9.3 Transposed-Free Methods

Remark 9.14. Motivation. As already explained in Remark 9.4, the transposed matrix A^T is in a number of situations not available. Then, algorithms that require the multiplication with A^T cannot be used. □

Remark 9.15. Idea of the Conjugate Gradient Squared (CGS) method. The CGS method can be derived from the Bi-CG method, see Algorithm 9.8. Sucessive insertion shows that the residual vector can be expressed as

$$\underline{r}^{(k)} = \phi_k(A)\underline{r}^{(0)}, \quad k \ge 0, \tag{9.6}$$

where $\phi_k(A)$ is a polynomial of degree k with $\phi_k(0) = 1$, compare (7.1). Similarly, it is

$$\underline{p}_k = \psi_{k-1}(A)\underline{r}^{(0)}, \quad k \ge 1, \tag{9.7}$$

where $\psi_{k-1}(A)$ is a polynomial of degree $(k-1)$ with $\psi_{k-1}(0) = 1$. From Algorithm 9.8, it can be observed that the vectors $\underline{\tilde{r}}^{(k)}$ and $\underline{\tilde{p}}_k$ are defined in the same way with the same polynomials where A is replaced by A^T

$$\underline{\tilde{r}}^{(k)} = \phi_k\left(A^T\right)\underline{\tilde{r}}^{(0)}, \quad \underline{\tilde{p}}_k = \psi_{k-1}\left(A^T\right)\underline{\tilde{r}}^{(0)}. \tag{9.8}$$

Inserting (9.6) – (9.8) in the definition of ν_k in line 8 of Algorithm 9.8 gives

$$\nu_k = \frac{\left(\phi_{k-1}\left(A^T\right)\underline{\tilde{r}}^{(0)}\right)^T \phi_{k-1}(A)\underline{r}^{(0)}}{\left(\psi_{k-1}\left(A^T\right)\underline{\tilde{r}}^{(0)}\right)^T A\psi_{k-1}(A)\underline{r}^{(0)}} = \frac{\left(\underline{\tilde{r}}^{(0)}\right)^T \phi_{k-1}^2(A)\underline{r}^{(0)}}{\left(\underline{\tilde{r}}^{(0)}\right)^T A\psi_{k-1}^2(A)\underline{r}^{(0)}}. \tag{9.9}$$

Similarly, one obtains for the parameter in line 12

$$\mu_{k+1} = \frac{\left(\phi_k\left(A^T\right)\underline{\tilde{r}}^{(0)}\right)^T \phi_k(A)\underline{r}^{(0)}}{\left(\phi_{k-1}\left(A^T\right)\underline{\tilde{r}}^{(0)}\right)^T \phi_{k-1}(A)\underline{r}^{(0)}} = \frac{\left(\underline{\tilde{r}}^{(0)}\right)^T \phi_k^2(A)\underline{r}^{(0)}}{\left(\underline{\tilde{r}}^{(0)}\right)^T \phi_{k-1}^2(A)\underline{r}^{(0)}}. \tag{9.10}$$

Thus, if it would be possible to derive recursions for the vectors $\phi_k^2(A)\underline{r}^{(0)}$ and $\psi_{k-1}^2(A)\underline{r}^{(0)}$, then the computation of ν_k and μ_{k+1} would be possible without using the transposed of A.

The goal of CGS consists now, besides the formulation of the recursions, to compute iterates whose residual satisfy

$$\underline{r}^{(k)} = \phi_k^2(A)\underline{r}^{(0)}, \tag{9.11}$$

instead of (9.6).

To derive these recursions, one starts with a recursion for the polynomials. Inserting (9.6) and (9.7) in line 10 of Algorithm 9.8, one obtains

$$\phi_k(A)\underline{r}^{(0)} = \phi_{k-1}(A)\underline{r}^{(0)} - \nu_k A\psi_{k-1}(A)\underline{r}^{(0)},$$

hence the recursion for the polynomial is

$$\phi_k(t) = \phi_{k-1}(t) - \nu_k t\psi_{k-1}(t). \tag{9.12}$$

Similarly, one obtains from line 13

$$\psi_k(t) = \phi_k(t) + \mu_{k+1}\psi_{k-1}(t). \tag{9.13}$$

Taking the square of the polynomials gives

$$\begin{aligned}\phi_k^2(t) &= \phi_{k-1}^2(t) - 2\nu_k t\phi_{k-1}(t)\psi_{k-1}(t) + \nu_k^2 t^2\psi_{k-1}^2(t),\\ \psi_k^2(t) &= \phi_k^2(t) + 2\mu_{k+1}\phi_k(t)\psi_{k-1}(t) + \mu_{k+1}^2\psi_{k-1}^2(t).\end{aligned}$$

The difficulty for deriving a recursion comes from the cross terms. The idea to overcome this difficulty consists in constructing a recursion for one of the cross terms, too. It is with (9.13)

$$\begin{aligned}\phi_{k-1}(t)\psi_{k-1}(t) &= \phi_{k-1}(t)\left(\phi_{k-1}(t) + \mu_k\psi_{k-2}(t)\right)\\ &= \phi_{k-1}^2(t) + \mu_k\phi_{k-1}(t)\psi_{k-2}(t).\end{aligned}$$

Collecting all equations, (9.12), and (9.13), one obtains the following relations (where for clarity of presentation the dependency on t will be neglected)

$$\phi_k^2 = \phi_{k-1}^2 - \nu_k t\left(2\phi_{k-1}^2 + 2\mu_k\phi_{k-1}\psi_{k-2} - \nu_k t\psi_{k-1}^2\right), \tag{9.14}$$

$$\begin{aligned}\phi_k\psi_{k-1} &= \left(\phi_{k-1} - \nu_k t\psi_{k-1}\right)\psi_{k-1} = \phi_{k-1}\psi_{k-1} - \nu_k t\psi_{k-1}^2\\ &= \phi_{k-1}^2 + \mu_k\phi_{k-1}\psi_{k-2} - \nu_k t\psi_{k-1}^2,\end{aligned} \tag{9.15}$$

$$\psi_k^2 = \phi_k^2 + 2\mu_{k+1}\phi_k\psi_{k-1} + \mu_{k+1}^2\psi_{k-1}^2. \tag{9.16}$$

Note that in (9.14) one can use the known quantities ϕ_{k-1}^2 and ψ_{k-1}^2 for computing ν_k, see (9.9), and ϕ_{k-1}^2 and ϕ_{k-2}^2 for computing μ_k, see (9.10). Thus, the recursion (9.14) and (9.15) are well defined. After having computed ϕ_k^2, one can compute μ_{k+1}, which is needed in (9.16). Altogether, (9.14) – (9.16) is a recursion for the quantities ϕ_k^2, ψ_k^2, and $\phi_k\psi_{k-1}$.

An efficient implementation of the recursion leads to the CGS method. This method was developed in Sonneveld (1989). One step of this method requires two matrix-vector products with the matrix A but no product with A^T. CGS works well in many cases (Saad, 2003, Section 7.4.1), however rounding errors might become important since the residual polynomial is squared. An irregular convergence might occur that can even lead to an overflow. □

Remark 9.16. Idea of the Bi-Conjugate Gradient Stabilized (BiCGStab) method. The BiCGStab method is a generalization of the CGS method that was developed in van der Vorst (1992) to stabilize the CGS method. Instead of computing a residual vector of the form (9.11), the residual vector computed with BiCGStab should fulfill

$$\underline{r}^{(k)} = \pi_k(A)\phi_k(A)\underline{r}^{(0)},$$

where $\phi_k(t)$ is the polynomial associated with the Bi-CG algorithm and $\pi_k(t)$ is a new polynomial of degree k. The choice of $\pi_k(A)$ should contribute to the stabilization of the algorithm.

In the BiCGStab method, the polynomial $\pi_k(t)$ is defined by the recurrence

$$\pi_{k+1}(t) = (1 - \omega_k t)\,\pi_k(t), \quad \pi_0(t) = 1,$$

where $\omega_k \in \mathbb{R}$ has to be yet determined. The choice of this polynomial leads in fact to a more stable algorithm than CGS. The parameter ω_k is chosen in such a way that the norm of a certain residual that occurs in the algorithm is minimized, for more details see van der Vorst (1992) and (Saad, 2003, Section 7.4.2). □

Algorithm 9.17. Bi-Conjugate Gradient Stabilized (BiCGStab). Given a non-singular matrix $A \in \mathbb{R}^{n\times n}$, a right-hand side $\underline{b} \in \mathbb{R}^n$, an initial iterate $\underline{x}^{(0)} \in \mathbb{R}^n$ and a tolerance $\varepsilon > 0$.

```
 1. $\underline{r}^{(0)} = \underline{b} - A\underline{x}^{(0)}$, choose $\underline{\tilde{r}}^{(0)}$ arbitrarily such that
    $\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(0)} \neq 0$
 2. $\underline{p}_1 = \underline{r}^{(0)}$
 3. $k = 0$
 4. while $\left\|\underline{r}^{(k)}\right\|_2 > \varepsilon$
 5.     $k = k + 1$
 6.     $\underline{s} = A\underline{p}_k$
 7.     $\nu_k = \dfrac{\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(k-1)}}{\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{s}}$
 8.     $\underline{w} = \underline{r}^{(k-1)} - \nu_k \underline{s}$
 9.     $\underline{z} = A\underline{w}$
10.     $\omega_k = \dfrac{\underline{z}^T \underline{w}}{\underline{z}^T \underline{z}}$
11.     $\underline{x}^{(k)} = \underline{x}^{(k-1)} + \nu_k \underline{p}_k + \omega_k \underline{w}$
12.     $\underline{r}^{(k)} = \underline{w} - \omega_k \underline{z}$
13.     $\mu_{k+1} = \dfrac{\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(k)}}{\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(k-1)}} \dfrac{\nu_k}{\omega_k}$
14.     $\underline{p}_{k+1} = \underline{r}^{(k)} + \mu_{k+1}\left(\underline{p}_k - \omega_k \underline{s}\right)$
15. endwhile
```

□

Remark 9.18. BiCGStab.

- Each iteration requires two matrix-vector products.
- In van der Vorst (1992), it is proposed to use $\underline{\tilde{r}}^{(0)} = \underline{r}^{(0)}$.
- In exact arithmetics, BiCGStab terminates in at most n iterations.
- There is the possibility that BiCGStab terminates in finite precision without having computed the solution, e.g., if $\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(k-1)}$ is (close to) zero. Then, one can try to use a different vector $\underline{\tilde{r}}^{(0)}$ or one can switch to another method like GMRES.
- There are other variants of BiCGStab than given in Algorithm 9.17 that are equivalent in exact arithmetics but not in finite precision computation, see van der Vorst (1992). For instance, in MATLAB there is different variant implemented. Different variants might behave differently.
- One should compute and store $\left(\underline{\tilde{r}}^{(0)}\right)^T \underline{r}^{(k)}$ before line 13 to be used in the next iteration at line 7 and at line 13.
- BiCGStab is besides GMRES the most popular iterative scheme for solving large linear systems of equations with non-symmetric matrices.

□

Remark 9.19. Transposed-free QMR. There is also a transposed-free version of the QMR algorithm, see (Saad, 2003, Section 7.4.3). However, this algorithm is not popular. □

Chapter 10
Summary and Outlook

Remark 10.1. Practical use of iterative solvers. The core of many algorithms is the solution of linear systems of equations. There are many applications where these systems are large and the system matrix is sparse. In this situation, appropriate iterative methods are often the most efficient solver.

- In practice, iterative methods, like all Krylov subspace methods, are usually used in combination with a preconditioner.
- For systems with symmetric and positive definite matrix, the by far most popular method is PCG, see Algorithm 8.8.
- For general matrices, preconditioned GMRES and BiCGStab are popular. Occasionally, also CGS is used.
- If the preconditioner is not a matrix but an iterative method, it might change from iteration to iteration. There are so-called flexible methods, like flexible GMRES, to cope with this situation.
- Special forms of the matrix require sometimes the construction of special preconditioners. For instance, standard preconditioners, like Jacobi or Gauss–Seidel, cannot be applied for matrices that are of so-called saddle point form
$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & 0 \end{pmatrix},$$
since there are zero entries in the diagonal.

□

Remark 10.2. Sparse direct solvers. In the past decades, there has been an enormous development of direct solvers for sparse linear systems of equations, so-called sparse direct solvers. Popular packages are UMFPACK, PARDISO, and MUMPS. In particular, for linear systems of equations arising in the discretization of partial differential equations, there are several comparisons with iterative solvers. It turned out that it makes a difference whether the partial differential equation is defined in a two-dimensional or three-dimensional domain.

- In 2d, sparse direct solvers often outperform iterative solvers for small and medium-sized systems, up to a matrix dimension of around 10^6.
- In 3d, which is the natural dimension for many real world problems, iterative methods are usually more efficient if the dimension of the matrix exceeds 10^4. That means, sparse direct solvers should be used only for small problems.

The reason for these differences is the different sparsity pattern of matrices from discretizations of partial differential equations in 2d and 3d. □

Remark 10.3. Multigrid methods. The discretization of partial differential equations is usually based on a triangulation of the underlying domain. Consider a uniform triangulation with mesh size h. Then, it turns out that the spectral condition number of discretization matrices A behaves asymptotically like $\kappa_2(A) = \mathcal{O}\left(h^{-2}\right)$ or even worse with respect to h.

Consider a symmetric positive definite matrix A with $\kappa_2(A) = \mathcal{O}\left(h^{-2}\right)$ and the CG method. Then, if follows from Remark 7.8 that the number of iterations for achieving a prescribed reduction of the error behaves like $\sqrt{\kappa_2(A)} = \mathcal{O}\left(h^{-1}\right)$. Thus, refining the mesh once, i.e., $h \to h/2$, leads to the expectation that the number of iterations for the same reduction of the error on the fine grid is twice as large as the number on the coarser grid. This non-optimal behavior can be observed in practice, compare also the corresponding problem from the exercises.

An optimal solver has to satisfy two requirements:

- the number of iterations for reducing the error with a certain factor is independent of h, i.e., the spectral norm of the iteration matrix should be smaller than a number $\rho_0 < 1$ with ρ_0 independent of h,
- the cost per iteration scales linearly with the number of unknowns.

The second requirement cannot be improved since for solving a linear system of equations, each unknown has to be touched at least once.

A class of solvers that satisfy these requirements, at least for certain problems, are (geometric) multigrid methods. For such methods, one needs a hierarchy of grids, starting with a coarsest one, then a next finer one, and so on until the finest grid, which is the grid where the solution should be computed. On each grid, one defines an appropriate linear system of equations and between the grids one has to define appropriate transfer operators. Starting on the finest grid, one applies on each grid, but the coarsest one, a simple iterative scheme, e.g., one of the classical iterative schemes from Chapter 3. On the coarsest grid, where the dimension of the linear system of equations is much smaller than on the finest grid, one solves this system, e.g., with a direct solver.

In practice, however, one often has only one grid, in particular if the domain is complicated, and this grid is already fine. Thus, one cannot build a grid hierarchy. For such situations, so-called algebraic multigrid methods (AMG) has been proposed. These method are building a hierarchy of matrices that starts with the matrix on the given grid. Then, matrices with smaller and smaller dimension, and corresponding right-hand sides, are constructed. Such constructions are based on the sparsity pattern and the size of the entries. There are different approaches that lead to different kinds of AMG methods. □

Remark 10.4. Software. There are many packages that provide iterative solvers. A quite popular one, which provides also many other tools, is PETSc. □