Lecture 27

Preconditioned Conjugate Gradient

Learning Outcomes

- Gaining insights into the role and importance of the condition number $\kappa_2(A)$ of a matrix A in the context of the steepest descent and conjugate gradient methods, and understanding how a large condition number can lead to inefficiencies in minimizing along the gradient due to zigzag behavior.
$$\kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$
- Acquiring knowledge on the concept and application of preconditioning to transform the problem $Ax = b$ to a more suitable form for numerical solving methods, ensuring faster convergence by reducing the condition number of the problem. Specifically, understanding the transformation:
$$Ax = b \xrightarrow{\text{transform to}} AMM^{-1}x = b$$
where AM has a better condition number than A.
- Learning various techniques and strategies for choosing an effective preconditioner M, balancing between computational cost and approximation accuracy, and understanding how different preconditioners like diagonal inverses, incomplete factorizations, and linear solvers can be applied depending on the properties of the matrix A and the specific requirements of the problem at hand.

Preconditioning of CG method

The condition number of a matrix is defined as follows,

$$\kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Also note that for the special type of matrix, symmetric positive definite matrix, considered in the steepest descent and conjugate gradient methods, the eigenvalues are always real and the corresponding eigenvectors are always orthogonal. Moreover, eigenvalues corresponds to how much the vector is stretched in the direction of the eigenvectors. This means that the condition number represents how elongated the contour graph of the matrix is.

Generally minimising along the gradient is not very efficient when the condition number of A is large. This is because the gradient is very weak along the eigenvector that corresponds to the smallest eigenvalue resulting in the zigzag behaviour we discussed in the previous chapter.

Thus, in most cases, preconditioning is necessary to ensure fast convergence of the steepest descent and conjugate gradient methods. Preconditioning, which is the application of a transformation, called the preconditioner, is typically related to reducing a condition number of the problem. It conditions a given problem into a form that is more suitable for numerical solving methods. In particular,

$$Ax = b \xrightarrow{\text{transform to}} AMM^{-1}x = b$$

such that AM has a better condition number than A.

There are many techniques to generate M that can be chosen depending on the properties of the matrix A. In the following, the discussion will be restricted to Krylov subspace method. Since for every regular $M \in \mathbb{R}^{n\times n}$

$$b = Ax = AMM^{-1}x = (AM)(M^{-1}x) =: AM\tilde{x},$$

an appropriate choice of M may yield a system with $\kappa(AM) < \kappa(A)$. But a problem emerges: Even if $M \in \mathbb{R}^{n\times n}$ is positive definite (which we assume from now on) there is no guarantee that AM is positive definite. To solve this problem let us recall what we effectively need for **CG**. The symmetry of A is equivalent to

$$\langle Ax, y\rangle = \langle x, Ay\rangle \quad \forall x, y \in \mathbb{R}^{n\times n}.$$

This means that A considered as a linear operator on $\mathbb{R}^n$ is self-adjoint with respect to the standard inner product. Consider now the bilinear form

$$\langle\cdot,\cdot\rangle_M : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad \langle x, y\rangle_M := \langle x, My\rangle$$

which, thanks to the positivity of M, is an inner product (recall the arguments around (1.4) on page 30, and recall also the notation $\|\cdot\|_M$ for the associated norm). It turns out that $\tilde{A} = AM$ is self-adjoint with respect to $\langle\cdot,\cdot\rangle_M$: For all $x, y \in \mathbb{R}^n$

$$\langle AMx, y\rangle_M = \langle AMx, My\rangle = x^T M^T A^T My \underbrace{=}_{A,M \text{ symmetric}} x^T MAMy = \langle x, M(AM)y\rangle = \langle x, AMy\rangle_M.$$

Hence, we may just replace the standard inner product by $\langle\cdot,\cdot\rangle_M$ in algorithm **CG** and thus obtain a method for computing the solution to $b = AM\tilde{x}$. The essential lines read as follows where we inserted the definition of $\langle\cdot,\cdot\rangle_M$:

```
1: $d^{(0)} := r^{(0)} := b - AM\tilde{x}^{(0)}$
2: $l^{(0)} := \langle r^{(0)}, Mr^{(0)}\rangle$
3: if $l^{(0)} \le \varepsilon_r$ then
4:     return $\tilde{x}^{(0)}$
5: else
6:     for $k = 1, 2, \ldots$ do
7:         $h^{(k-1)} := AMd^{(k-1)}$
8:         $\alpha^{(k-1)} := l^{(k-1)}/\langle d^{(k-1)}, Mh^{(k-1)}\rangle \left(= \langle r^{(0)}, Mr^{(0)}\rangle/\langle Md^{(k-1)}, AMd^{(k-1)}\rangle\right)$
9:         $\tilde{x}^{(k)} := \tilde{x}^{(k-1)} + \alpha^{(k-1)}d^{(k-1)}$
10:        $r^{(k)} := r^{(k-1)} - \alpha^{(k-1)}h^{(k-1)} \left(= r^{(k-1)} - \alpha^{(k-1)}AMd^{(k-1)}\right)$
```

```
11:         $l^{(k)} := \langle r^{(k)}, Mr^{(k)} \rangle$
12:         **if** $l^{(k)} \le \varepsilon_r$ **then**
13:             return $\tilde{x}^{(k)}$
14:         **end if**
15:         $\beta^{(k)} := l^{(k)}/l^{(k-1)} \Big( = \langle r^{(k)}, Mr^{(k)} \rangle / \langle r^{(k-1)}, Mr^{(k-1)} \rangle \Big)$
16:         $d^{(k)} := r^{(k)} + \beta^{(k)} d^{(k-1)}$
17:     **end for**
18: **end if**
```

As we are interested in x rather than $\tilde{x}$ we may multiply line 9 with M from the left to obtain $x^{(k)} := x^{(k-1)} + \alpha^{(k-1)} M d^{(k-1)}$. Doing the same with line 16 we see that we may introduce the vectors $q^{(k)} := Md^{(k)}$ to replace the vectors $d^{(k)}$. The preconditioner M only acts on the residual, and it is possible to formulate and implement the algorithm such that only one matrix-vector multiplication with M per step is required. Introducing $s^{(k)} := Mr^{(k)}$ we arrive at:

Algorithm 14 PCG (preconditioned conjugate gradient method)

```
input:  $A, M \in \mathbb{R}^{n \times n}$ positive definite, $b, x^{(0)} \in \mathbb{R}^n$, $\varepsilon_r > 0$.
output:  $x \in \mathbb{R}^n$ with $\|Ax - b\|_M \le \varepsilon_r$.
1: $r^{(0)} := b - Ax^{(0)}$
2: $q^{(0)} := s^{(0)} := Mr^{(0)}$
3: $l^{(0)} := \langle r^{(0)}, s^{(0)} \rangle$
4: if $l^{(0)} \le \varepsilon_r$ then
5:     return $x^{(0)}$
6: else
7:     for $k = 1, 2, \ldots$ do
8:         $h^{(k-1)} := Aq^{(k-1)}$
9:         $\alpha^{(k-1)} := l^{(k-1)} / \langle q^{(k-1)}, h^{(k-1)} \rangle$
10:        $x^{(k)} := x^{(k-1)} + \alpha^{(k-1)} q^{(k-1)}$
11:        $r^{(k)} := r^{(k-1)} - \alpha^{(k-1)} h^{(k-1)}$
12:        $s^{(k)} := Mr^{(k)}$
13:        $l^{(k)} := \langle r^{(k)}, s^{(k)} \rangle$
14:        if $l^{(k)} \le \varepsilon_r$ then
15:            return $x^{(k)}$
16:        end if
17:        $\beta^{(k)} := l^{(k)}/l^{(k-1)}$
18:        $q^{(k)} := s^{(k)} + \beta^{(k)} q^{(k-1)}$
19:    end for
20: end if
```

The big question now: How to choose M? One will want that the computation of $s^{(k)} = Mr^{(k)}$ is cheap. But a good approximation of A^{-1} or, at least, keeping $\kappa(AM)$ low, is desired as well. To find a good trade-off depends strongly on the actual problem and the used computing hardware so, usually, requires some testing and trying. Some ideas to define preconditioners are stated below. Observe that in algorithm **PCG** we need the action of M on a vector which may be cheaper to compute than building the matrix M and employing the usual matrix-vector product algorithm.

- Very simple but occasionally highly effective: Choose $M := D^{-1}$ where D is the diagonal of A.

- Computing the LU factorisation or, since A is symmetric, the Cholesky factorisation $A = LL^T$ usually leads to a fill-in, i.e., zero-entries in a sparse matrix A become non-zero in the triangular matrices of the factorisation. But often those entries are 'small' compared to the other entries. So one may neglect the fill-in and compute an incomplete factorisation $A \approx \tilde{A} = \tilde{L}\tilde{L}^T$. The preconditioner then is $M := \left(\tilde{L}\tilde{L}^T\right)^{-1}$.
- The linear solvers can serve as preconditioners as well. For instance, the action of M on a vector may correspond to a few steps of the Jacobi method.
- Modern general preconditioners also include (possibly algebraic) multigrid methods (not discussed in this course).

Lecture 28

Power Iteration

Learning Outcomes

- Understanding of the eigenvalue problem (EVP) and the realization that finding eigenvalues and eigenvectors are equivalent problems, especially in the context of engineering where matrices are often used. Awareness of Abel's theorem and its implications on the solutions of the characteristic polynomial for $n \geq 5$.
- Acquisition of knowledge on the power iteration method, a simple iterative algorithm used to find the vector corresponding to the largest in magnitude eigenvalue of a diagonalizable matrix A. Familiarity with the practical implementation of the algorithm, including normalization steps to prevent overflows and underflows.
- Insight into the Hermitian eigenvalue problem and the properties of Hermitian matrices. Recognition of the Rayleigh quotient as a natural way to estimate an eigenvalue from an eigenvector, and the ability to implement an algorithm to find both the eigenvector and the corresponding eigenvalue.

Power Iteration Method

There are many engineering examples where you might want to find the eigenvalues and the eigenvectors of a matrix $A \in \mathbb{C}^{n\times n}$. It should also be noted that finding eigenvalues and eigenvectors are equivalent problems. This is known as the eigenvalue problem (EVP).

Finding the eigenvalue of A is equivalent to finding the roots of the characteristic polynomial. However, there is the following result of Abel (1824):

Theorem 28.1. *If $n \geq 5$ then there is a polynomial of degree n with rational coefficients that has a real root which cannot be expressed by only using rational numbers, $+$, $-$, $*$, $/$, $(\cdot)^{\frac{1}{k}}$ with $k \in \mathbb{N}$.*

Consequently, algorithms for solving EVPs will be iterative. This chapter will provide an introduction on some of the methods that are used to solve the EVP. The first method we will learn about is called the power iteration method which is a simple method that yields a vector that corresponds to the largest in magnitude eigenvalue.

Assume that the matrix A is diagonalisable. The aim then is to find its eigenvalues and eigenvectors of that matrix,

$$Ax_i = \lambda_i x_i$$

Now ordering the eigenvalues such that,

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \cdots \geq |\lambda_n|$$

Since A is diagonalisable we have,

$$A = X\Lambda X^{-1}$$

where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and $X = (x_1|x_2|\ldots|x_n)$.

Idea: iterate $z^{(k)} = Az^{(k-1)}$ and hope that the iterates align with the eigenspace of λ_1.

Algorithm 15 PI (power iteration)

```
input: $A \in \mathbb{R}^{n\times n}$ diagonalisable, $z^{(0)} \in \mathbb{R}^n$.
output: $z^{(k)} \in \mathbb{R}^n$ approximation to $x_1$.
1: for $k = 1, 2, \ldots$ do
2:     $z^{(k)} := Az^{(k-1)}$
3: end for
```

Example: consider

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}, \quad z^{(0)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Then

$$z^{(1)} = \begin{pmatrix} 1 \\ 2^{-1} \end{pmatrix}, z^{(2)} = \begin{pmatrix} 1 \\ 2^{-2} \end{pmatrix}, \ldots, z^{(k)} = \begin{pmatrix} 1 \\ 2^{-k} \end{pmatrix} \to \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

which is an eigenvector to $\lambda_1 = 1$.

So we have a vector $z^{(0)}$ which can be written as a linear combination of the eigenvectors x_i's,

$$\begin{aligned} z^{(0)} &= Xy \\ &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ &= \sum_{i=1}^{n} c_ix_i \end{aligned}$$

where y is vector of coefficients.

Iterating once yields,

$$\begin{aligned} z^{(1)} &= Az^{(0)} \\ &= A\sum_{i=1}^{n} c_ix_i = \sum_{i=1}^{n} c_iAx_i = \sum_{i=1}^{n} c_i\lambda_ix_i \end{aligned}$$

Iterating k times yields,

$$\begin{aligned} z^{(k)} &= A^k z^{(0)} \\ &= A^k \sum_{i=1}^{n} c_i x_i = \sum_{i=1}^{n} c_i A^k x_i = \sum_{i=1}^{n} c_i \lambda^k x_i \\ &= \lambda_1^k \left(c_1 x_1 + \sum_{i=2}^{n} c_i \left(\frac{\lambda_i}{\lambda_1} \right)^k x_i \right) \end{aligned}$$

Thus as $k \to \infty$, the second term in the above equation will go to 0, i.e,

$$z^{(k)} \approx \lambda_1^k c_1 x_1$$

Finally to prevent overflows and underflows, the vector $z^{(k)}$ needs to be normalised after each iteration yielding the following practical algorithm.

Algorithm 16 PI (power iteration normalised)

```
input:  $A \in \mathbb{R}^{n \times n}$ symmetric, $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 \neq 0$ and $\langle z^{(0)}, x_1 \rangle \neq 0$.
output:  $z^{(k)} \in \mathbb{R}^n \in \mathbb{R}$ approximation to $x_1$.
1: $k = 1$
2: repeat
3:     $w^{(k)} := A z^{(k-1)}$
4:     $z^{(k)} := w^{(k)} / \|w^{(k)}\|_2$
5:     $k := k + 1$
6: until Stopping criterion fulfilled
```

For computing an eigenvector, a possible stopping criterion is $\|z^{(k)} - z^{(k-1)}\|_2 \leq \varepsilon_r$ or $\|z^{(k)} + z^{(k-1)}\|_2 \leq \varepsilon_r$ where we have to distinguish the sign because λ_1 may be negative. In fact, for

$$A = \begin{pmatrix} -1 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}, \quad z^{(0)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

we obtain the iterates

$$z^{(1)} = \begin{pmatrix} -1 \\ 2^{-1} \end{pmatrix}, z^{(2)} = \begin{pmatrix} 1 \\ 2^{-2} \end{pmatrix}, z^{(3)} = \begin{pmatrix} -1 \\ 2^{-3} \end{pmatrix} \ldots, z^{(k)} = \begin{pmatrix} (-1)^k \\ 2^{-k} \end{pmatrix}$$

which do not converge, but the vectors $(-1)^k z^{(k)} = (1, (-1)^k 2^{-k})^T$ converge to $(1, 0)^T$.

So this is a practical algorithm that gives us the eigenvector, but it does not give us the eigenvalue! The problem of finding an eigenvalue from an eigenvector can then be described as follows. Given an eigenvector $x \in \mathbb{C}^n$, find the
ambda $\in \mathbb{C}$ which minimises

$$\|\lambda x - Ax\|_2$$

Since x is an eigenvector the minimum of 0 is attained for the corresponding eigenvalue which can be shown to be given by,

$$\lambda = \frac{\langle x, Ax \rangle}{\langle x, x \rangle}$$

This result shows that the following definition provides a natural way to estimate an eigenvalue from an eigenvector.

Definition 28.1. *The Rayleigh quotient of a matrix $A \in \mathbb{C}^{n\times n}$ is defined as,*

$$r_A(x) := \frac{\langle x, Ax\rangle}{\langle x, x\rangle}$$

for all $x \in \mathbb{C}^n$

Using this definition, we get that

$$Ax = r_A(x)x$$

for all eigenvectors x of A.

The following algorithm can then be readily used.

Algorithm 17 PI (power iteration and eigenvalue)

input: $A \in \mathbb{R}^{n\times n}$ symmetric, $z^{(0)} \in \mathbb{R}^n$ with $\|z^{(0)}\|_2 \neq 0$ and $\langle z^{(0)}, x_1\rangle \neq 0$.
output: $z^{(k)} \in \mathbb{R}^n$ and $\lambda^{(k)} \in \mathbb{R}$ approximation to x_1 and λ_1.

```
1: $k = 1$
2: repeat
3:     $w^{(k)} := Az^{(k-1)}$
4:     $z^{(k)} := w^{(k)}/\|w^{(k)}\|_2$
5:     $k := k + 1$
6: until Stopping criterion fulfilled
7: $\lambda := r_A(z)$
```

Hermitian eigenvalue problem

From now on we will restrict our attention to Hermitian real matrices $A \in \mathbb{C}^{n\times n}$, where the eigenvalues are real and denoted by $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$. Associated normalised eigenvectors are denoted by $x_1, \ldots, x_n \in \mathbb{R}^n$, which will always denote an orthonormal system of eigenvectors of A.