CHAPTER I

Introduction

In this lecture, we discuss theory, numerics and application of advanced problems in linear algebra:

(II) matrix equations (example: solve $AX + XB = C$),

(III) matrix functions: compute $f(A)$ or $f(A)b$, where $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$,

(IV) randomized algorithms.

The main focus is on problems defined by real matrices/vectors. In most chapters, we have to make the distinction between problems defined by

- dense matrices of small /moderate dimensions and
- large, sparse matrices, e.g. $A \in \mathbb{C}^{n\times n}$, $n > 10^4$ or greater, but only $\mathcal{O}(n)$ nonzero entries, often from PDEs.

We first have to review two important standard problems in numerical linear algebra, namely solving linear systems of equations and eigenvalue problems.

I.1 Linear systems of equations

We consider the linear system

$$Ax = b, \tag{I.1}$$

with $A \in \mathbb{C}^{n\times n}\,(\mathbb{R}^{n\times n})$, $\quad b \in \mathbb{C}^n\,(\mathbb{R}^n)$. The linear system (I.1) admits a unique solution, if and only if

- there exists an inverse A^{-1}
- $\det(A) \neq 0$
- no eigenvalues/ singular values are equal to zero
- ...

Numerical methods for small and dense $A \in \mathbb{C}^{n\times n}$

Gaussian Elimination (LU-factorization):

We decompose A such that

$$A = LU, \quad L = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{bmatrix}.$$

We obtain, that

$$\text{(I.1)} \quad \Leftrightarrow \quad LUx = b \quad \Leftrightarrow \quad x = U^{-1}(L^{-1}b).$$

Hence, we solve (I.1) in two steps:

1. Solve $Ly = b$ via backward substitution.
2. Solve $Ux = y$ via backward substitution.

This procedure is numerically more robust with pivoting $PAQ = LU$, where P, $Q \in \mathbb{C}^{n,n}$ are permutation matrices. This method has a complexity of $\mathcal{O}(n^3)$ and is, therefore, only feasible for small (moderate) dimensions.

QR-decomposition:

We decompose A into a product of Q and R where Q is an orthogonal matrix and R is an upper triangular matrix leading to the so-called Gram-Schmidt or the modified Gram-Schmidt algorithm. Numerically this can be done either with Givens rotations or with Householder transformations.

Methods for large and sparse $A \in \mathbb{C}^{n\times n}$

Storing and computing dense LU-factors is infeasible for large dimensions n ($\mathcal{O}(n^2)$ memory, $\mathcal{O}(n^3)$ flops). One possibility are *sparse direct solvers*, i.e. find permutation matrices P and Q, such that $PAQ = LU$ has sparse LU-factors (cheap forward/ backward substitution and $\mathcal{O}(n)$ memory).

Example: We consider the LU-factorization of the following matrix

$$A = \begin{bmatrix} * & \cdots & * \\ \vdots & \ddots & \\ * & & * \end{bmatrix} = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \begin{bmatrix} \ddots & & \\ & \ddots & \\ & & \ddots \end{bmatrix}.$$

With the help of permutation matrices P and Q, we can factorize

$$PAQ = \begin{bmatrix} * & & * \\ & \ddots & \vdots \\ * & \cdots & * \end{bmatrix} = \begin{bmatrix} * & & \\ \vdots & \ddots & \\ * & & * \end{bmatrix} \begin{bmatrix} * & \cdots & * \\ & \ddots & \\ & & * \end{bmatrix}.$$

Algorithm 1 Arnoldi method

Input: $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$
Output: Orthonormal basis Q_k of (I.2)
1: Set $q_1 = \frac{b}{\|b\|}$ and $Q_q := [q_1]$.
2: **for** $j = 1, 2, \ldots$ **do**
3: Set $z = Aq_j$.
4: Set $w = z - Q_j(Q_j^{\mathrm{H}} z)$.
5: Set $q_{j+1} = \frac{w}{\|w\|}$.
6: Set $Q_{j+1} = [Q_j, q_{j+1}]$.
7: **end for**

Finding such P and Q and still ensuring numerical robustness is difficult and based e.g. on graph theory.

In MATLAB, sparse-direct solvers are found in the "\"-command: $x = A\backslash b$ or $\mathrm{lu}(A)$-routine. (Never use $\mathrm{inv}(A)$!)

Iterative methods

Often an approximation $\widehat{x} \approx x$ is sufficient. Hence, we generate a sequence $x_1, x_2, \ldots, x_k$ by an iteration, such that

$$\lim_{k\to\infty} x_k = x = A^{-1}b$$

and each x_k, $k \geqslant 1$ is generated efficiently (only $\mathcal{O}(n)$ computations). Of course, we want $x_k \approx x$ for $k \ll n$.

Idea: Search approximated solution in a low-dimensional subspace $\mathcal{Q}_k \subset \mathbb{C}^n$, $\dim(\mathcal{Q}_k) = k$. Let $\mathcal{Q}_k$ be given as $\mathrm{range}(Q_k) = \mathcal{Q}_k$ for a matrix $Q_k \in \mathbb{C}^{n\times k}$.

A good choice of the subspace is the Krylow-subspace

$$\mathcal{Q}_k = \mathcal{K}_k(A, b) = \mathrm{span}\{b, Ab, \ldots, A^{k-1}b\}. \tag{I.2}$$

It holds for $z \in \mathcal{K}_k(A, b)$, that $z = p(A)b$ for a polynomial of degree $k-1$ $p \in \Pi_{k-1}$. An orthonormal basis of $\mathcal{K}_k(A, b)$ can be constructed with the *Arnoldi process* presented in Algorithm 1.

The Arnoldi process requires matrix-vector products $z = Aq$. These are cheap for sparse A and therefore feasible for large dimensions.

We find an approximation $x_k \in x_0 + \mathcal{Q}_k$ by two common ways:

- Galerkin-approach:
  Impose $r = b - Ax_k \perp \mathrm{range}(Q_k) \quad \Leftrightarrow \quad (Q_k^{\mathrm{H}} A Q_k) y_k = Q_k^{\mathrm{H}} b.$

  We have to solve a k-dimensional system $\Rightarrow$ low costs.

- Minimize the residual:

$$\min_{x_k \in \mathrm{range}(Q_k)} \|b - Ax_k\|$$

  in some norm. If x_k is not good enough, we expand Q_k.

There are many Krylov-subspace methods for linear systems. (Simplification for $A = A^{\mathrm{H}}$: Arnoldi $\rightsquigarrow$ Lanczos)

In practice: Convergence acceleration by *preconditioning*:

$$Ax = b \quad \Leftrightarrow \quad P^{-1}Ax = P^{-1}b$$

for easily invertible $P \in \mathbb{C}^{n,n}$ and $P^{-1}A$ "nicer" than A ($\rightsquigarrow$ Literature NLA I).

Another very important building block is the numerical solution of eigenvalue problems.

I.2 Eigenvalue problems (EVP)

For a matrix $A \in \mathbb{C}^{n,n}$ we want to find the eigenvectors $0 \neq x \in \mathbb{C}^n$ and the eigenvalues $\lambda \in \mathbb{C}$ such that

$$Ax = \lambda x.$$

The set of eigenvalues $\Lambda(A) = \{\lambda_1, \ldots, \lambda_n\}$ is called the *spectrum of* A.

Small, dense problems:

Computing the Jordan-Normal-Form (JNF)

$$X^{-1}AX = J = \mathrm{diag}(J_{s_1}(\lambda_1), \ldots, J_{s_k}(\lambda_k)), \quad J_{s_j}(\lambda_j) := \begin{bmatrix} \lambda_j & 1 & \\ & \ddots & 1 \\ & & \lambda_j \end{bmatrix}$$

to several eigenvalues and eigenvectors is numerically infeasible, unstable (NLA I).

Theorem I.1 (Schur)**:** For all $A \in \mathbb{C}^{n\times n}$ exists a unitary matrix $Q \in \mathbb{C}^{n,n}$ ($Q^{\mathrm{H}}Q = I$), such that

$$Q^{\mathrm{H}}AQ = R = \underbrace{\begin{bmatrix} \lambda_1 & & * \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}}_{\text{Schur form of } A}$$

with $\lambda_i \in \Lambda(A)$ in arbitrary order.

The Schur form can be numerically stable computed in $\mathcal{O}(n^3)$ (NLA I) by the Francis-QR-algorithm. It is this basis for dense eigenvalue computations. In MATLAB we use $[\mathrm{Q}, \mathrm{R}] = \mathrm{schur(A)}$. Additionally, the routine $\mathrm{eigs(A)}$ uses the Schur form. In general, the columns of Q are no eigenvectors of A, but $Q_k = Q(:, 1 : k)$ spans an A-*invariant subspace* for all k:

$$AQ_k = Q_k R_k, \quad \text{for a matrix } R_k \in \mathbb{C}^{k\times k} \text{ with } \Lambda(R_k) \subseteq \Lambda(A).$$

But because of the $\mathcal{O}(n^3)$ complexity and $\mathcal{O}(n^2)$ memory, the Schur form is infeasible for large and sparse matrices A.

Eigenvalue problems defined by large and sparse matrices A can again be treaded with the Arnoldi-process and projections on the Krylov-subspace $\mathcal{K}_k(A, b) = \mathrm{range}(Q_k)$. We obtain the approximated eigenpair $x_k = Q_k y_k \approx x,\ \mu \approx \lambda$ by using the Galerkin-condition on the residual of the eigenvalue problem:

$$r_k = Ax_k - \mu\, x_k \perp \mathrm{range}(Q_k) \quad \Leftrightarrow \quad Q_k^{\mathrm{H}} A Q_k y_k = \mu\, y_k,$$

which means (μ, y_k) are the eigenpairs of the $k \times k$-dimensional eigenvalue problem for $Q_k^{\mathrm{H}} A Q_k$. This small eigenvalue problem is solvable by the Francis-QR-method. This is the basis of the $\mathrm{eigs(A)}$ routine in MATLAB for computing a few ($\ll n$) eigenpairs of A.

Summary: Solving linear systems and eigenvalue problems is for small or large and sparse matrices A no problem!