Numerical Linear Algebra for Computational Science and Information Engineering

Lecture 14
Preconditioned Iterative Methods for Linear Systems

Nicolas Venkovic
nicolas.venkovic@tum.de

Group of Computational Mathematics
School of Computation, Information and Technology
Technical University of Munich

Summer 2025

Outline I

1 Preconditioned iterations 1

- Introduction 1
- Preconditioned conjugate gradient (PCG) method 4
- Preconditioned GMRES 11
- Other methods 19
- Flexible variants 20

2 Preconditioners 27

- Introduction 27
- Stationary iterative methods 28
- Incomplete factorizations 31
- Sparse approximate inverses (SPAI) 47
- Other preconditioners 54

3 Homework problems 55

4 Practice session 56

Outline II

5 References 57

Preconditioned iterations

Introduction

- The rate of **convergence of Krylov subspace methods** applied to solving $Ax = b$ is strongly **influenced by** the **eigenvalues distribution** of the matrix A and, in the case of non-symmetric matrices, by the **eigenvectors** as well.

  If the matrix A is ill-conditioned, or has unfavorable spectral properties, then, **Krylov subspace methods** applied to solving $Ax = b$ are likely to **converge very slowly**.

- **Preconditioning**, which consists of *transforming an intractable problem into a more efficiently solvable form*, is an essential component **to improve the efficiency and robustness of Krylov subspace iteration methods**.

  For the iterative solve of $Ax = b$ with $A \in \mathbb{R}^{n \times n}$, a **preconditioner**, which we denote by M, is, in its general form, a *map*

$$x \in \mathbb{R}^n \mapsto M^{-1}x \in \mathbb{R}^n$$

  which can be **efficiently evaluated**, and such that M is a **good approximation of** A, in some sense. The map must not be linear.

Introduction, cont'd$_1$

- In agreement with this loose definition of preconditioning, a preconditioner can consist of
  - A **factorization** of M which allows for efficient solve of $Mz = x$,
  - A (sparse) **matrix** M^{-1} which allows for efficient matrix-vector product $x \mapsto M^{-1}x$ evaluation,
  - An efficient **matrix-free linear map** $x \mapsto M^{-1}x$,
  - An **iterative linear solver** applied to $Mz = x$.
- We denote three different ways to apply a preconditioner M to a linear system $Ax = b$:
  - **Left-preconditioning**, which applies the preconditioner from the left:
    $$\boxed{M^{-1}Ax = M^{-1}b}.$$
  - **Right-preconditioning**, which applies the preconditioner from the right:
    $$\boxed{AM^{-1}u = b \;\text{ with }\; x = M^{-1}u}.$$
  - **Split-preconditioning**, which, for a given factorization $M = M_L M_R$, applies the preconditioner from both side:
    $$\boxed{M_L^{-1}AM_R^{-1}u = M_L^{-1}b \;\text{ with }\; x = M_R^{-1}u}.$$

Introduction, cont'd$_2$

- Here, we assume that both A and M are non-singular.

  The *left-preconditioned* $M^{-1}A$, *right-preconditioned* AM^{-1} and *split-preconditioned* $M_L^{-1}AM_R^{-1}$ coefficient matrices are **similar**. Thus, they share the **same eigenvalue distribution** although they do have **distinct eigenvector sets**. As a result,

  - If both A and M are **SPD**, then the **convergence behavior of CG** applied to a preconditioned system **is the same irrespective of the type of preconditioning**, i.e., left, right, or split.
  - For **non-normal** matrices A and M, the **Krylov subspace solver** may **behave very differently from one type of preconditioning to another**, as the convergence behavior depends on the eigenvector set.

  Indeed, the behavior of a Krylov subspace linear iterative solver depends on both the eigenvalue distribution, and the conditioning of eigenvectors of the preconditioned coefficient matrix.

  A valid strategy for the design of a preconditioner is to make the preconditioned coefficient matrix as close to normal as possible, in the limit of which, eigenvectors are ideally conditioned.

Preconditioned conjugate gradient (PCG) method

- The **symmetry** of the coefficient matrix, i.e., $A^T = A$, an essential feature in the definition and derivation of the CG method, is **not preserved**, neither **through left-preconditioning**, **nor through right-preconditioning**, i.e.,

  $$(M^{-1}A)^T \neq M^{-1}A \text{ and } (AM^{-1})^T \neq AM^{-1}$$

  even if the preconditioner, M, is symmetric.
- A priori, preserving the symmetry of the coefficient matrix is essential for the defintion and derivation of a **preconditioned conjugate gradient** (**PCG**) method.

  However, since our definition and derivation of the CG algorithm in lecture 13 relies on an *abstract definition of the inner product*, it is sufficient to find an **inner product with respect to which the preconditioned coefficient matrix is self-adjoint**.

Preconditioned conjugate gradient (PCG) method, cont'd$_1$

- The M-inner product is such that:

$$(x, y)_M := (Mx, y) = (x, M^T y) \;\forall\; x, y \in \mathbb{R}^n$$

where $(\cdot, \cdot)$ is the dot product.

Then, the **symmetry of M and A implies that $M^{-1}A$ is self-adjoint for the M-inner product**. That is:

$$\begin{aligned}
(M^{-1}Ax, y)_M &= (MM^{-1}Ax, y) \\
&= (Ax, y) \\
&= (x, A^T y) \\
&= (x, Ay) \\
&= (x, MM^{-1}Ay) \\
&= (M^T x, M^{-1}Ay) \\
&= (Mx, M^{-1}Ay)
\end{aligned}$$

so that $(M^{-1}Ax, y)_M = (x, M^{-1}Ay)_M$.

Preconditioned conjugate gradient (PCG) method, cont'd$_2$

- We can then rewrite the CG algorithm for the left-preconditioned system $M^{-1}Ax = M^{-1}b$ using an M-inner product:

Algorithm 1 CG: $(x_0, \varepsilon) \mapsto x_j$

```
r_0 := b - A x_0                                  $r_0 := b - Ax_0$
```

```
$r_0 := b - Ax_0$
$p_1 := r_0$
for $j = 1, 2 \dots$ do
    $\alpha_j := (r_{j-1}, r_{j-1})/(Ap_j, p_j)$
    $x_j := x_{j-1} + \alpha_j p_j$
    $r_j := r_{j-1} - \alpha_j Ap_j$
    if $\|r_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (r_j, r_j)/(r_{j-1}, r_{j-1})$
    $p_{j+1} := r_j + \beta_j p_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_2$

- We can then rewrite the CG algorithm for the left-preconditioned system $M^{-1}Ax = M^{-1}b$ using an M-inner product:

Algorithm 2 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
z_0 := M^{-1}b - M^{-1}Ax_0                                ▷ $r_j \mapsto z_j$
$p_1 := z_0$
**for** $j = 1, 2 \dots$ **do**
    $\alpha_j := (z_{j-1}, z_{j-1})_M / (M^{-1}Ap_j, p_j)_M$
    $x_j := x_{j-1} + \alpha_j p_j$
    $z_j := z_{j-1} - \alpha_j M^{-1}Ap_j$
    **if** $\|M^{-1}z_j\|_2 < \varepsilon\|b\|_2$ **then** Stop     ▷ Keep criterion as $\|b - Ax_j\| < \varepsilon\|b\|_2$
    $\beta_j := (z_j, z_j)_M / (z_{j-1}, z_{j-1})_M$
    $p_{j+1} := z_j + \beta_j p_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_2$

- We can then rewrite the CG algorithm for the left-preconditioned system $M^{-1}Ax = M^{-1}b$ using an M-inner product:

Algorithm 2 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$                                ▷ Introduce $r_j := Mz_j$
$z_0 := M^{-1}r_0$
$p_1 := z_0$
for $j = 1, 2 \dots$ do
    $\alpha_j := (z_{j-1}, z_{j-1})_M / (M^{-1}Ap_j, p_j)_M$
    $x_j := x_{j-1} + \alpha_j p_j$              ▷ $(z_{j-1}, z_{j-1})_M = (MM^{-1}r_{j-1}, z_{j-1}) = (r_{j-1}, z_{j-1})$
    $r_j := r_{j-1} - \alpha_j Ap_j$             ▷ $(M^{-1}Ap_j, p_j)_M = (MM^{-1}Ap_j, p_j) = (Ap_j, p_j)$
    $z_j := M^{-1}r_j$
    if $\|r_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (z_j, z_j)_M / (z_{j-1}, z_{j-1})_M$
    $p_{j+1} := z_j + \beta_j p_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_2$

- We can then rewrite the CG algorithm for the left-preconditioned system $M^{-1}Ax = M^{-1}b$ using an M-inner product:

Algorithm 2 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
r_0 := b - Ax_0
$z_0 := M^{-1}r_0$
$p_1 := z_0$
**for** $j = 1, 2 \dots$ **do**
    $\alpha_j := (r_{j-1}, z_{j-1})/(Ap_j, p_j)$
    $x_j := x_{j-1} + \alpha_j p_j$
    $r_j := r_{j-1} - \alpha_j Ap_j$
    $z_j := M^{-1}r_j$
    **if** $\|r_j\|_2 < \varepsilon\|b\|_2$ **then** Stop
    $\beta_j := (r_j, z_j)/(r_{j-1}, z_{j-1})$
    $p_{j+1} := z_j + \beta_j p_j$
```

The resulting PCG algorithm requires **one preconditioner application** $r_j \mapsto M^{-1}r_j =: z_j$ **per iteration, to the non-preconditioned residual**.

Preconditioned conjugate gradient (PCG) method, cont'd$_3$

- An alternative derivation of the PCG algorithm is possible through **split-preconditioning**.
- Since M is SPD, it admits a Cholesky decomposition $M = LL^T$ which is used as follows to split-precondition the linear system $Ax = b$:

  $$L^{-1}AL^{-T}u = L^{-1}b \text{ with } x = L^{-T}u$$

  where the preconditioned coefficient matrix $L^{-1}AL^{-T}$ is SPD.

  Consequently, the CG algorithm can be applied to the split-preconditioned linear system in order to generate a sequence of iterates $u_1, u_2, \dots$, with an initial guess u_0, to approximate u.

  From such a sequence, some associated iterates $x_0, x_1, x_2, \dots$ can be constructed as $x_j := L^{-T}u_j$ for $j = 0, 1, 2, \dots$ to approximate the solution x of the original system.
- In what follows, we show that such a sequence of iterates $x_1, x_2, \dots$ can be formed, for any initial guess x_0, without requiring actual knowledge of the Cholesky factor L.

Preconditioned conjugate gradient (PCG) method, cont'd$_4$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  We obtain the following algorithm:

Algorithm 3 CG: $(u_0, \varepsilon) \mapsto u_j$

```
$\tilde{r}_0 := L^{-1}b - L^{-1}AL^{-T}u_0$
$\tilde{p}_1 := \tilde{r}_0$
**for** $j = 1, 2 \ldots$ **do**
    $\alpha_j := (\tilde{r}_{j-1}, \tilde{r}_{j-1})/(L^{-1}AL^{-T}\tilde{p}_j, \tilde{p}_j)$
    $u_j := u_{j-1} + \alpha_j \tilde{p}_j$
    $\tilde{r}_j := \tilde{r}_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    **if** Convergence is achieved **then** Stop
    $\beta_j := (\tilde{r}_j, \tilde{r}_j)/(\tilde{r}_{j-1}, \tilde{r}_{j-1})$
    $\tilde{p}_{j+1} := \tilde{r}_j + \beta_j \tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_4$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Instead of the iterate u_j, we compute $x_j := L^{-T}u_j$:

Algorithm 3 CG: $(u_0, \varepsilon) \mapsto u_j$

```
$\tilde{r}_0 := L^{-1}b - L^{-1}AL^{-T}u_0$
$\tilde{p}_1 := \tilde{r}_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (\tilde{r}_{j-1}, \tilde{r}_{j-1})/(L^{-1}AL^{-T}\tilde{p}_j, \tilde{p}_j)$
    $L^{-T}u_j := L^{-T}u_{j-1} + \alpha_j L^{-T}\tilde{p}_j$
    $\tilde{r}_j := \tilde{r}_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    if Convergence is achieved then Stop
    $\beta_j := (\tilde{r}_j, \tilde{r}_j)/(\tilde{r}_{j-1}, \tilde{r}_{j-1})$
    $\tilde{p}_{j+1} := \tilde{r}_j + \beta_j\tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Instead of the iterate u_j, we compute $x_j := L^{-T}u_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$\tilde{r}_0 := L^{-1}b - L^{-1}Ax_0$
$\tilde{p}_1 := \tilde{r}_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (\tilde{r}_{j-1}, \tilde{r}_{j-1})/(L^{-1}AL^{-T}\tilde{p}_j, \tilde{p}_j)$
    $x_j := x_{j-1} + \alpha_j L^{-T}\tilde{p}_j$
    $\tilde{r}_j := \tilde{r}_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    if $\|b - Ax_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (\tilde{r}_j, \tilde{r}_j)/(\tilde{r}_{j-1}, \tilde{r}_{j-1})$
    $\tilde{p}_{j+1} := \tilde{r}_j + \beta_j\tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  We introduce the non-preconditioned residual $r_j := L\tilde{r}_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$L\tilde{r}_0 := b - Ax_0$
$\tilde{p}_1 := L^{-1}L\tilde{r}_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (L\tilde{r}_{j-1}, M^{-1}L\tilde{r}_{j-1})/(L^{-1}AL^{-T}\tilde{p}_j, \tilde{p}_j)$
    $x_j := x_{j-1} + \alpha_j L^{-T}\tilde{p}_j$
    $L\tilde{r}_j := L\tilde{r}_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    if $\|b - Ax_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (L\tilde{r}_j, M^{-1}L\tilde{r}_j)/(L\tilde{r}_{j-1}, M^{-1}L\tilde{r}_{j-1})$
    $\tilde{p}_{j+1} := L^{-1}L\tilde{r}_j + \beta_j\tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  We introduce the non-preconditioned residual $r_j := L\tilde{r}_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$\tilde{p}_1 := L^{-1}r_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (r_{j-1}, M^{-1}r_{j-1})/(L^{-1}AL^{-T}\tilde{p}_j, \tilde{p}_j)$
    $x_j := x_{j-1} + \alpha_j L^{-T}\tilde{p}_j$
    $r_j := r_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    if $\|r_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (r_j, M^{-1}r_j)/(r_{j-1}, M^{-1}r_{j-1})$
    $\tilde{p}_{j+1} := L^{-1}r_j + \beta_j\tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Consider the transformed search direction $p_j := L^{-T}\tilde{p}_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$L^{-T}\tilde{p}_1 := L^{-T}L^{-1}r_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (r_{j-1}, M^{-1}r_{j-1})/(AL^{-T}\tilde{p}_j, L^{-T}\tilde{p}_j)$
    $x_j := x_{j-1} + \alpha_j L^{-T}\tilde{p}_j$
    $r_j := r_{j-1} - \alpha_j L^{-1}AL^{-T}\tilde{p}_j$
    if $\|r_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (r_j, M^{-1}r_j)/(r_{j-1}, M^{-1}r_{j-1})$
    $L^{-T}\tilde{p}_{j+1} := L^{-T}L^{-1}r_j + \beta_j L^{-T}\tilde{p}_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Consider the transformed search direction $p_j := L^{-T}\tilde{p}_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$p_1 := M^{-1}r_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (r_{j-1}, M^{-1}r_{j-1})/(Ap_j, p_j)$
    $x_j := x_{j-1} + \alpha_j p_j$
    $r_j := r_{j-1} - \alpha_j L^{-1}Ap_j$
    if $\|r_j\|_2 < \varepsilon\|b\|_2$ then Stop
    $\beta_j := (r_j, M^{-1}r_j)/(r_{j-1}, M^{-1}r_{j-1})$
    $p_{j+1} := M^{-1}r_j + \beta_j p_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Finally, let $z_j := M^{-1}r_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
r_0 := b - A x_0                                         $r_0 := b - Ax_0$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_5$

- We apply the CG algorithm to form iterates denoted by u_j and approximate the solution of the split-preconditioned system $L^{-1}AL^{-T}u = L^{-1}b$.

  We denote the associated search directions by $\tilde{p}_j$.

  Finally, let $z_j := M^{-1}r_j$:

Algorithm 4 PCG: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$z_0 := M^{-1}r_0$
$p_1 := z_0$
for $j = 1, 2 \ldots$ do
    $\alpha_j := (r_{j-1}, z_{j-1})/(Ap_j, p_j)$
    $x_j := x_{j-1} + \alpha_j p_j$
    $r_j := r_{j-1} - \alpha_j L^{-1} A p_j$
    $z_j := M^{-1} r_j$
    if $\|r_j\|_2 < \varepsilon \|b\|_2$ then Stop
    $\beta_j := (r_j, z_j)/(r_{j-1}, z_{j-1})$
    $p_{j+1} := z_j + \beta_j p_j$
```

Preconditioned conjugate gradient (PCG) method, cont'd$_6$

- As previously mentioned, we now see that left- and split-preconditioning CG are equivalent methods.
- Analogously, the right-preconditioned coefficient matrix AM^{-1} is self-adjoint with respect to the M^{-1}-inner product, which can be leveraged into another derivation on an equivalent form of the PCG algorithm.

Preconditioned GMRES method

- For the GMRES method, **left-**, **right-** and **split-preconditioning** exhibit **fundamental differences**.
- GMRES applied to the **left-preconditioned** system $M^{-1}Ax = M^{-1}x$ yields:

Algorithm 5 Left-preconditioned GMRES: $(x_0, \varepsilon) \mapsto x_j$

```
$z_0 := M^{-1}b - M^{-1}Ax_0$
$\beta := \|z_0\|_2$
$v_1 := z_0/\beta$
for $j = 1, 2 \dots$ do
    $w := M^{-1}Av_j$
    $w := \Pi^{(j)}w$                                  ▷ $\Pi^{(j)}$ is a projector onto span$\{v_1, \dots, v_j\}^\perp$
    Compute $h_{1:j+1,j}$
    Solve for $\tilde{y} = \arg\min_{y \in \mathbb{R}^j} \|\beta e_1^{(j+1)} - \underline{H}_j y\|$      ▷ Still using Givens rotations
    if $\|\beta e_1^{(j+1)} - \underline{H}_j \tilde{y}\|_2 < \varepsilon \|M^{-1}b\|_2$ then Stop
    $v_{j+1} := w/h_{j+1,j}$
$x_j := x_0 + V_j \tilde{y}$
```

The residual $z_j = V_{j+1}(\beta e_1^{(j+1)} - \underline{H}_j \tilde{y})$ with norm $\|z_j\|_2 = \|\beta e_1^{(j+1)} - \underline{H}_j \tilde{y}\|_2$ is that of the left-preconditioned system, i.e., $z_j = M^{-1}b - M^{-1}Ax_j$.

Preconditioned GMRES method, cont'd$_1$

- Besides **evaluating** $z_j \mapsto Mz_j$, which is **not** even **always possible**, there is **no practical way to access the non-preconditioned residual** $b - Ax_j$ and its norm.

  Consequently, **convergence** has to be **monitored in terms of** the norm of the **preconditioned residual**, i.e., $\|z_j\|_2 < \varepsilon \|M^{-1}b\|_2$.

- The **basis** $v_1, \dots, v_j$ **generated by left-preconditioned GMRES** spans the Krylov subspace of $M^{-1}A$ generated by $z_0 := M^{-1}(b - Ax_0)$, i.e.,

$$\text{span}\{v_1, \dots, v_j\} = \mathcal{K}_j(M^{-1}A, z_0)$$

  and the computed Hessenberg matrix $\underline{H_j}$ is the projection of $M^{-1}A$ in this subspace, i.e.,

$$\underline{H_j} = V_{j+1}^T M^{-1} A V_j.$$

- The iterates of **left-preconditioned GMRES iterations** are given by

Find $x_j \in x_0 + \mathcal{K}_j(M^{-1}A, z_0)$ s.t. $x_j = \underset{x \in x_0 + \mathcal{K}_j(M^{-1}A, z_0)}{\arg\min} \|M^{-1}(b - Ax)\|_2$.

Preconditioned GMRES method, cont'd$_2$

- A **right-preconditioned** variant is obtained by applying GMRES to $AM^{-1}u = b$, from which an iterate is formed through $x_j := M^{-1}u_j$:

Algorithm 6 Right-preconditioned GMRES: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$\beta := \|r_0\|_2$
$v_1 := r_0/\beta$
**for** $j = 1, 2 \dots$ **do**
    $w := AM^{-1}v_j$
    $w := \Pi^{(j)}w$                                        ▷ $\Pi^{(j)}$ is a projector onto span$\{v_1, \dots, v_j\}^\perp$
    Compute $h_{1:j+1,j}$
    Solve for $\tilde{y} = \arg\min_{y \in \mathbb{R}^j} \|\beta e_1^{(j+1)} - \underline{H_j} y\|$        ▷ Still using Givens rotations
    **if** $\|\beta e_1^{(j+1)} - \underline{H_j}\tilde{y}\|_2 < \varepsilon \|b\|_2$ **then** Stop
    $v_{j+1} := w/h_{j+1,j}$
$x_j := x_0 + M^{-1}V_j\tilde{y}$
```

The residual $r_j = V_{j+1}(\beta e_1^{(j+1)} - \underline{H_j}\tilde{y})$ with norm $\|r_j\|_2 = \|\beta e_1^{(j+1)} - \underline{H_j}\tilde{y}\|_2$ is that of the non-preconditioned system, i.e., $r_j = b - Ax_j$.

Preconditioned GMRES method, cont'd$_3$

- The **basis** $v_1, \ldots, v_j$ **generated by right-preconditioned GMRES** spans the Krylov subspace of AM^{-1} generated by $r_0 := b - Ax_0$, i.e.,

  $$\text{span}\{v_1, \ldots, v_j\} = \mathcal{K}_j(AM^{-1}, r_0)$$

  and the computed Hessenberg matrix $\underline{H_j}$ is the projection of AM^{-1} in this subspace, i.e.,

  $$\underline{H_j} = V_{j+1}^T AM^{-1} V_j.$$

- The iterates of **right-preconditioned GMRES iterations** are given by

$$\text{Find } x_j \in x_0 + M^{-1}\mathcal{K}_j(AM^{-1}, r_0) \text{ s.t. } \underset{x \in x_0 + M^{-1}\mathcal{K}_j(AM^{-1}, r_0)}{\arg\min} \|b - Ax\|_2.$$

Preconditioned GMRES method, cont'd$_4$

- A **split-preconditioned** variant is obtained by combining both left- and right-preconditioning, this yields

Algorithm 7 Split-preconditioned GMRES: $(x_0, \varepsilon) \mapsto x_j$

```
$z_0 := M_L^{-1}b - M_L^{-1}Ax_0$
$\beta := \|z_0\|_2$
$v_1 := z_0/\beta$
for $j = 1, 2 \ldots$ do
    $w := M_L^{-1}AM_R^{-1}v_j$
    $w := \Pi^{(j)}w$                                        ▷ $\Pi^{(j)}$ is a projector onto span$\{v_1, \ldots, v_j\}^{\perp}$
    Compute $h_{1:j+1,j}$
    Solve for $\tilde{y} = \arg\min_{y \in \mathbb{R}^j} \|\beta e_1^{(j+1)} - \underline{H}_j y\|$     ▷ Still using Givens rotations
    if $\|\beta e_1^{(j+1)} - \underline{H}_j\tilde{y}\|_2 < \varepsilon\|M_L^{-1}b\|_2$ then Stop
    $v_{j+1} := w/h_{j+1,j}$
$x_j := x_0 + M_R^{-1}V_j\tilde{y}$
```

The residual $z_j = V_{j+1}(\beta e_1^{(j+1)} - \underline{H}_j\tilde{y})$ with norm $\|z_j\|_2 = \|\beta e_1^{(j+1)} - \underline{H}_j\tilde{y}\|_2$ is that of the left-preconditioned system, i.e., $z_j = M_L^{-1}b - M_L^{-1}Ax_j$.

Preconditioned GMRES method, cont'd$_5$

- Just like for the split-preconditioned variant, besides **evaluating** $z_j \mapsto M_L z_j$, which is **not** even **always possible**, there is **no practical way to access the non-preconditioned residual** $b - Ax_j$ and its norm. Consequently, **convergence** has to be **monitored in terms of** the norm of the **left-preconditioned residual**, i.e., $\|z_j\|_2 < \varepsilon \|M_L^{-1} b\|_2$.
- The **basis** $v_1, \dots, v_j$ **generated by split-preconditioned GMRES** spans the Krylov subspace of $M_L^{-1} A M_R^{-1}$ generated by $z_0 := M_L^{-1}(b - Ax_0)$, i.e.,

$$\text{span}\{v_1, \dots, v_j\} = \mathcal{K}_j(M_L^{-1} A M_R^{-1}, z_0)$$

and the computed Hessenberg matrix $\underline{H_j}$ is the projection of $M_L^{-1} A M_R^{-1}$ in this subspace, i.e.,

$$\underline{H_j} = V_{j+1}^T M_L^{-1} A M_R^{-1} V_j.$$

- The iterates of **split-preconditioned GMRES iterations** are given by

$$\begin{aligned} \text{Find } & x_j \in x_0 + M_R^{-1} \mathcal{K}_j(M_L^{-1} A M_R^{-1}, z_0) \\ \text{s.t. } & x_j = \arg \min_{x \in x_0 + M_R^{-1} \mathcal{K}_j(M_L^{-1} A M_R^{-1}, z_0)} \|M_L^{-1}(b - Ax)\|_2. \end{aligned}$$

Preconditioned GMRES method, cont'd$_6$

- In summary, each variant of preconditioned of the GMRES method grants access to a different form of residual:
  - Left-preconditioning: $M^{-1}(b - Ax_j)$,
  - Right-preconditioning: $b - Ax_j$,
  - Split-preconditioning: $M_L^{-1}(b - Ax_j)$.
- Monitoring convergence through the norm of these different forms of residuals **affects the stopping criterion**.
- In the case of the **left-** and **split-preconditioned variants**, **the iteration may stop either prematurely** or, **with delay**.

  This phenomenon is **more likely to happen when** M (for left-preconditioning) or M_L (for split-preconditioning) **is very ill-conditioned**.

Preconditioned GMRES method, cont'd$_7$

- The search space of the left- and right-preconditioned variants are related as follows:

$$\begin{aligned}
M^{-1}\mathcal{K}_j(AM^{-1}, r_0) &= M^{-1}\mathsf{span}\{r_0, AM^{-1}r_0, \dots, (AM^{-1})^{j-1}r_0\} \\
&= \mathsf{span}\{M^{-1}r_0, M^{-1}AM^{-1}r_0, \dots, M^{-1}(AM^{-1})^{j-1}r_0\} \\
&= \mathsf{span}\{M^{-1}r_0, (M^{-1}A)M^{-1}r_0, \dots, (M^{-1}A)^{j-1}M^{-1}r_0\} \\
&= \mathsf{span}\{z_0, (M^{-1}A)z_0, \dots, (M^{-1}A)^{j-1}z_0\} \\
&= \mathcal{K}_j(M^{-1}A, z_0).
\end{aligned}$$

Thus, the left- and right-preconditioned GMRES iterates are in the **same Krylov subspace**.

The only difference is that the **left-preconditioned** variant **minimizes** the **left-preconditioned residual**, while the **right-preconditioned** variant **minimizes** the **actual residual**, but **both over the same subspace**.

Other methods

- **All** the **linear iterative solvers** we presented in class **can be preconditioned**.

  This is the case for both **stationary** and **Krylov-based** solvers:

  - See Jacobi (1845) for a **preconditioned Jacobi** iteration.
  - See Algo. 4P in Greenbaum (1997) for a **preconditioned MINRES**.
- We already saw that the concept of **preconditioning** can be extended to **eigenvalue problems**. E.g., preconditioning plays an important role in:
  - **LOBPCG** (Lecture 10),
  - **Jacobi-Davidson** method (Lecture 12).
- **Most problems** which are solved approximately by an iterative method **can be preconditioned**. E.g.,
  - **Least-squares** problems (Lecture 7),
  - **Matrix function** evaluation (Lecture 18),
  - ...

Jacobi, C. G. J. (1845). Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkom- menden lineären Gleichungen. Astronomische Nachrichten, 22(20):297–306.

Greenbaum, A. (1997). Iterative methods for solving linear systems. Society for Industrial and Applied Mathematics.

Flexible variants

- Until now, we have assumed $x \mapsto M^{-1}x$ is a **linear map**, that is, M is **constant**, i.e., it does not vary depending on what it's applied to.

  Formally, this reads $M^{-1}(\alpha x + y) = \alpha M^{-1}x + M^{-1}y$ for all $\alpha \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$.
- As an alternative, we mentioned that **preconditioning** is sometimes **achieved by applying an iterative solver**, or any other procedure **that can approximate the solution of** $Mz = x$.

  Typically, such procedures are *not* linear maps, in a way, that is, M is not constant, it *varies*, i.e., we have $x \mapsto M^{-1}(x)x$.
- The preconditioned variants of iterative linear solvers we presented until now cannot accommodate for such *varying* preconditioners.

  Instead, **flexible variants** of those solvers need be deployed to allow for the use of such *varying* preconditioners while still guaranteeing convergence.

Flexible GMRES

- A **flexible** variant of the right-**preconditioned GMRES** method was proposed by Saad (1993). We present this method here.
- The **right-preconditioned GMRES** iterates are given by

$$x_j = x_0 + M^{-1}V_j\tilde{y} = x_0 + \sum_{i=1}^{j} \tilde{y}_i M^{-1} v_i \text{ where } \tilde{y} =: \begin{bmatrix} \tilde{y}_1 \\ \vdots \\ \tilde{y}_j \end{bmatrix}$$

so that $x_j - x_0 = \sum_{i=1}^{j} \tilde{y}_i M^{-1} v_i$, i.e., $x_j - x_0$ is a **linear combination of** the **preconditioned Krylov basis vectors** $M^{-1}v_1$ through $M^{-1}v_j$. If M is a fixed precodnitioner, then we have

$$\sum_{i=1}^{j} \tilde{y}_i M^{-1} v_i = M^{-1} \sum_{i=1}^{j} \tilde{y}_i v_i,$$

so that the vectors $M^{-1}v_1, \dots, M^{-1}v_j$ **need not be stored** to form x_j. Standard implementations store $v_1, \dots, v_j$, then form $\sum_{i=1}^{j} \tilde{y}_i v_i$, and, eventually, apply the preconditioner to the resulting vector, i.e., $M^{-1}(V_j\tilde{y})$.

Saad, Y. (1993). A flexible inner-outer preconditioned GMRES algorithm. SIAM Journal on Scientific Computing, 14, 461–469.

Flexible GMRES, cont'd$_1$

- Now, let us consider the case of a **varying preconditioner**, so that the mapping $x \mapsto M^{-1}(x)x$ is not linear, i.e., $M^{-1}(x)$ is not fixed.

  Then, the right-preconditioned GMRES iterate is given by

  $$x_j = x_0 + \sum_{i=1}^{j} \tilde{y}_i M_i^{-1} v_i = x_0 + Z_j \tilde{y} \;\text{ where }\; \tilde{y} =: \begin{bmatrix} \tilde{y}_1 \\ \vdots \\ \tilde{y}_j \end{bmatrix}$$

  in which $\tilde{y}$ is computed as before, and $Z = [z_1, \dots, z_j]$, where $z_i := M_i^{-1} v_i$ for $i = 1, \dots, j$.

  In this case, the *preconditioner cannot be factored out*, and the vectors $z_1 := M_1^{-1} v_1, \dots, z_j := M_j^{-1} v_j$ **must all be stored to form** x_j.

Flexible GMRES, cont'd$_2$

- Thus, a natural modification of the right-preconditioned GMRES algorithm to allow for the use of varying preconditioners is as follows:

Algorithm 8 Flexible (right-preconditioned) GMRES: $(x_0, \varepsilon) \mapsto x_j$

```
$r_0 := b - Ax_0$
$\beta := \|r_0\|_2$
$v_1 := r_0/\beta$
for $j = 1, 2 \ldots$ do
    $z_j := M^{-1}v_j$
    $w := Az_j$
    $w := \Pi^{(j)}w$                                    ▷ $\Pi^{(j)}$ is a projector onto span$\{v_1, \ldots, v_j\}^\perp$
    Compute $h_{1:j+1,j}$
    Solve for $\tilde{y} = \arg\min_{y \in \mathbb{R}^j} \|\beta e_1^{(j+1)} - \underline{H}_j y\|$    ▷ Still using Givens rotations
    if $\|\beta e_1^{(j+1)} - \underline{H}_j \tilde{y}\|_2 < \varepsilon\|b\|_2$ then Stop
    $v_{j+1} := w/h_{j+1,j}$
$x_j := x_0 + Z_j\tilde{y}$
```

The main difference with the standard right-preconditioned GMRES is that Z_j **must be stored in addition to** V_j.

Flexible GMRES, cont'd$_3$

- Clearly, the flexible GMRES (FGMRES) method is equivalent to the standard right-preconditioned variant when the preconditioned is constant, i.e., when $M_1 = \cdots = M_j = M$.
- In practice, the choice of the $z_1, \dots, z_j$ vectors is important, to the point that, a "**poor selection**" of those vectors can go as far as causing the procedure to **breakdown**.
- In the standard right-preconditioned variant, the Arnoldi relation is given by

  $$AM^{-1}V_j = V_{j+1}\underline{H_j}.$$

  This relation does not hold anymore with FGMRES. Instead, we have

  $$AZ_j = V_{j+1}\underline{H_j}.$$

  It follows from our definition of the iterate in FGMRES that

  $$x_j \in x_0 + \text{span}\{z_1, \dots, z_j\}.$$

Flexible GMRES, cont'd$_4$

- Analogously to the case of the standard right-preconditioned variant, the FGMRES residual is recast as follows using the Arnoldi relation:

$$\begin{aligned} r_j := b - Ax_j &= b - Ax_0 - AZ_j\tilde{y} \\ &= r_0 - V_{j+1}\underline{H_j}\tilde{y} \\ &= V_{j+1}(\beta e_1^{(j+1)} - \underline{H_j}\tilde{y}) \end{aligned}$$

  where $\tilde{y} = \arg\min_{y\in\mathbb{R}^j} \|\beta e_1^{(j+1)} - \underline{H_j}y\|_2$.
  That is, the FGMRES iterate still minimizes the non-preconditioned residual $b - Ax_j$. The difference is that it does it over a different subspace:

Theorem (Optimality of FGMRES iterates)

Let x_j be the FGMRES iterate after j iterations started with an initial guess x_0. Then,

$$x_j = \arg\min_{x\in x_0+\mathit{range}(Z_j)} \|b - Ax\|_2.$$

Flexible GMRES, cont'd$_5$

- The **search space of FGMRES**, that is

  $$(x_0+)\text{span}\{z_1, \dots, z_j\}$$

  is, in general, **not Krylov** anymore.

  Besides the extra memory requirement to store Z_j, **FGMRES requires no additional computation** with respect to the standard right-preconditioned variant.

  Actually, FGMRES even removes the need for a final preconditioner application to construct the last iterate.

- As mentioned earlier, the varying preconditioner applications, denoted by

  $$v_1 \mapsto M_1^{-1} v_1, \dots, v_j \mapsto M_1^{-1} v_j,$$

  can consist of any approximation procedure, that is, for example, any of the previously presented iterative solvers, i.e., stationary and Krylov-based.

  It is also **possible to combine several approximation procedures with complementary effects**, alternating from one iteration to another.

Preconditioners

Introduction

- The **first documented effort to precondition a linear system** was **by Jacobi** (1845), where the Jacobi iterative method was invented, and for which a preconditioner was introduced so as to accelerate the convergence of the Jacobi iteration.
- A **good preconditioner accelerates convergence**, i.e., it reduces the iteration count to reach convergence, while only **entailing a limited additional computational cost** per solver iteration.
  Finding a good preconditioner can be difficult.
  The **performance** of preconditioners is **highly problem-dependent**, and an **optimal general-purpose preconditioner** is **unlikely to exist**.
  Preconditioner design is sometimes described as a combination of art and science, often relies on heuristics with **limited theoretical insight**, and **significant input from domain expert knowledge**.
  In practice, **preconditioner design** is often the **source of significant efforts**, especially when multiple linear systems need be solved with one, or multiple similar coefficient matrices.

Jacobi, C. G. J. (1845). Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkom- menden lineären Gleichungen. Astronomische Nachrichten, 22(20):297–306.

Stationary iterative methods as preconditioners

- In Lecture 8, we saw that **stationary**, i.e., basic, **iterative methods** are induced by splittings $A = M - N$ of the coefficient matrix, where M is **non-singular**.
  Then, we know that the **solution** $A^{-1}b$ **of** $Ax = b$ is a **fixed-point** of
  $$x \mapsto M^{-1}Nx + M^{-1}b$$
  so that a **stationary iteration** is given by
  $$x_{j+1} = M^{-1}Nx_j + M^{-1}b.$$
  This iteration **attempts to solve**
  $$(I_n - M^{-1}N)x = M^{-1}b,$$
  in which using the splitting leads to
  $$(I_n - M^{-1}(M - A))x = M^{-1}b$$
  $$(I_n - I_n + M^{-1}A)x = M^{-1}b$$
  so that we iterate to solve the **left-preconditioned linear system**:
  $$M^{-1}Ax = M^{-1}b.$$

Stationary iterative methods as preconditioners, cont'd$_1$

- **Any splitting** $M - N$ of the coefficient matrix A with a **non-singular** M **leads to a preconditioner** M:

Definition (Jacobi preconditioner)

- The **Jacobi iteration** induces a so-called **Jacobi preconditioner** with $M := D_A$, where D_A denotes the **diagonal part of** A.
- **Applying the Jacobi preconditioner** to the coefficient matrix, i.e., the map $A \mapsto D_A^{-1}A$ is **equivalent to scaling all rows of** A to make the diagonal entries equal to one. This is also known as **diagonal scaling**.

Definition (Gauss-Seidel preconditioner)

- The **Gauss-Seidel iteration** induces a **Gauss-Seidel preconditioner** with $M := D_A - L_A$, which is the **lower-triangular part of** A.
- **Applying the Gauss-Seidel preconditioner**, i.e., evaluating $x \mapsto M^{-1}x$, **amounts to** complete **a triangular solve**, i.e., a forward substitution.

Stationary iterative methods as preconditioners, cont'd$_2$

Definition (Successive over-relaxation (SOR) preconditioner)

- The **successive over-relaxation** (**SOR**) iteration induces a **SOR preconditioner** with $M := D_A - \omega L_A$, which is also a **lower-triangular** matrix, and where ω is a relaxation parameter such that $0 < \omega < 2$.
- **Applying the SOR preconditioner**, i.e., evaluating $x \mapsto M^{-1}x$, also **amounts to** complete **a triangular solve**, i.e., a forward substitution.

- There exist more matrix splittings leading to stationary methods and associated preconditioners than what we covered in Lecture 8. E.g.,
  - **Symmetric Gauss-Seidel** (**SGS**) splitting (Saad, 2003),
  - **Symmetric successive over-relaxation** (**SSOR**) splitting (Saad, 2003),
  - **Hermitian and skew-Hermitian** (**HSS**) splitting (Bai & Pan, 2021),
  - **Normal and skew-Hermitian** (**NSS**) splitting (Bai & Pan, 2021),
  - **Positive-definite and skew-Hermitian** (**PSS**) splitting (Bai & Pan,'21),
  - **Richardson** splitting (Ciaramella & Gander, 2022).

Saad, Y. (2003). Iterative methods for sparse linear systems. Society for Industrial and Applied Mathematics.

Bai, Z. Z., & Pan, J. Y. (2021). Matrix analysis and computations. Society for Industrial and Applied Mathematics.

Ciaramella, G. & Gander, M. J., (2022). Iterative methods and preconditioners for systems of linear equations. Society for Industrial and Applied Mathematics.

Incomplete factorizations

- A common type of preconditioner, denoted by M_LM_R, and referred to as **incomplete factorization**, decomposes the coefficient matrix as follows:

  $$\boxed{A = M_LM_R + E}$$

  where M_L and M_R are known, at least in the sense that **one can efficiently evaluate**

  $$\boxed{x \mapsto (M_LM_R)^{-1}x},$$

  and the **error** term denoted by E is sufficiently **small in some sense**.

  Several types of incomplete factorizations exist, e.g.,

  - Incomplete **LU**: M_L and M_R are sparse lower- and upper-triangular.
  - Incomplete **Cholesky**: M_L is sparse lower-triangular, and $M_R = M_L^T$.
  - Incomplete **QR**: M_L is general sparse, and M_R is sparse upper-triangular.
  - Incomplete **Givens**: M_L is general sparse, and M_R is sparse upper-triangular. Both M_L and M_R are obtained using Givens rotations.

Incomplete LU (ILU) factorizations

- The incomplete LU (ILU) factorization was first introduced by Buleev (1960) and, independently, by Varga (1960).
- Given a sparse matrix $A \in \mathbb{R}^{n \times n}$, we denote its **sparsity pattern** as

$$\mathcal{S}(A) \subset \{(i,j),\, i,j \in \{1,2,\dots,n\}\}$$

  such that $(i,j) \in \mathcal{S}(A)$ iff $a_{ij} \neq 0$.
- **Fill-ins** are usually unavoidable when constructing the **LU factorization** of a **sparse matrix** by **Gaussian elimination**.
  That is, the **triangular factors** L and U of a "complete" LU factorization, i.e., $A = LU$, are usually **much less sparse** than the matrix A.
- An **ILU** factorization is obtained by **dropping** some of these **fill-ins**. Following prescribed sparsity patterns, we obtain

$$A = LU - E$$

  where E is the error induced by the dropped fill-ins.

Buleev, N. I. (1960). A numerical method for solving two-dimensional diffusion equations. Atomic Energy, 6, 222–224. (In Russian).

Buleev, N. I. (1960). A numerical method for solving two- and three-dimensional diffusion equations. Matematicheskii Sbornik, 51, 227–238. (In Russian).

Varga, R. S. (1960). Factorization and normalized iterative methods. In Boundary Problems in Differential Equations,

Incomplete LU (ILU) factorizations, cont'd$_1$

- **Different ILU preconditioners** $M = LU$ are obtained **depending on** the **dropping strategy** and **induced sparsity** $\mathcal{S}(M)$.

  In practice, **fill-ins** can be **dropped based on** different criteria, such as **position**, **value**, or combination of these factors.

  For a given sparsity pattern $\mathcal{S} := \mathcal{S}(L) \cup \mathcal{S}(U)$, an ILU factorization is computed as follows, in-place, within A:

Algorithm 9 General ILU factorization: $\mathcal{S} \mapsto (L, U)$

```
for $k = 1, \ldots, n-1$ do
    for $i = k+1, \ldots, n$ and $(i,k) \in \mathcal{S}$ do
        $a_{ik} := a_{ik} / a_{kk}$
        for $j = k+1, \ldots, n$ and $(i,j) \in \mathcal{S}$ do
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
```

- In theory, the prescribed sparsity pattern $\mathcal{S}$ is arbitrary.

  In practice, it is customary to, at least, include **all diagonal entries**, and **all non-zero entries of** A. Such a bare-minimum strategy with **no fill-ins** is referred to as the **ILU(0)** factorization.

Incomplete LU (ILU) factorizations, cont'd$_2$

- In practice, Algo. 9 is not recommended because, at the k-th step, all rows from $k+1$ to n are modified. To circumvent this unfavorable data movement, an IKJ version of the algorithm is introduced:

Algorithm 10 General ILU factorization: $\mathcal{S} \mapsto (L, U)$

```
For $(i,j) \notin \mathcal{S}$, set $a_{ij} := 0$
for $i = 2, \dots, n$ do
    for $k = 1, \dots, i-1$  and  $(i,k) \in \mathcal{S}$ do
        $a_{ik} := a_{ik}/a_{kk}$
        for $j = k+1, \dots, n$  and  $(i,j) \in \mathcal{S}$ do
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
```

- Algos. 9 and 10 are equivalent, but the latter is more cache-friendly.
- If $\mathcal{S}(A) \subseteq \mathcal{S}$, then Step 1 of Algo. 10 can be skipped.
- From the decomposition $A = LU - E$, we have

$$a_{ij} = \sum_{k=1}^{i} \ell_{ik} u_{kj} - e_{ij}$$

Incomplete LU (ILU) factorizations, cont'd$_3$

where

$$u_{kj} = \begin{cases} a_{kj} - \sum_{i=1}^{k-1} \ell_{ki} u_{ij} & \text{for } k \leq j \leq n \text{ and } (k,j) \in \mathcal{S}, \\ 0 & \text{for } k \leq j \leq n \text{ and } (k,j) \notin \mathcal{S}, \end{cases}$$

$$\ell_{ik} = \begin{cases} \left(a_{ik} - \sum_{j=1}^{k-1} \ell_{ij} u_{jk} \right) \Big/ u_{kk} & \text{for } k < i \leq n \text{ and } (i,k) \in \mathcal{S}, \\ 0 & \text{for } k < i \leq n \text{ and } (i,k) \notin \mathcal{S} \end{cases}$$

for $k = 1, \ldots, n$ so that

$$e_{ij} = \begin{cases} 0 & \text{for } (i,j) \in \mathcal{S}, \\ \sum_{k=1}^{i} \ell_{ik} u_{kj} & \text{for } (i,j) \notin \mathcal{S}. \end{cases}$$

- Clearly, defining ILU(0) by specifying $\mathcal{S}(LU) = \mathcal{S}(A)$ does not yield a unique factorization.

  Instead, the ILU(0) factorization is uniquely defined by construction through Algos. 9 and 10 and such that $\mathcal{S}(L) \cup \mathcal{S}(U) = \mathcal{S}(A)$.

Incomplete LU (ILU) factorizations, cont'd$_4$

- If the matrix A is **SPD**, then the approach of Algos. 9 and 10 is analogously applied to construct **incomplete Cholesky** (**IC**) factorizations.

  The **IC factorization** was popularized by the **analysis** of Meijerink & van der Vorst (1977).

- The **ILU(0)** and **IC(0)** factorizations, which are relatively **easy to implement**, **work well for** some problems.

  In particular, this is the case for **low-order discretizations of constant-coefficient elliptic PDEs**, which often leads to **diagonally dominant matrices**.

- **For more challenging problems, ILU(0)/IC(0) do not approximate A sufficiently well to constitute good preconditioners**.

  For such problems, **more accurate incomplete factorizations** need be **developed**, which is done **by incorporating some level of fill-ins** in the prescribed sparsity pattern.

Meijerink, J. A. & van der Vorst, H. A. (1977). An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comp., 31.

Incomplete LU (ILU) factorizations, cont'd$_5$

- A **level-of-fill** is a **non-negative integer attributed to each entry of** A during the construction of an incomplete factorization.

  The **level-of-fill** of a_{ij}, which we denote by lev_{ij}, is **initialized** as follows:

  $$lev_{ij} := \begin{cases} 0 & \text{if } i = j \text{ or } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases} \tag{1}$$

  When a_{ij} is updated by **Gaussian elimination**, i.e., after

  $$a_{ij} := a_{ij} - a_{ik} * a_{kj},$$

  the **level-of-fill** needs be **updated too**. Different approaches exist:

  - A basic **updating rule** for the **level-of-fill** is

    $$lev_{ij} := \min\{lev_{ij}, lev_{ik} + lev_{kj}\}.$$

    Then, the level-of-fill never increases, and its final value is either 0 (keep the fill-in), or ∞ (drop the fill-in).

Incomplete LU (ILU) factorizations, cont'd$_6$

- A **more nuanced and practical updating rule** is given by

$$lev_{ij} := \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}. \quad (2)$$

  Then, if a_{ij} changes from zero to non-zero, its level-of-fill changes from ∞ to a finite non-zero integer.

- Different **ILU preconditioners** are **defined based on different dropping strategies using** a criterion based on **the level-of-fill** lev_{ij}.
- The **ILU factorization of level** p, denoted by ILU(p), defines a sparsity pattern given by

$$\mathcal{S}_p := \{(i,j) \text{ such that } lev_{ij} \leq p,\ i,j = 1,\dots,n\}$$

  where lev_{ij} is the **final value of the level-of-fill** after all updates have been made.

  The ILU factorization of level $p = 0$ is consistent with our previous definition of ILU(0).

Incomplete LU (ILU) factorizations, cont'd$_7$

- Although different updating rule of the level-of-fill exist, using the update formula of Eq. (2) with an initialization by Eq. (1), the algorithm used to construct the ILU(p) factorization is as follows:

Algorithm 11 ILU(p) factorization: $p \mapsto (L, U)$

```
Set initial value of $lev_{ij}$ with Eq. (1)
for $i = 2, \dots, n$ do
    for $k = 1, \dots, i-1$ and $lev_{ik} \leq p$ do
        $a_{ik} := a_{ik}/a_{kk}$
        for $j = k+1, \dots, n$ do
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
            $lev_{ij} := \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}$
    for $j = 1, \dots, n$ and $lev_{ij} > p$ do
        $a_{ij} := 0$
```

- For most problems, ILU(1) provides considerably improved preconditioning performance compared to ILU(0).
- Higher levels ILU(p) are rarely used in practice due to rapid increases of memory requirement and computational cost.

Incomplete LU (ILU) factorizations, cont'd$_8$

- The ILU(p) algorithms have several drawbacks:
  - The **amount of fill-ins** and computational costs are **not predictable** for $p > 0$.
  - The cost of **updating** the **levels can be expensive**.
  - **Level-of-fills** can be **poor indicators of entry sizes for indefinite matrices**, ultimately **leading to inaccurate factorizations**, that is, large error matrices E, and degraded convergence behaviors.

Incomplete LU (ILU) factorizations, cont'd$_9$

- **For indefinite** or **strongly non-diagonally dominant matrices**, **ILU(p)** may only **lead to poor approximations** of the coefficient matrix.

  This is caused by entries having high values of level-of-fill, despite having a large magnitude.

- An alternative strategy exists where **fill-ins** are **dropped according to their absolute values** rather than their positions.

  For this, a **drop tolerance** τ is introduced and **used in the dropping rule**.

  That is, only fill-ins that are greater than τ in absolute value are stored and used, which indicates that the sparsity pattern is determined dynamically.

- To remedy the case of badly scaled matrices, it is suggested to use a **relative drop tolerance**.

  That is, a fill-in is dropped if it is less than the product of τ with the norm of either the row or the column in which the fill-in is located.

  A drawback of this approach is that it is difficult to predict the amount of storage needed to store the ILU factors.

Incomplete LU (ILU) factorizations, cont'd$_{10}$

- Saad (1994) proposed the **dual threshold ILU** factorization, denoted by **ILUT(p, τ)** in which the following two dropping strategies are used:
  - **DS1**. The fill-in is dropped if it is less than the relative tolerance obtained by multiplying τ with the norm of the original row or column.
  - **DS2**. Keep at most the p largest entries in each of the L and U parts of the row or column, in addition to the diagonal entry, which is always kept.

  In the resulting ILUT(p, τ) procedure, τ is a parameter that helps reducing the computational cost, while p helps control the memory requirement.

  The ILUT(p, τ) algorithm is derived from Gaussian elimination employing the threshold strategies **DS1** and **DS2**.

Saad, Y. (1994). ILUT: A dual threshold incomplete LU factorization, Numerical Linear Algebra with Applications, 1, 387–402.

Incomplete LU (ILU) factorizations, cont'd$_{11}$

In the resulting ILUT(p, τ) algorithm, w is a full-length working row vector with w_k being its k-th entry, and $\| \cdot \|$ is a suitably chosen norm.

Algorithm 12 ILUT(p, τ) factorization: $(p, \tau) \mapsto (L, U)$

```
for $i = 1, \dots, n$ do
    for $k = 1, \dots, n$ do
        $w_k := a_{ik}$                                    ▷ Copy the $i$-th row of $A$ to $w$
    $\tau_i := \tau \|w\|$                                 ▷ Relative drop tolerance
    for $k = 1, \dots, i-1$ and $w_k \neq 0$ do
        $w_k := w_k / a_{kk}$
        if $|w_k| < \tau_i$ then                           ▷ Apply DS1 on $w_k$
            $w_k := 0$
        else
            for $j = k+1, \dots, n$ do
                $w_j := w_j - w_k u_{kj}$
    for $k = i+1, \dots, n$ and $|w_k| < \tau_i$ do        ▷ Apply DS1 on $w$
        $w_k := 0$
```

Incomplete LU (ILU) factorizations, cont'd$_{11}$

In the resulting ILUT(p, τ) algorithm, w is a full-length working row vector with w_k being its k-th entry, $\|\cdot\|$ is a suitably chosen norm.

Algorithm 12 cont'd ILUT(p, τ) factorization: $(p, \tau) \mapsto (L, U)$

```
    ▷ Apply DS2 on $w$
    Find the largest $p$ entries of $w[1 : i-1]$ and drop the others
    Find the largest $p$ entries of $w[i+1 : n]$ and drop the others
    for $k = 1, \ldots, i-1$ do
        $\ell_{ik} := w_k$
    for $k = i, \ldots, n$ do
        $u_{ik} := w_k$
```

Incomplete QR factorizations

- The **existence of** an **ILU** factorization is **not guaranteed for general matrices**, its **computation** is **prone to breakdown**, and **instabilities** can occur **in** the **sparse triangular solves**.

  These drawbacks motivate the consideration of other types of incomplete factorizations.

- **Incomplete QR** (**IQR**) factorizations of the form $A = QR - E$, where Q is a **general matrix**, **more or less sparse**, such that E and $Q^TQ - I_n$ are **small in some sense**, and R is **upper-triangular sparse**.

  Saad (1988) constructed **IQR** factorizations using **incomplete modified Gram-Schmidt** (**IMGS**) procedures incorporating dropping rules.

  Existence and **stability** of **IQR** factorizations obtained by **only dropping entries of the upper-triangular factor** were proved by Wang et al. (1997).

  As observed by Saad (1988), the **incomplete factor** Q is **only** guaranteed to be **non-singular for** very **low sparsity levels**.

Saad, Y. (1988). Preconditioning techniques for nonsymmetric and indefinite linear systems, Journal of Computational and Applied Mathematics, 24, 89–105.

Wang, X.-G., Gallivan, K. A. & Bramley, R. B. (1997). CIMGS: An incomplete orthogonal factorization preconditioner, SIAM Journal on Scientific Computing, 18, 516–536.

Incomplete QR factorizations, cont'd

- An alternative to Gram-Schmidt procedures for the construction of incomplete QR factorizations is the use of **Givens rotations**.

  For this, the **incomplete Givens orthogonalization (IGO)** was proposed by Zai et al. (2001). For each column, the IGO method does three things:

  (a) **Annihilate**, using Givens rotations, the **non-zero entries located in the strictly lower-triangular part** of A from the bottom up to the first sub-diagonal;

  (b) **Update the incomplete orthogonal matrix** Q by **post-multiplying by the transpose of the Givens rotation using some dropping rule**;

  (c) After Steps (a) and (b) have been done for all non-zeros in the current column, **form the corresponding row of the incomplete upper-triangular matrix** R **using some dropping rule**.

- Detailed implementations of the IMGS and IGO methods can be found in Bai & Pan (2021).

Bai, Z.-Z., Duff, I. S. & Wathen, A. J. (2001). A class of incomplete orthogonal factorization methods. I: Methods and theories, BIT, 41 53–70.

Bai, Z. Z., & Pan, J. Y. (2021). Matrix analysis and computations. Society for Industrial and Applied Mathematics.

Sparse approximate inverses (SPAI)

- **Sparse approximate inverse** (**SPAI**) preconditioning, which consists of building a sparse matrix M^{-1} which is close to A^{-1} in some sense, **offers considerable advantages** over incomplete factorizations **for parallel processing**.

  The main advantages of the SPAI approach are

  - **Inherently parallel construction**, because each column (or row) of M^{-1} can be processed independently;
  - **Autonomous identification of non-zero entries**;
  - **Preconditioner application** done by **sparse matrix-vector** (**SpMV**) product, which is also **parallelizable**.
- The **inverse of a sparse matrix** being **generally dense**, there is, a priori, **no guarantee** that a sparse approximate inverse M^{-1} **will be close to** A^{-1}.

  But, as the **matrix inverse** A^{-1} often **contains numerous entries of small magnitude**, SPAI matrices often can reasonably approximate A^{-1}.
- An SPAI matrix M^{-1} can be expressed either as a single matrix, or as the product of two or more matrices.

Sparse approximate inverses (SPAI), cont'd$_1$

- Designing an **SPAI** preconditioner amounts to **minimize the Frobenius norm of the residual matrix**:

$$F(M^{-1}) := \|I_n - AM^{-1}\|_F^2.$$

A matrix M^{-1} for which the value of $F(M^{-1})$ is small is an accurate **right-approximat**e inverse of A.
Alternatively, setting the objective function to

$$F(M^{-1}) := \|I_n - M^{-1}A\|_F^2 \text{ and } F(M_L^{-1}, M_R^{-1}) := \|I_n - M_L^{-1}AM_R^{-1}\|_F^2$$

enables the design of **left-** and **split-approximate** inverses.

- The objective function for the **right-approximate** inverse can be recast into

$$F(M^{-1}) = \sum_{j=1}^{n} \left\| e_j^{(n)} - Am_j^{-1} \right\|_2^2 \text{ where } m_j^{-1} := M^{-1}e_j^{(n)}$$

is the j-th column of M^{-1}. This makes the **minimization of** $F(M^{-1})$ **embarrassingly parallelizable**.

Sparse approximate inverses (SPAI), cont'd$_2$

- To minimize the objective function F, we can use a **global iteration** where M^{-1} is **unknown**, and which we solve for by a **descent-type method**. Using the **inner-product** $(X, Y) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n} \mapsto \text{tr}(Y^T X)$, for a given **iterate** M_i^{-1}**of right-approximate inverse** of $A \in \mathbb{R}^{n\times n}$, a new iterate is obtained as follows:

$$\begin{aligned} \text{Find } \; & M_{i+1}^{-1} \in M_i^{-1} + \text{span}\{G_i\} \\ \text{such that } \; & R_{i+1} := I_n - AM_{i+1}^{-1} \perp A\,\text{span}\{G_i\}, \end{aligned}$$

where $G_i \in \mathbb{R}^{n\times n}$ is a given **search direction**, and so that

$$M_{i+1}^{-1} = M_i^{-1} + \alpha_i G_i \;\text{ where }\; \alpha_i = \frac{(R_i, AG_i)}{(AG_i, AG_i)} = \frac{\text{tr}(R_i^T AG_i)}{\|AG_i\|_F^2}.$$

Since $X \in \mathbb{R}^{n\times n} \mapsto \|X\|_F$ is a norm induced by the inner-product, we can show that M_{i+1}^{-1} is **optimal** in the sense that:

$$\|R_{i+1}\|_F = \|I_n - AM_{i+1}^{-1}\|_F = \min_{M^{-1}\in M_i^{-1}+\text{span}\{G_i\}} \|I_n - AM^{-1}\|_F.$$

Sparse approximate inverses (SPAI), cont'd$_3$

- The simplest choice of **search direction** is $G_i := R_i = I_n - AM_i^{-1}$. The resulting approach is analogous to the **minimal residual** method (see Section 5.3.2 in Saad, 2003), but applied to the system $AM^{-1} = I_n$. Similarly as with the **minimal residual** method, we have a **monotonic decrease of residual norm**, that is, $\|R_{i+1}\|_F \leq \|R_i\|_F$. In practice, with this approach, **the iterate** M_i^{-1} tends to **become denser** and denser from one iteration to another. In order to **contain** the amount of **non-zero components** in M^{-1}, **numerical droppings** are applied, which leads to the following algorithm:

Algorithm 13 Global minimal residual descent

```
Set an initial $M^{-1} \in \mathbb{R}^{n \times n}$
while Not converged do
    $C := AM^{-1}$
    $G := I_n - C$
    $\alpha := \mathrm{tr}(G^T AG)/\|C\|_F^2$        ▷ $G := R \implies (R, AG) = (G, AG) = \mathrm{tr}(G^T AG)$
    $M^{-1} := M^{-1} + \alpha G$
    Apply numerical droppings to $M^{-1}$
```

Sparse approximate inverses (SPAI), cont'd$_4$

- An alternative to the global minimal residual descent is by **setting the search direction to** the **opposite of the gradient of the residual norm**. One way to reveal this gradient is through Taylor expansion, i.e., the perturbed objective function reads

$$F(M^{-1}+E)=F(M^{-1})+(\nabla_{M^{-1}}F,E)+o(\|E\|_F)$$

where E is a small perturbation. Then, as we let $R:=I_n-AM^{-1}$, we get

$$\begin{aligned}
F(M^{-1}+E)-F(M^{-1}) &= \|I_n-A(M^{-1}+E)\|_F-\|I_n-AM^{-1}\|_F\\
&= \|R-AE\|_F-\|R\|_F\\
&= \mathrm{tr}\left((R-AE)^T(R-AE)\right)-\mathrm{tr}(R^TR)\\
&= \mathrm{tr}\left(RR^T-R^TAE-(AE)^TR+(AE)^TAE\right)-\mathrm{tr}(R^TR)\\
&= -\mathrm{tr}(R^TAE)-\mathrm{tr}((AE)^TR)+\mathrm{tr}((AE)^TAE)\\
&= -(R,AE)-(AE,R)+\|AE\|_F^2\\
&= -2(R,AE)+\|AE\|_F^2=-2(A^TR,E)+\|AE\|_F^2
\end{aligned}$$

which indicates that the gradient is given by $-2A^TR$.

Sparse approximate inverses (SPAI), cont'd$_5$

- A **steepest descent** method is obtained by setting the search direction to the opposite of the gradient of the residual norm, i.e., $G_i := A^T R_i$.

  The resulting algorithm is given by:

Algorithm 14 Global steepest descent

```
Set an initial $M^{-1} \in \mathbb{R}^{n \times n}$
while Not converged do
    $R := I_n - AM^{-1}$
    $G := A^T R$
    $\alpha := \|G\|_F^2 / \|AG\|_F^2$      ▷ $G := R^T A \implies (R, AG) = (A^T R, A^T R) = \|G\|_F^2$
    $M^{-1} := M^{-1} + \alpha G$
    Apply numerical droppings to $M^{-1}$
```

Sparse approximate inverses (SPAI), cont'd$_6$

- Remember that the residual norm of a right-approximate inverse can be decomposed into

$$F(M^{-1}) = \|I_n - AM^{-1}\|_F = \sum_{j=1}^{n} \left\| e_j^{(n)} - Am_j^{-1} \right\|_2^2 \text{ where } m_j^{-1} := M^{-1} e_j^{(n)}$$

is the j-th column of M^{-1}.

This suggests a column-oriented approach where n independent linear systems

$$Am_j^{-1} = e_j^{(n)} \text{ for } j = 1, 2, \dots, n$$

are approximately solved using a **sparse iterative solver**.

That is, the **initial guess** is **sparse**, and the **subsequent iterates remain sparse**.

For the case of Arnoldi-based approaches, the Arnolid basis vectors are also sparse.

Other preconditioners

- Besides what we covered here, there are numerous other types of preconditioners. Some of these are
  - **Multigrid methods**: Class of iterative solvers that rely on the application of stationary methods (e.g., Gauss-Seidel, Jacobi, ...) at different resolution scales, from fine grids to coarser, lower-dimensional ones, all the way down to the coarsest level, where a more accurate solve is conducted.
  - **Domain decomposition methods**: Techniques that partition the computational domain into smaller subdomains, solve subproblems on each subdomain (possibly in parallel), and combine the solutions using appropriate interface conditions to reconstruct the global solution.
  - **Block preconditioners**: Methods that exploit the block structure of matrices by constructing preconditioners based on approximations of diagonal blocks, Schur complements, or block factorizations, often leading to more efficient preconditioning for structured problems.
  - **Hierarchical preconditioners**: Matrices based on the approximation of certain blocks of a matrix by low-rank decompositions.

Homework problems

Homework problems

Turn in **your own** solution to **Pb. 31**:

Pb. 31 Let $A \in \mathbb{R}^{n\times n}$ be a non-singular coefficient matrix for the linear system $Ax = b$, and $M \in \mathbb{R}^{n\times n}$ be a non-singular preconditioner which admits a decomposition $M = M_L M_R$.

a. Show that $M^{-1}A$, AM^{-1} and $M_L^{-1}AM_R^{-1}$ are similar.

b. Given a right-eigenvector y of $M^{-1}A$, find an associated right-eigenvector for AM^{-1}, and another one for $M_L^{-1}AM_R^{-1}$.

Pb. 32 Let $A \in \mathbb{R}^{n\times n}$ be any matrix. Then, show that

a. $\|I_n - AM^{-1}\|_F$ is minimized by $M^{-1} \in \mathbb{R}^{n\times n}$ if $A^TAM^{-1} = A^T$.

b. $\|I_n - M^{-1}A\|_F$ is minimized by $M^{-1} \in \mathbb{R}^{n\times n}$ if $M^{-1}AA^T = A^T$.

Practice session

Practice session

1. Implement the PCG method.
2. Implement the split-preconditioned GMRES method.

References

References

- Bai, Z. Z., & Pan, J. Y. (2021). Matrix analysis and computations. Society for Industrial and Applied Mathematics.
- Saad, Y. (2003). Iterative methods for sparse linear systems. Society for Industrial and Applied Mathematics.