examples.

Complete LU-decompositions. The LU-decomposition process (Gaussian elimination) constructs a sequence $(\mathbf{A}_j)$ of $n \times n$ matrices as follows.

$\mathbf{A}_1 \equiv \mathbf{A}$
for $j = 1, \ldots, n-1$
1) $\nu_j \equiv \mathbf{e}_j^* \mathbf{A}_j \mathbf{e}_j$ (the jth pivot)
2) ℓ_j be the n-vector defined by
$\mathbf{e}_i^* \ell_j = 0$ if $i \leq j$, $\mathbf{e}_i^* \ell_j = \dfrac{\mathbf{e}_i^* \mathbf{A}_j \mathbf{e}_j}{\nu_j}$ if $i > j$
($\nu_j \ell_j$ coincides with the jth column of $\mathbf{A}_j$ below its diagonal)
3) $\mathbf{A}_{j+1} = (\mathbf{I} - \ell_j \mathbf{e}_j^*)\mathbf{A}_j$
Put $\mathbf{U} \equiv \mathbf{A}_n$, $\widetilde{\mathbf{L}} \equiv [\ell_1, \ldots, \ell_{n-1}, \mathbf{0}]$, $\mathbf{L} \equiv \mathbf{I} + \widetilde{\mathbf{L}}$.

Then, $\widetilde{\mathbf{L}}$ is strictly lower triangular, $\mathbf{U}$ is upper triangular, and $\mathbf{A} = \mathbf{LU}$ is the LU-decomposition of $\mathbf{A}$, in the context of preconditioners, also called **complete LU-decomposition** of $\mathbf{A}$. Note that we did not incorporate a pivoting strategy, i.e., we implicitly assumed that all pivots ν_j are non-zero.

The matrix $\mathbf{A}$ of interest will be sparse. Efficiency requires a careful implementation of the above elimination process: multiplications by 0 result in 0 and should not be included in the computational steps. A similar remark applies to the incomplete elimination processes that we discuss below. Even if such trivial multiplications are excluded from the computational process, and even if $\mathbf{A}$ is sparse, the computation of the L and U-factors $\mathbf{L} = (L_{ij})$ and $\mathbf{U} = (U_{ij})$ of $\mathbf{A}$ can be unacceptable expensive, due to **fill**, also called **fill-in**, that is, there are locations $(i, j) \in \mathbb{I}$ for which the matrix entry $A_{ij} = 0$, while L_{ij} or U_{ij} is non-zero. In formula, if

$$\mathbb{S}(\mathbf{B}) \equiv \{(i, j) \in \mathbb{I} \,\Big|\, B_{ij} \neq 0\}$$

is the **sparsity pattern** or **sparsity set** of the $n \times n$ matrix $\mathbf{B} = (B_{ij})$, then the fill-in of the LU-decomposition of $\mathbf{A}$ equals $\mathbb{S}(|\mathbf{L}| + |\mathbf{U}|) \backslash \mathbb{S}(\mathbf{A})$. Reordering equations and unknowns (pivoting) (that is, replace $\mathbf{A}$ by $\mathbf{A}(J_r, J_c)$ for some permutations J_r and J_c of $(1, 2, \ldots, n)$ of rows and columns, respectively) may reduce fill. But often not enough to obtain a feasible decomposition process. Nevertheless, reordering can be an effective strategy also for incomplete elimination processes (as we learnt in b) of Exercise 8.6. See also, for instance, §B.6).

Incomplete LU-decompositions. An **ILU** (incomplete LU) **decomposition** of $\mathbf{A}$ consists of a *sparse* lower triangular matrix $\mathbf{L}$ and a *sparse* upper triangular matrix $\mathbf{U}$ that, in some sense, approximate the complete L and U-factors of $\mathbf{A}$: both $\mathbf{L}$ and $\mathbf{U}$ are non-singular $n \times n$ matrices. The idea is that the construction of these sparse L and U-factors is cheap (as compared to the

[3] If, for instance, the system $\mathbf{Ax} = \mathbf{b}$ has to be solved for the same matrix, but many right-hand side vectors $\mathbf{b}$, then the costs of the preprocess stage may get less dominant.

[4] Often the number of iteration steps increases if the problem size grows. With the ideal preconditioner, the total costs should be proportional to the problem size (proportional to the number of unknowns).

costs of computing the complete LU-decomposition; recall that trivial multiplications should be excluded from the computational process), and solving systems involving these sparse matrices $\mathbf{L}$ and $\mathbf{U}$ is also cheap (with costs similar to, or a modest multiple of, the costs of a multiplication by $\mathbf{A}$). Then $\mathbf{M} \equiv \mathbf{LU}$ will be the preconditioner. Of course the preconditioner $\mathbf{M}$ will be stored in factorized form, and 'Solve $\mathbf{Mu} = \mathbf{r}$ for $\mathbf{u}$' will be implemented as first 'solve $\mathbf{M}\widetilde{\mathbf{u}} = \mathbf{r}$ for $\widetilde{\mathbf{u}}$', followed by '$\mathbf{Mu} = \widetilde{\mathbf{u}}$ for $\mathbf{u}$' (see also Exercise 10.5).

Dropping strategies. Often[5] incomplete LU-decompositions are constructed using a **dropping strategy** or **dropping operator**, that is, a map Π that maps $n \times n$ matrices $\mathbf{B}$ to $n \times n$ matrices $\Pi(\mathbf{B})$ where $\Pi(\mathbf{B})$ equals $\mathbf{B}$ at certain (i,j)-entries and 0 at the other entries. Though Π need not be linear on the space of all $n \times n$ matrices, it 'projects': $\Pi(\Pi(\mathbf{B})) = \Pi(\mathbf{B})$. What entries are put to zero (are dropped) depends on the dropping strategy (and can also depend on the matrix $\mathbf{B}$). Once a dropping strategy has been selected an incomplete LU-decomposition is obtained by by extending line 3) in the above scheme for obtaining complete L and U-factors to

$$3') \qquad \widetilde{\mathbf{A}}_{j+1} = (\mathbf{I} - \ell_j \mathbf{e}_j^*)\mathbf{A}_j, \quad \mathbf{A}_{j+1} = \Pi(\widetilde{\mathbf{A}}_{j+1}).$$

Note that, in the computation of $\widetilde{\mathbf{A}}_{j+1}$, only the rows of $\mathbf{A}_j$ change that correspond to non-zero coordinates of ℓ_j. The standard dropping operators will require an update only of these rows, also in cases where the operator is not linear.

ILU. An important class of incomplete decompositions is obtained by first fixing a sparsity set, that is, a (sparse) subset $\mathbb{F}$ of $\mathbb{I}$. Then drop matrix entries outside $\mathbb{F}$, that is, the dropping operator Π is defined by

$$\Pi(\mathbf{B})_{ij} \equiv B_{ij} \;\text{ if }\; (i,j) \in \mathbb{F} \quad \text{and} \quad \Pi(\mathbf{B})_{ij} \equiv 0 \;\text{ if }\; (i,j) \notin \mathbb{F}.$$

Note that this operator Π is linear. $\mathbb{F}$ is known before the start of the incomplete elimination process. It may depend on the matrix $\mathbf{A}$ for which preconditioning is required, but not on the intermediate matrices (as $\widetilde{\mathbf{A}}_{j+1}$) of the incomplete elimination process. $\mathbb{F}$ limits the set of entries at which fill is permitted. Therefore, the computations of the (k,m)-entries of $\widetilde{\mathbf{A}}_{j+1}$ in step 3') can be skipped for the (k,m) not in this set (and for $k < j, m < j$).

A popular choice for $\mathbb{F}$ is $\mathbb{F} \equiv \mathbb{S}(\mathbf{A}) \equiv \{(i,j) \,\big|\, A_{i,j} \neq 0\}$ (for other related choices of $\mathbb{F}$, see §B.2 below). Note that with this choice, the dropping strategy leads to incomplete L and U factors with (in lower and upper triangular part) the same (or smaller) sparsity pattern as $\mathbf{A}$ and the combination of 'solve $\mathbf{L}\widetilde{\mathbf{u}} = \mathbf{r}$ for $\widetilde{\mathbf{u}}$' and 'solve $\mathbf{U}\mathbf{u} = \widetilde{\mathbf{u}}$ for $\mathbf{u}$' is more or less as costly as 'computing $\mathbf{c} = \mathbf{Au}$' by matrix-vector multiplication.

The first incomplete LU-decompositions in literature relied on this dropping operator and sparsity set. The name 'ILU preconditioner' often refers to this specific incomplete decomposition.

Drop tolerance. Another popular strategy drops small matrix entries. A **drop tolerance** parameter $\tau > 0$ is fixed and all matrix entries below this value are dropped: $\Pi(\mathbf{B})_{ij} = 0$ whenever $|B_{ij}| < \tau$, else $\Pi(\mathbf{B})_{ij} = B_{ij}$. Smallness can also be measured in some relative sense: $|B_{ij}| < \tau \|\mathbf{B}\|_{\mathrm{M}}$ rather than $|B_{ij}| < \tau$ (or $|B_{ij}| < \tau \|\mathbf{e}_i^*\mathbf{B}\|_2$ or $|B_{ij}| < \tau\sqrt{|B_{ii}B_{jj}|}$).

Note that now the fill pattern is not known in advance, which makes coding a bit more tricky. Moreover, the resulting incomplete factors need not be sparse. Therefore, this drop tolerance strategy is usually used in combination with a fixed sparsity set (see also §B.4).

Exercise 10.6. **Incomplete LU.** Costs (flops en memory)

(a) ...

Modified ILU-decompositions. Let Π be a dropping operator. A **modified** variant is obtained by adding (also called **lumping**) the dropped elements to the diagonal: $\Pi_m(\mathbf{B})$ equals

[5] For other constructions that lead to ILU-decompositions, see, for instance, §B.3.

$\Pi(\mathbf{B})$ except at the diagonal entries, the ith diagonal entry of $\Pi_m(\mathbf{B})$ equals $B_{ii}+\sum_j B_{ij}$, where we sum over all j for which $\Pi(\mathbf{B})_{ij}=0$ (and $B_{ij}\neq 0$). Note that

$$\Pi_m(\mathbf{B})\mathbf{1}=\mathbf{B}\mathbf{1}.$$

Here $\mathbf{1}$ is the n-vector with all coordinates equal to 1.

This last property implies that $\mathbf{LU1}=\mathbf{A1}$ for MILU-decompositions (modified ILU-decompositions; see, Exercise 10.7).

The vector $\mathbf{1}$ forms the ultimate smooth vector. The idea of MILU is that for 'smooth' vectors the action of the preconditioner is close to the action the matrix, that is, if the solution $\mathbf{u}$ of $\mathbf{Au}=\mathbf{r}$ is 'smooth', then the solution $\mathbf{u}'$ of $\mathbf{LUu}'=\mathbf{r}$ is close to $\mathbf{u}$: with respect to smooth solutions MILU forms good preconditioners, often much better than its ILU variant.

Exercise 10.7.

(a) Prove that $\mathbf{L}=[(\mathbf{I}-\ell_{n-1}\mathbf{e}_{n-1}^*)\ldots(\mathbf{I}-\ell_1 e_1^*)]^{-1}=(\mathbf{I}+\ell_1 e_1^*)\ldots(\mathbf{I}+\ell_{n-1}\mathbf{e}_{n-1}^*)$.

(b) Prove that $\mathbf{LU1}=\mathbf{A1}$ if the $\mathbf{L}$ and $\mathbf{U}$ factors are constructed with a modified dropping strategy.

RILU(ω)-decompositions. In practise lumping the dropped elements to the diagonal (as in MILU) may lead to diagonal elements that are very small in absolute value, whence to absolute small pivots. This would make the U factors unstable. To avoid this instability, but still profit from a better preconditioner, **relaxed** variants have been introduced.

Select an $\omega\in[0,1]$. Then $\Pi_\omega(\mathbf{B})$ equals $\Pi(\mathbf{B})$ except at the diagonal entries: the ith diagonal entry of $\Pi_\omega(\mathbf{B})$ is equal to $B_{i,i}+\omega(\sum_j B_{ij})$, where we sum over all j for which $\Pi(\mathbf{B})_{ij}=0$.

Note that, if Π does not drop diagonal entries (which is usually the case), RILU equals ILU for $\omega=0$ and equals MILU for $\omega=1$. For discretised Laplacian type of operators $\omega=0.95$ is a popular choice. The optimal choice of ω is very much problem dependent.

Exercise 10.8. **RILU.**

(a) ...

B.2 ILU(k).

For subsets $\mathbb{F}$ of $\mathbb{I}$, define $\mathbb{F}^+$ by

$$\mathbb{F}^+\equiv\mathbb{F}\cup\{(i,j)\in\mathbb{I}\,\big|\,(i,k),(k,j)\in\mathbb{F}\ \text{ for some }\ k<\min(i,j)\}.$$

Recursive application of this definition, $\mathbb{E}_{j+1}\equiv\mathbb{F}_j^+$, starting with some subset $\mathbb{F}_0$ of $\mathbb{I}$, leads to the sequence $\mathbb{F}_0\subset\mathbb{F}_1\subset\mathbb{F}_2,\ldots$ of subsets of $\mathbb{I}$.

The set $\mathbb{F}_j(\mathbf{A})\equiv\mathbb{F}_j$ obtained in this way starting with $\mathbb{F}_0\equiv\mathbb{S}(\mathbf{A})\equiv\{(i,j)\,\big|\,A_{i,j}\neq 0\}$ is called the **fill pattern** of $\mathbf{A}$ of **level** j. In particular $\mathbb{S}(\mathbf{A})$ is the full pattern of level 0. If we take $\mathbb{F}=\mathbb{F}_k(\mathbf{A})$ as dropping set, that is, as fill-pattern, then we obtain **level** k ILU: **ILU**(k).

Note that the set $\mathbb{F}_k(\mathbf{A})$ can be determined without floating point operations. It is actually a computation of graphs, starting with the graph of the matrix $\mathbf{A}$.[6] If a k is fixed, then the set $\mathbb{F}_k(\mathbf{A})$ is determined (without using the actual values of the matrix entries) before the incomplete elimination process is started.

Exercise 10.9. **Fill-patterns.** Let us call non-zero entries at location $(i,j)\in\mathbb{F}_0(\mathbf{A})$ a matrix entries of level 0. If, in the (full) Gaussian elimination process, a non-zero arises by a combination of level 0 matrix entries, then we call that entry an entry of level 1, combinations of level 1 entries are level 2 entries, etc.

(a) Show that the set of locations (i,j) of the matrix entries of level k is contained in $\mathbb{F}_k(\mathbf{A})$.

[6] The graph of the matrix $\mathbf{A}$ consists set of vertices $\mathcal{V}\equiv\{1,\ldots,n\}$ and the set of edges $\mathcal{E}\equiv\{(i,j)\,\big|\,A_{i,j}\neq 0\}$: $\mathbb{F}_0(\mathbf{A})=\mathbb{S}(\mathbf{A})$ is (equivalent to) the set of edges of this graph. The next level graph, corresponding to $\mathbb{F}_1(\mathbf{A})$, arises by adding an edge between two vertices if there is a path length of 2 between these vertices.

Theorem 10.1 *Assume the incomplete decompositions are based on a fixed fill-pattern* $\mathbb{F}$ *that contains* $\mathbb{S}(\mathbf{A})$.
1) *If* $\mathbf{L}$ *and* $\mathbf{U}$ *are the obtained (incomplete) factors and* $\mathbf{M}$ *is the product* $\mathbf{M} = \mathbf{LU}$*, then the* (i,j)*-entries of* $\mathbf{A}$ *and* $\mathbf{M}$ *coincide at* $(i,j) \in \mathbb{F}, i \neq j$*, for ILU, MILU and RILU.*
For ILU, we have that $\mathrm{diag}(\mathbf{A}) = \mathrm{diag}(\mathbf{M})$*, and for MILU,* $\mathbf{M1} = \mathbf{A1}$.
2) *Let* $\mathbf{L}$ *and* $\mathbf{U}$ *be lower- and upper triangular matrices, respectively, with* $\mathrm{diag}(\mathbf{L}) = \mathbf{I}$ *and with fill-pattern contained in* $\mathbb{F}$ *such that the entries of the product matrix* $\mathbf{M} = \mathbf{LU}$ *coincide with those of* $\mathbf{A}$ *at the* $(i,j) \in \mathbb{F}, i \neq j$.
If $\mathrm{diag}(\mathbf{A}) = \mathrm{diag}(\mathbf{M})$*, then* $\mathbf{L}$ *and* $\mathbf{U}$ *are the* $\mathbf{L}$ *and* $\mathbf{U}$ *factors as obtained with ILU.*
If $\mathbf{M1} = \mathbf{A1}$*, then* $\mathbf{L}$ *and* $\mathbf{U}$ *are the* $\mathbf{L}$ *and* $\mathbf{U}$ *factors as obtained with MILU.*

Exercise 10.10. ***Proof of Theorem*** 10.1**.**
(a) Prove the theorem (Hint. It suffices to check the correctness of the theorem only for step 3') of the algorithm and only for $j = 1$. Why?).

B.3 D-ILU preconditioners. To keep the fill-pattern simple and the costs for constructing the incomplete factors low, the D-ILU variants of (10.8) have been considered.

Note that, for any non-singular diagonal $\mathbf{D}$, $\mathbf{M}$ of (10.8) is an incomplete ILU preconditioner, where the fill-pattern of the $\mathbf{L}$ and $\mathbf{U}$ factors of $\mathbf{M}$ are the same as the fill-pattern of $\mathbf{A}$.

With (the cheap choice) $\mathbf{D} = \omega \mathbf{D}_A$ for some parameter $\omega \in (0,1)$ we have the so-called SSOR preconditioner.
Other popular (slighly more expensive) choices are obtained by determining $\mathbf{D}$ such that

- $\mathrm{diag}(\mathbf{M} - \mathbf{A}) = \mathbf{0}$ (D-ILU),
- $\mathbf{M1} = \mathbf{A1}$ (D-MILU), or
- some combination (D-RILU(ω)).

As compared to symmetric Gauss-Seidel ($\mathbf{D} = \mathbf{D}_A$), SSOR requires the determination of one scalar only (namely, ω), while the three other variants require the computation of n scalars. For ILU(0), the number of scalars to be computed equals the number of non-zeros of $\mathbf{A}$. Generally, the more costly the construction of the preconditioner is, the better it reduces iteration steps. However, the action of better preconditioners is generally also more expensive, with an exception of the D-ILU preconditioners: the Eisenstat trick allows to use them for free. As a consequence, D-ILU preconditioners are often more efficient (lead to less total costs) than more powerful ILU-preconditioners (even if they are less successful in reducing the number of iteration steps).

Exercise 10.11. Suppose $\mathbf{A}$ has non-zeros at the main diagonal or at the pth if $|p|$ is in some subset E of $\{1, \ldots, n\}$: $A_{i,j} \neq 0, i \neq j$, then $|i - j| \in E$. Suppose that

$$\text{for each} \quad p_i, p_j \in E \quad \text{we have that} \quad p_i - p_j \notin E. \tag{10.12}$$

(a) Show that, in case $\mathbf{A}$ is the discretized Laplacian on a 2-d. unit square, $E = \{1, n_x, n_x n_y\}$ satisfies this condition (10.12) if $n_x, n_y > 2$. Here, n_x, n_y is the number of gridpoints in x-, y-direction, respectively.
(b) Conclude that the namings D-ILU and D-MILU are justified (Hint: use Theorem 10.1).

B.4 ILUT(p,τ). The strategies in the ILU(k) preconditioners of §B.2 are 'static': the fill-pattern that is allowed for the L and U factors is determined before the incomplete elimination process starts. A 'dynamic' strategy that drops all entries that are small (in some relative sense), may lead to non-sparse L and U factors. The ILUT(p,τ) preconditioner is dynamic: it incorporates a drop tolerance strategy. But it also preserves sparsity simply by only preserving the p most significant values per row outside the fixed sparsity set $\mathbb{S}(\mathbf{A})$. Here, p is a modest number in $\mathbb{N}$. To be more precise, for each i, it selects from $\{j \mid (i,j) \notin \mathbb{S}(\mathbf{A}), |B_{i,j}| > \tau \|\mathbf{e}_i^* \mathbf{A}\|_2\}$ the p j's with largest $|B_{ij}|$ and sets $\Pi(\mathbf{B})_{ij} = B_{ij}$ for these j and for the j with $(i,j) \in \mathbb{S}(\mathbf{A})$. For the other values of j, $\Pi(\mathbf{B})_{ij} = 0$.

B.5 Block ILU. The above forms of incomplete LU-decomposition have block variants, where the matrix entries are replaced by block matrices.

Suppose $\mathbf{A}$ is partioned into $p \times p$ blocks with diagonal blocks $\mathbf{A}_{i,i}$ of size (j_i, j_i) with other blocks $\mathbf{A}_{i,j}$ of mathing dimension: $n = j_1 + \ldots + j_p$ and

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \ldots & \mathbf{A}_{1,p} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \ldots & \mathbf{A}_{2,p} \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_{p,1} & \mathbf{A}_{p,2} & \ldots & \mathbf{A}_{p,p} \end{bmatrix}$$

The diagonal block matrix with diagonal blocks $\mathbf{A}_{i,i}$ is often used (called block Jacobi preconditioner), or an incomplete preconditioner arising by forming an incomplete LU decomposition of each of the diagonal blocks $\mathbf{A}_{i,j}$.

Such a partitioning into blocks arises in applications as *Domain Decomposition*s. The partitioning can be induces by a natural partitioning of the domain of an underlying partial differential equation that defines $\mathbf{A}$ or can be the result of an allocation of parts of the matrix to certain processors in a multiple core computer.

B.6 Multilevel ILU. More advanced preconditioners, as MILU (multilevel ILU), AMG (algebraic multigrid), etc., rely on blocks with sizes that decrease during process of forming the decomposition, use a reordering (pivoting) strategy, and use a threshold parameter τ for dropping that may be determined dynamically (where "smallness' depends on the computed matrices).

The construction procedure of a **multilevel ILU** of $\mathbf{A}$ first determines a reordering J_r and a reordening J_c (permutations of $(1, 2, 3, \ldots, n)$) of the rows and of the columns of $\mathbf{A}_0 \equiv \mathbf{A}$. Then it partitions the reordered matrix $\mathbf{A}_0(J_r, J_c)$ into an 2×2 block matrix and computes the block LU-decomposition as indicated in (10.13), where the matrix at the left is the 2×2 block matrix representation of $\mathbf{A}_0(J_r, J_c)$.

$$\begin{bmatrix} \mathbf{D} & \mathbf{E} \\ \mathbf{F} & \mathbf{B} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F}\mathbf{D}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{D} & \mathbf{E} \\ \mathbf{0} & \widetilde{\mathbf{A}}_1 \end{bmatrix} \quad \text{with} \quad \widetilde{\mathbf{A}}_1 \equiv \mathbf{B} - \mathbf{F}\mathbf{D}^{-1}\mathbf{E}. \tag{10.13}$$

$\widetilde{\mathbf{A}}_1$ is the so-called **Schur complement** (of $\mathbf{D}$ for the matrix $\mathbf{A}_0(J_r, J_c)$). Then, the multilevel procedure sparsifies the Schur complement $\widetilde{\mathbf{A}}_1$ to a matrix $\mathbf{A}_1$, by dropping (or lumping to the diagonal) matrix entries of $\widetilde{\mathbf{A}}_1$ that are small in some sense. Reordering and partioning schould be such that

(i) $\mathbf{D}$ is easily invertible, for instance, a diagonal matrix,

(ii) if $\widetilde{\mathbf{A}}_1$ is $n_1 \times n_1$, then the dimension n_1 should be considerable smaller than $n_0 \equiv n$, for instance, $n_1 \leq \frac{1}{2} n_0$,

(iii) $\widetilde{\mathbf{A}}_1$ allows sparsification.

If $\mathbf{A}_0(J_r, J_c)$ is sparse and $\mathbf{D}$ is diagonal, then the Schur complement is also sparse (if $\mathbf{E}$ and $\mathbf{F}$ contain at most p non-zeros per row as well as per column, then $\widetilde{\mathbf{A}}_1$ contains at most p^2 non-zeros per row and per column. If $p \ll n$, then p^2 is still small). But in view of the next steps in the construction, it might not be sparse enough. Note that it is the sparsification of the Schur complement that turns the LU-decomposition into an incomplete one.

Now this procedure is repeated with $\mathbf{A}_1$ instead of $\mathbf{A}_0$, leading to an $\widetilde{\mathbf{A}}_2$, and so on. The procedure is stopped when sparsification is not possible anymore, say ending with $\widetilde{\mathbf{A}}_m$. Then a complete LU-decomposition of $\widetilde{\mathbf{A}}_m$ is computed to complete the computation of the incomplete L and U factors of (a reordered version of) $\mathbf{A}$. These factors appear as a product of (lower and upper, respectively) block matrices.

The idea is that, with $n_m \times n_m$ the dimension of $\widetilde{\mathbf{A}}_m$, n_m is small enough to allow complete LU-factorisation: if $n_{j+1} \leq \frac{1}{2} n_j$ then $n_m \leq 2^{-m} n$ (the dimension of the $\widetilde{\mathbf{A}}_j$ blocks decreases exponentially with increasing j).

Note that reordering and partitioning also determines the succes of sparsification. The procedures for the construction of the most succesful multilevel ILU preconditioners let smallness in each sparsification step depend on the effect of dropping onto the next step (the procedure "**look**s **ahead**" for one step).

B.7 Sparse approximate inverses. ILU-preconditioners $\mathbf{M}$ can be viewed as approximations of $\mathbf{A}$ that allow efficient inversion (or, to be more precise, systems involving the preconditioner can efficiently be solved).[7] Another class of preconditioners arise from attempts to approximate $\mathbf{A}^{-1}$ by sparse matrices. If $\mathbf{V}$ is such a **SPAI** (**sparse approximate inverse**) preconditioner then the action of the preconditioner is simply multiplication by $\mathbf{V}$: rather than, 'solve $\mathbf{Mc} = \mathbf{r}$ for $\mathbf{c}$', we have 'compute $\mathbf{c} = \mathbf{Vr}$ by matrix-vector multiplication'.
Generally, ILU-preconditioners are more successful in reducing the number of iterations steps as well as in reducing the total number of floating point operations than SPAI-preconditioners. However, solving systems involving L and U factors are hard to parallelise (systems with L factors have to be solved from top to bottom and are intrinsically sequential), whereas multiplications with sparse factors are relatively easy to parallelise. Depending on the computer architecture and properties of the linear systems that are to be solved, SPAI-preconditioners may lead to faster (in real time) solution processes than ILU-preconditioners on multicore computers.

C Deflation

The purpose of preconditioning is to cluster the eigenvalues away form 0. The deflation technique, to be discussed below, aims to remove absolute small eigenvalues, without touching (too much) the other eigenvalues.

Let $\mathbf{A}$ be a non-singular $n \times n$ matrix.
Let $\mathbf{U}$ be an $n \times \ell$ matrix. Put $\mathbf{V} \equiv \mathbf{AU}$. Let $M \equiv \mathbf{U}^*\mathbf{AU} = \mathbf{U}^*\mathbf{V}$ be the **interaction matrix**. Put

$$\Pi_1 \equiv \mathbf{I} - \mathbf{V}M^{-1}\mathbf{U}^*, \quad \Pi_0 \equiv \mathbf{I} - \mathbf{U}M^{-1}\mathbf{U}^*\mathbf{A}, \quad \text{and} \quad \mathbf{A}' \equiv \Pi_1\mathbf{A}. \tag{10.14}$$

For an n-vector $\mathbf{b}$ let $\mathbf{x}'$ be the n-vector that solves the projection of the problem $\mathbf{Ax} = \mathbf{b}$:

$$\mathbf{A}'\mathbf{x}' = \mathbf{b}' \equiv \Pi_1\mathbf{b} \quad \text{and} \quad \mathbf{x}' \perp \text{span}\,(\mathbf{U}). \tag{10.15}$$

Then the solution $\mathbf{x}$ of $\mathbf{Ax} = \mathbf{b}$ is given by

$$\mathbf{x} = \mathbf{U}\vec{\beta} + \Pi_0\mathbf{x}' = \mathbf{x}' + \mathbf{U}M^{-1}\mathbf{U}^*(\mathbf{b} - \mathbf{Ax}') \quad \text{where} \quad \vec{\beta} \equiv M^{-1}\mathbf{U}^*\mathbf{b}. \tag{10.16}$$

Note that $\mathbf{b}' = \mathbf{b} - \mathbf{V}\vec{\beta}$.

$\mathbf{A}'$ maps $\mathbf{U}^\perp$ to $\mathbf{U}^\perp$. Therefore, Krylov subspace methods (started with a vector as $\mathbf{b}'$ [i.e., $\mathbf{x}'_0 = \mathbf{0}$] that is orthogonal to $\mathbf{U}$) can safely be applied for solving (10.15). $\mathbf{A}'$ restricted to $\mathbf{U}^\perp$ is the **deflated matrix**: span $(\mathbf{U})$ is 'deflated' from its domain and range.

If span $(\mathbf{U})$ contains a left eigenvector, then the associated eigenvalue is "replaced" by 0 in $\mathbf{A}'$ and it is 'removed' in $\mathbf{A}'$ as a map from $\mathbf{U}^\perp$ to $\mathbf{U}^\perp$. If the other eigenvalues are not much affected by the deflation, then we the deflation reduces the condition number ($C_2(\mathbf{A}) < \mathcal{C}_2(\mathbf{A}'\big|_{\mathbf{U}^\perp})$) and Krylov subspace methods may expected to converge faster

Exercise 10.12.

(a) Prove that Π_1 is a skew projection that projects onto $\mathbf{U}^\perp$ with kernel span($\mathbf{V}$).

(b) Prove that Π_0 is a skew projection that projects onto $(\mathbf{A}^*\mathbf{U})^\perp$ with kernel span($\mathbf{U}$).

(c) Show that $\mathbf{A}' = \Pi_1\mathbf{A} = \Pi_0\mathbf{A}$.

(d) Prove that $\mathbf{b}' \perp \mathbf{U}$, and $\mathbf{A}'$ maps $\mathbf{U}^\perp$ to $\mathbf{U}^\perp$. In particular, $\mathcal{K}_k(\mathbf{A}', \mathbf{b}') \subset \mathbf{U}^\perp$.

[7] Or, if one wishes, as approximations of a product of the L- and U-factors of the complete LU-decomposition of $\mathbf{A}$

(e) Prove (10.16).

(f) Assume $\mathbf{A}$ is Hermitian. Prove that $\Pi_1^* = \Pi_0$ and that $\mathbf{A}'$ is Hermitian.

Exercise 10.13. **Deflation.** Suppose $\mathbf{U}$ spans a space spanned by left eigenvectors.

(a) Show that $\mathbf{A}$ maps $\mathbf{U}^\perp$ to $\mathbf{U}^\perp$.

(b) Show that, for $\mathbf{b}' \perp \mathbf{U}$, the problem "Solve $\mathbf{A}\mathbf{x}' = \mathbf{b}'$ for $\mathbf{x}' \perp \mathbf{U}$" has a unique solution. Show that this solution also solves (10.19).

Suppose we have methods that solve any $\mathbf{A}\mathbf{u} = \mathbf{c}$ approximately for $\mathbf{u}$:

$$\mathbf{u} = \mathcal{M}_i(\mathbf{A}, \mathbf{c}),$$

i.e., the ith method produces and approximate solution $\mathbf{u}$ for the problem $\mathbf{A}\mathbf{u} = \mathbf{c}$. Then, these methods can be combined to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$ by subsequently applying the methods to the problem shifted by the previous residual: with $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{r}_0 = \mathbf{b}$,

$$\mathbf{u}_i = \mathcal{M}_i(\mathbf{A}, \mathbf{r}_{i-1}), \quad \mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{u}_i, \quad \mathbf{r}_i = \mathbf{r}_{i-1} - \mathbf{A}\mathbf{u}_i \quad (i = 1, 2, \ldots, k).$$

Upon termination, $\mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{r}_k$.

The idea is that each methods produces efficiently good approximations for certain components of the solution vector $\mathbf{x}$ and are inefficient with respect to other components, components for which another method (or combination of) is efficient.

The deflation technique in (10.14)-(10.16) can be viewed as being of this type as we will learn in Exercise 10.14.

$$\begin{aligned} &\mathcal{M}_1\text{: Solve for } \mathbf{u} \in \text{span}(\mathbf{U}) \text{ such that } \mathbf{c} - \mathbf{A}\mathbf{u} \perp \mathbf{U}. \\ &\mathcal{M}_2\text{: Solve for } \mathbf{u} \perp \mathbf{U} \text{ such that } \mathbf{c} - \mathbf{A}\mathbf{u} \in \text{span}(\mathbf{V}). \\ &\mathcal{M}_3 = \mathcal{M}_1. \end{aligned} \tag{10.17}$$

Then, (10.15) is a representation of $\mathcal{M}_2$ with $\mathbf{c} \perp \mathbf{U}$ and with $\mathbf{x}'$ as in (10.15), we have (cf., (10.16))

$$\mathbf{x} = \mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3 \quad \text{with} \quad \mathbf{u}_1 = \mathbf{U}M^{-1}\mathbf{U}^*\mathbf{b},\ \mathbf{u}_2 = \mathbf{x}',\ \mathbf{u}_3 = \mathbf{U}M^{-1}\mathbf{U}^*(\mathbf{b}' - \mathbf{A}\mathbf{x}').$$

Note that, replacing the third method by

$$\mathcal{M}_3\text{: Solve for } \mathbf{u} \in \text{span}\,(\mathbf{U}) \text{ such that } \mathbf{c} - \mathbf{A}\mathbf{u} \perp \mathbf{V} \tag{10.18}$$

also leads to the exact solution if all solutions are exact.

Exercise 10.14.

(a) With $\mathbf{u}_1 \equiv \mathbf{U}M^{-1}\mathbf{U}^*\mathbf{b}$, we have $\mathbf{u}_1 \in \text{span}\,(\mathbf{U})$ and $\mathbf{b}' \equiv \Pi_1\mathbf{b} = \mathbf{b} - \mathbf{A}\mathbf{u}_1 \perp \mathbf{U}$.

(b) Prove that (10.15) represents

$$\text{with} \quad \mathbf{b}' \perp \mathbf{U}, \quad \text{solve} \quad \mathbf{b}' - \mathbf{A}\mathbf{x}' \in \text{span}\,(\mathbf{V}) \quad \text{for} \quad \mathbf{x}' \perp \mathbf{U}. \tag{10.19}$$

(c) $\mathbf{b}'' \equiv \mathbf{b}' - \mathbf{A}\mathbf{x}' \in \text{span}\,(\mathbf{V}) \quad \Leftrightarrow \quad \Pi_1(\mathbf{b}'') = \Pi_1(\mathbf{b}' - \mathbf{A}\mathbf{x}') = \mathbf{b}' - \mathbf{A}'\mathbf{x}' = \mathbf{0}$

(d) With $\mathbf{u}_3 \equiv \mathbf{U}M^{-1}\mathbf{U}^*\mathbf{b}''$, we have $\mathbf{u}_3 \in \text{span}\,(\mathbf{U})$ and $\Pi_1(\mathbf{b}'') = \mathbf{b}'' - \mathbf{A}\mathbf{u}_3 \perp \mathbf{U}$.

(e) Since $\Pi_1(\mathbf{b}'') = \mathbf{0}$, we have that $\mathbf{x} = \mathbf{u}_1 + \mathbf{x}' + \mathbf{u}_3$.

In (10.14)–(10.16), we assumed that $\mathcal{M}_1$ and $\mathcal{M}_2$ solve the problem exactly. There are variants possible (and often used) where the methods (or one of the methods) produce an approximate solution. Then, the methods can be applied in a cyclic fashion: $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_1, \mathcal{M}_1, \mathcal{M}_2, \ldots$. In **GCRO** (GCR orthogonalised), (10.15) is solved with (i.e., $\mathcal{M}_2$ is) ℓ steps of GMRES for

some fixed ℓ and for the next cycle of $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_1$, the matrix $\mathbf{U}$ is expanded with the GMRES solution.[8]

If the projected problem (cf., (10.20) below) that is to be solved by $\mathcal{M}_1$ is still high dimensional, the approach can be nested (as is the standard strategy in multigrid).

Note that $\mathcal{M}_1$ for $\mathbf{A}\mathbf{u}_i = \mathbf{r}_{i-1}$ reads as, find $y \in \mathbb{C}^s$ such that

$$\mathbf{U}^*\mathbf{A}\mathbf{U}y = \mathbf{U}^*\mathbf{r}_{i-1} \quad \text{and} \quad \mathbf{u}_i = \mathbf{U}y. \tag{10.20}$$

In some applications (in multigrid [MG] and domain decomposition [DD]), the map $\mathbf{U}$ is viewed as a **restriction operator** (restricting to a courser grid) and $\mathbf{U}^*$ as a **prolongation** (to the fine grid). Then (10.20) is the course grid problem with course grid operator $M \equiv \mathbf{U}^*\mathbf{A}\mathbf{U}$. When solved for $\mathbf{u}_i$ followed by $\mathbf{x}_i = \mathbf{x}_{i-1} + \mathbf{u}_i$ a course grid correction has been applied. In standard multigrid, the second method is an application of a fixed finite number of steps with a classical iteration method for solving $\mathbf{A}\mathbf{u}_i = \mathbf{r}_{i-1}$. In particular, no deflation is applied to $\mathbf{A}$ (there is only deflation in the right-hand side vector $\mathbf{r}_i$). In domain decomposition, $\mathbf{V}$ and $\mathbf{U}$ are both replaced in (10.19) by some (the same) $\widetilde{\mathbf{U}}$. MG nor DD find an exact solution of (10.19) and require recycling of the methods. However convergence is very fast.

Exercise 10.15. **Restarted GMRES and FOM.** For some $n \times n$ matrix $\mathbf{A}$ and an n-vector $\mathbf{b}$, consider the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$.
Let $\mathbf{U}$ span the Krylov subspace $\mathcal{K}_\ell(\mathbf{A}, \mathbf{b})$. Consider the situation in (10.14)–(10.16).

(a) Prove that

$$\mathcal{K}_k(\mathbf{A}', \mathbf{b}') \oplus \operatorname{span}(\mathbf{U}) = \mathcal{K}_{\ell+k}(\mathbf{A}, \mathbf{b}) \quad \text{and} \quad \mathbf{A}'\,\mathcal{K}_k(\mathbf{A}', \mathbf{b}') \oplus \operatorname{span}(\mathbf{V}) = \mathbf{A}\mathcal{K}_{\ell+k}(\mathbf{A}, \mathbf{b}).$$

(b) Suppose we solve the problem (10.15) with k steps of FOM (i.e., inexact $\mathcal{M}_2$ of (10.17)). Then, we (exactly) correct the resulting residual shifted problem with $\mathcal{M}_3 = \mathcal{M}_1$ of (10.17) and initial guess $\mathbf{0}$. Prove that in this way we obtain the same result (the same residual) as with $\ell + k$ steps of FOM for $\mathbf{A}\mathbf{x} = \mathbf{b}$ and initial guess $\mathbf{0}$.

(c) Suppose we solve the problem (10.15) with k steps of GMRES (i.e., inexact $\mathcal{M}_2$ of (10.17)). Then, we (exactly) correct the resulting residual shifted problem with $\mathcal{M}_3$ of (10.18) and initial guess $\mathbf{0}$. Prove that in this way we obtain the same result (the same residual) as with $\ell + k$ steps of GMRES for the original problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ and initial guess $\mathbf{0}$.

(d) Let $\mathbf{U}'$ be an $n \times \ell'$ matrix with $\ell' \leq \ell$ and $\operatorname{span}(\mathbf{U}') \subset \operatorname{span}(\mathbf{U})$. Instead of (10.15), or, equivalently, the second problem in (10.17), we

$$\text{solve for } \mathbf{u} \perp \mathbf{U}' \text{ such that } \mathbf{c} - \mathbf{A}\mathbf{u} \in \operatorname{span}(\mathbf{V}'), \text{ where } \mathbf{V}' \equiv \mathbf{A}\mathbf{U}'. \tag{10.21}$$

Show that k steps of FOM or GMRES for an inexact solve of (10.21) with the same preprocess and postprocess as above, lead to the same result (the same residual as with $\ell + k$ steps of these methods for the original problem).

(e) Conclude that after ℓ steps the FOM and GMRES process for solving the original problem with initial guess $\mathbf{x}_0 = \mathbf{0}$ behaves as these processes with a matrix from which Ritz vectors in $\mathcal{K}_\ell(\mathbf{A}, \mathbf{b})$ have been deflated.

Exercise 10.16. Let $\mathbf{U}$ be orthonormal. Put $\mathbf{R} \equiv \mathbf{U}M - \mathbf{A}\mathbf{U}$ and $\mathbf{S}^* \equiv M\mathbf{U}^* - \mathbf{U}^*\mathbf{A}$.

(a) Show that $\Pi_1\mathbf{A} = \mathbf{A} - \mathbf{U}M^{-1}\mathbf{U}^* + \mathbf{U}\mathbf{S}^* + \mathbf{R}\mathbf{U}^* - \mathbf{R}M^{-1}\mathbf{S}^*$ and

$$\Pi_1\mathbf{A} = (\mathbf{I} - \mathbf{U}\mathbf{U}^*)(\mathbf{A} - \mathbf{R}M^{-1}\mathbf{S}^*)(\mathbf{I} - \mathbf{U}\mathbf{U}^*).$$

The projected matrix $\Pi_1\mathbf{A}$ equals a 'perturbed' matrix $\mathbf{A} - \mathbf{R}M^{-1}\mathbf{S}^*$ from which that space $\operatorname{span}(\mathbf{U})$ has been deflated.

[8] This method was first introduced as a GCR method where the selection of the expansion vector $\mathbf{u}_k$ is the approximate solution of (10.15), rather than $\mathbf{r}_k$ as in standard GCR. Due to the projection Π_1 the vector $\mathbf{c}_k \equiv \Pi_1\mathbf{A}\mathbf{u}_k$ is already orthogonal to $\mathbf{c}_0, \ldots, \mathbf{c}_{k-1}$ and the orthogonalization loop of GCR can be skipped.

(b) If $\mathbf{A}$ is Hermitian and $\mathbf{U} = \mathbf{u}$ is an approximate eigenvector, then $\mathbf{r} \equiv \mathbf{R}$ is the residual $\mathbf{r} = \mathbf{u}\vartheta - \mathbf{A}\mathbf{u}$ with $\vartheta \equiv \mathbf{u}^*\mathbf{A}\mathbf{u}$ and $\mathbf{s} = \mathbf{r}$. Then, $\mathbf{R}M^{-1}\mathbf{S} = \frac{1}{\vartheta}\,\mathbf{r}\mathbf{r}^*$ can be viewed as a perturbation of $\mathbf{A}$ of size $\|\mathbf{r}\|_2^2/|\vartheta|$ and $\Pi_1(\mathbf{A})$ is the perturbed matrix from which the approximate eigenvector $\mathbf{u}$ has been deflated.

CG as a deflated process. In methods as Lanczos (and in CG) for Hermitian matrices $\mathbf{A}$, the expansion vector $\mathbf{v}_{k+1}$ is obtained by orthonormalizing $\mathbf{A}\mathbf{v}_k$ against $\mathbf{v}_k$ and $\mathbf{v}_{k-1}$. Then, due to symmetry of $\mathbf{A}$, $\mathbf{v}_{k+1}$ is automatically orthogonal to $\mathbf{v}_j$ for all $j < k$. In some sense, the multiplication by $\mathbf{A}$ can be viewed as being a multiplication by a deflated matrix. The following result (formulated for CG) makes this point of view explicit.

Consider the (unpreconditioned) CG process for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ starting with $\mathbf{x}_0 = \mathbf{0}$. Here, $\mathbf{A}$ is positive definite. Then $\mathbf{r}_0 = \mathbf{b}$ and, with $\mathbf{u}_0 \equiv \mathbf{0}$, the kth step of CG reads as

$$\begin{aligned}
&\rho_k = \mathbf{r}_k^*\mathbf{r}_k, \quad \beta_k = \tfrac{\rho_k}{\rho_{k-1}},\\
&\mathbf{u}_{k+1} = \mathbf{r}_k - \beta_k\mathbf{u}_k, \quad \mathbf{c}_{k+1} = \mathbf{A}\mathbf{u}_{k+1},\\
&\sigma_{k+1} = \mathbf{u}_{k+1}^*\mathbf{c}_{k+1}, \quad \alpha_k = \tfrac{\rho_k}{\sigma_{k+1}},\\
&\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{u}_{k+1}, \quad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{c}_{k+1}.
\end{aligned} \tag{10.22}$$

(Note: here we took the scheme from Exercise 7.2, but to ease notation below, that is, to allow working with index k rather than $k+1$, we increased in the above scheme the index of the $\mathbf{u}_k$ and $\mathbf{c}_k$ by 1).

Proposition 10.2 *Select a positive integer k. Consider the matrix $\widetilde{\mathbf{A}}$ defined by*

$$\widetilde{\mathbf{A}} \equiv \mathbf{A} - \frac{1}{\sigma_k}\mathbf{c}_k\,\mathbf{c}_k^* = \mathbf{A}\,\left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{u}_k\,\mathbf{c}_k^*\right) = \left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{c}_k\,\mathbf{u}_k^*\right)\,\mathbf{A}. \tag{10.23}$$

Then, $\widetilde{\mathbf{A}}$ is Hermitian and, with $\mathcal{K} \equiv \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0)$ and $\widetilde{\mathbf{r}}_0 \equiv \mathbf{r}_k$, we have that

$$\widetilde{\mathbf{A}} : \mathcal{K} \to \mathcal{K}, \qquad \widetilde{\mathbf{A}} : \mathcal{K}^\perp \to \mathcal{K}^\perp, \quad \textit{and} \quad \widetilde{\mathbf{r}}_0 \perp \mathcal{K}.$$

Moreover, the equation $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{r}}_0$ has exactly one solution $\widetilde{\mathbf{x}} \perp \mathcal{K}$. Solve this equation for $\widetilde{\mathbf{x}} \perp \mathcal{K}$ with CG starting with $\widetilde{\mathbf{x}}_0 = \mathbf{0}$. Then, with CG approximations $\widetilde{\mathbf{x}}_j$ and residuals $\widetilde{\mathbf{r}}_j$, we have

$$\mathbf{r}_{k+j} = \widetilde{\mathbf{r}}_j, \quad \|\mathbf{r}_{k+j}\|_{A^{-1}} = \|\widetilde{\mathbf{r}}_j\|_{\widetilde{A}^{-1}}, \tag{10.24}$$

$$\mathbf{x} = \mathbf{x}_k + \left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{u}_k\,\mathbf{c}_k^*\right)\widetilde{\mathbf{x}}, \quad \textit{and} \quad \mathbf{x}_{k+j} = \mathbf{x}_k + \left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{u}_k\,\mathbf{c}_k^*\right)\widetilde{\mathbf{x}}_j. \tag{10.25}$$

In Lecture 7 (see the transparancies), we explained superlinear convergence of CG by bounding the CG process by a CG process for a matrix from which converged eigenvectors have been removed (using the fact that zeros of the CG polynomial are Ritz values, cf., Theorem 6.1). The explanation assumes that a Ritz value is very close to an (extremal) eigenvalue and relies on a *bounding* CG process. The above theorem allows to view the CG process from step k on as actually *being* a CG process with a deflated matrix. In Lecture 12, we will see theorems that allow us the estimate the eigenvalues of $\widetilde{\mathbf{A}}$ and to explain superlinear convergence also in cases where Ritz values are not very close to eigenvalues.

Exercise 10.17. *Proof of Proposition* 10.2. Recall that the vectors produced by CG for solving $\mathbf{A}\mathbf{x} = \mathbf{r}_0$ have the following properties

$$\mathbf{x}_k \in \mathcal{K} \equiv \mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = \mathrm{span}(\mathbf{r}_0, \ldots, \mathbf{r}_{k-1}) = \mathrm{span}(\mathbf{u}_1, \ldots, \mathbf{u}_k)$$

and

$$\mathbf{r}_k \perp \mathbf{r}_j, \quad \mathbf{r}_k \perp \mathbf{u}_{j+1}, \quad \mathbf{c}_k \perp \mathbf{u}_{j+1} \quad (j < k).$$

(a) Prove that $\widetilde{\mathbf{A}}\mathbf{u}_k = \mathbf{0}$. Hence, $\widetilde{\mathbf{A}}\mathbf{y} \perp \mathbf{u}_k$ for all $\mathbf{y} \in \mathbb{C}^n$.
Prove that, $\mathbf{r}_k^*\widetilde{\mathbf{A}}\mathbf{r}_{k-1} = \mathbf{r}_{k-1}^*\widetilde{\mathbf{A}}\mathbf{r}_k = 0$, $\widetilde{\mathbf{A}}\mathbf{y} \in \mathcal{K}$ for all $\mathbf{y} \in \mathcal{K}$, and $\widetilde{\mathbf{A}}\mathbf{y} \perp \mathcal{K}$ for all $\mathbf{y} \perp \mathcal{K}$.
(b) For $\widetilde{\mathbf{x}} \perp \mathcal{K}$, put

$$\mathbf{x} \equiv \mathbf{x}_k + \left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{u}_k\,\mathbf{c}_k^*\right)\widetilde{\mathbf{x}}.$$

Prove that $\mathbf{A}\mathbf{x} = \mathbf{r}_0$ iff $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{r}}_0$. Conclude that there is exactly one $\widetilde{\mathbf{x}} \perp \mathcal{K}$ that solves $\widetilde{\mathbf{A}}\widetilde{\mathbf{x}} = \widetilde{\mathbf{r}}_0$.
(c) Prove that $\mathcal{K}_{k+j}(\mathbf{A}, \mathbf{r}_0) = \mathcal{K} \oplus \mathcal{K}_j(\widetilde{\mathbf{A}}, \widetilde{\mathbf{r}}_0)$ for all j. Conclude that $\mathbf{r}_{k+j}$ and $\widetilde{\mathbf{r}}_j$ are co-linear.
Put

$$\mathbf{y} \equiv \mathbf{x}_k + \left(\mathbf{I} - \frac{1}{\sigma_k}\mathbf{u}_k\,\mathbf{c}_k^*\right)\widetilde{\mathbf{x}}_j$$

Show that $\mathbf{r}_0 - \mathbf{A}\mathbf{y} = \widetilde{\mathbf{r}}_j$. Conclude that $\mathbf{y} = \mathbf{x}_{k+j}$ and $\mathbf{r}_{k+j} = \widetilde{\mathbf{r}}_j$.
(d) Prove that

$$\mathbf{r}_{k+j}^*\mathbf{A}^{-1}\mathbf{r}_{k+j} = \mathbf{r}_{k+j}^*(\mathbf{x} - \mathbf{x}_{k+j}) = \widetilde{\mathbf{r}}_j^*(\widetilde{\mathbf{x}} - \widetilde{\mathbf{x}}_j) = \widetilde{\mathbf{r}}_j^*\widetilde{\mathbf{A}}^{-1}\widetilde{\mathbf{r}}_j.$$

Here $\widetilde{\mathbf{A}}$ is inverted on the space $\mathcal{K}^\perp$.

D Parallelisation

Exercise 10.18. On parallel computers we want to have independent computations. Why is this requirement inherently difficult to combine with a good preconditioner?

Exercise 10.19.

(a) Propose an efficient algorithm to add together the results of the partial inner products on a distributed memory computer. Your algorithm should require as little communication as possible (for subsequent computations, the value of the inner product should be available on all processors involved).

(b) Which orthogonalization scheme (modified, classical Gram-Schmidt, or ...) should be used within GMRES on a distributed memory parallel computer? Explain your answer.