



Prof. Dr. U. Rüdte
Benjamin Mann

Winter Term
2024/2025

Algorithms of Numerical Linear Algebra Assignment 7

Exercise 1 (*Conjugate Gradients*)

2P.

Make sure to follow the requirements for programming tasks stated on the information sheet!

In this final assignment, we will implement and analyze krylov subspace methods. As a warm-up, we begin with the most straightforward one of these methods.

Implement the Python3 function `(x, [r_b]) = cg(A, b, tol)` which applies the conjugate gradient algorithm to compute an approximate solution for the linear system $Ax = b$ for input values $A \in \mathbb{R}^{m \times m}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^m$. The iteration should stop after at most m iterations. It should also stop once the relative norm $\|r\|/\|b\|$ of the residual $r = Ax - b$ is less than the given tolerance `tol`. The return values should be the approximate solution x and a list containing $\|r_k\|/\|b\|$ for all iterations k , including the initial value (i.e. `len(r_b)` should be number of iterations + 1).

Exercise 2 (*Generalized Minimal Residuals with Preconditioning*)

12P.

Make sure to follow the requirements for programming tasks stated on the information sheet!

Since this is a bit more involved than the CG algorithm, we will do it step by step:

- (a) (*Arnoldi*): Implement the Python3 function `(h, q) = arnoldi_n(A, Q, P)` which runs the n -th step of the Arnoldi iteration. The input values are the system matrix $A \in \mathbb{R}^{m \times m}$, a matrix $Q \in \mathbb{R}^{m \times n}$ containing the first n orthogonal basis vectors q_1, \dots, q_n of the krylov space and $P \in \mathbb{R}^{m \times m}$. For the moment, let's ignore P . The output should be the n -th column of the Hessenberg matrix \tilde{H}_n and the $(n+1)$ -th basis vector q_{n+1} , computed as in Algorithm 33.1 in [1]. The algorithm should not alter the input matrices in any way e.g. by adding columns!
- (b) (*GMRES*): Implement the Python3 function `(x, [r_b]) = gmres(A, b, P, tol)` which runs the GMRES algorithm. As in the first exercise, we are interested in an approximate solution to $Ax = b$. The description of input and return values as well as stopping criteria are identical to the ones used for the CG algorithm (once again, we ignore P for the moment). When taking a look at Algorithm 35.1 in [1], you will realize that in each iteration we require both an Arnoldi step and a least squares fit. For the Arnoldi part we simply employ the function we just implemented. The minimization problem however, needs some special care: Do not simply use a blackbox LSQ solver! Since the problem has upper Hessenberg form, it is more efficient to do this 'by hand'. For the sake of simplicity, you are allowed to use `numpy.linalg.qr` and `scipy.linalg.solve_triangular`, though.
- (c) (*Preconditioning*): We now come back to the parameter P . As previously stated, it is a matrix with the same dimensions as A . In fact, it is an approximation to A where the inverse

is relatively cheap to evaluate. Instead of solving for $Ax = b$ we now attempt to solve $P^{-1}Ax = P^{-1}b$. Of course, we still don't compute P^{-1} explicitly, but solve a corresponding linear system instead. Incorporate the necessary adaptations in both `arnoldi_n` and `gmres`! You can assume that the preconditioner P is always upper triangular, i.e., the corresponding systems can be solved using `scipy.linalg.solve_triangular`.

- (d) (*GMRES with Givens rotations*): Now implement a more efficient version of the GMRES algorithm in the function `(x, [r_b]) = gmres_givens(A, b, P, tol)`. The function should yield the same result as the one from the previous task. However, instead of using the built-in QR-factorization, implement an algorithm, which only introduces **one additional Givens rotation in each iteration**.

Exercise 3 (Discussion)

6P.

It is time to put our algorithms to the test! Before we begin, make yourself familiar with the given script `krylov_comparison.py`.

- What preconditioners are implemented here? Which of them is applied in our benchmark?
- What algorithm is employed in the function `magic(A, b, tol)`? Why is this implementation bad?
- Run the script and attach the resulting plots to your pdf-submission. What can be said about the convergence of the four methods? (What impact do the properties of the system matrix have? How do you explain these results? Are they as expected? ...)

References

- [1] L.N. Trefethen and D. Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.