Numerical Linear Algebra

Exam questions

Concerned people

Skoltech, 2020

PLEASE, PUT YOUR SOLUTIONS NOT IN main.tex, BUT IN №.tex. ALSO TRY TO SOLVE WARNINGS AND ERRORS :)

These tickets are made up by students and may contain semantic errors – be careful.

Note for writers use
\paper{question number}{question description},
to automatically add a ticket to the table of contents, select a new page for it, and place all tickets in the same font.

Contents

1. Representation of numbers in computer. Fixed and floating point arithmetics. Loss of significance.

Information in computer is stored in bits. Each bit saves either value of zero or one. There are several ways to represent a real number in computer, we will talk about *fixed-point* and *floating-point* arithmetics.

- *Fixed-point representation*, otherwise called **Qm.n** format, takes the fixed amount of bits for magnitude (**integer**) and **fractional** parts, to be more precise, m and n bits in each case (that is where the name comes from).
  One bit in the beginning is reserved for sign of a number, so in total we need *m + n + 1* bits to represent number this way. Hence, the range of numbers we can represent this way is $[-(2^m), 2^m - 2^{-n}]$ with *resolution* (fractional part accuracy) of 2^{-n}.
  Because of its fixed range and resolution, this kind of representation possesses **absolute** accuracy (you can represent any number in given range and resolution). Also, you should be aware of overflow when sum of two numbers is out of given range.

- *Floating-point representation.* This format uses the representation of number in following way:

  $$num = sign \times significant \times base^{exponent}$$

  Sign is either zero or one (as in fixed-point case), significand and base are positive integers, and exponent is either positive or negative depending on whether the number had fractional part.
  Note that in this case the accuracy is **relative** (varies depending on the scale of a number) due to its exponential nature. This can cause the problem called *loss of significance*: the components that does not fit the significand bits are truncated. Be aware while substracting numbers with large fractinal part.
  Example: *float32* in C: 1 bit for sign, 8 bits for exponent, 23 bits for significand.

2. Vector norms. Forward and backward stability. Disks in different norms. First norm and compressed sensing.

Vector norms

In a fixed basis of size n a vector can be represented as a 1D array with n numbers.

Vectors typically provide an (approximate) description of an object. And we should know **how accurate** the approximation is. The norms serve this purpose and used to compute the accuracy. The smaller the distance between vectors the more accurate the approximation.

Norm is a **qualitative measure of smallness of a vector** and is typically denoted as $\|x\|$.

The norm should satisfy certain properties:

- $\|\alpha x\| = |\alpha| \|x\|$
- $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)
- If $\|x\| = 0$ then $x = 0$

The distance between two vectors is then defined as

$$d(x, y) = \|x - y\|.$$

There are several norms, the most common class of them is p-norms:

$$\|x\|_p = \Big(\sum_{i=1}^{n} |x_i|^p\Big)^{1/p}.$$

There are three very important special cases:

1. The most well-known and widely used norm is L_2 norm (**Euclidean norm**):

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2},$$

which corresponds to the distance in our real life. If the vectors have complex elements, we use their **modulus**.

2. Infinity norm, or **Chebyshev norm** is defined as the maximal element:

$$\|x\|_\infty = \max_i |x_i|$$

3. L_1 norm (or **Manhattan distance**) which is defined as the sum of modules of the elements of x:

$$\|x\|_1 = \sum_i |x_i|$$

Equivalence of the norms

All norms are equivalent in the sense that

$$C_1 \|x\|_* \leq \|x\|_{**} \leq C_2 \|x\|_*$$

for some positive constants $C_1(n), C_2(n)$, $x \in \mathrm{R}^n$ for any pairs of norms $\|\cdot\|_*$ and $\|\cdot\|_{**}$. The equivalence of the norms basically means that if the vector is small in one norm, it is small in another norm.

Some examples:

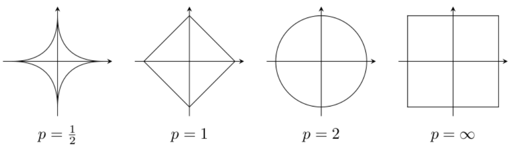
$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$
$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$
$\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$
In one line:
$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2 \leq n\|x\|_\infty$

Unit disk

A unit disk is a set of point such that $\|x\| \leq 1$

For the euclidean norm a unit disk is a usual disk

For other norms unit disks look very different



Note: $p = \frac{1}{2}$ is not a norm, because the triangle inequality is not satisfied.

First norm and compressed sensing

L_1 norm plays an important role in **compressed sensing**.

Compressed sensing is a signal processing technique for efficiently acquiring and reconstructing a signal, by ***finding solutions to underdetermined linear systems***. This is based on the principle that, through optimization, the sparsity of a signal can be exploited to recover it from far fewer samples than required by the Nyquist–Shannon sampling theorem (the sampling rate should be twice higher then signal's highest frequency).

In other words, if we know that the signal is sparse, we can use fewer samples or a lower sample rate. And L1 norm helps in this.

The simplest formulation of the considered problem is as follows:

- You have some observations f
- You have a linear model $Ax = f$, where A is an $n \times m$ matrix, A is **known**. (x is signal, which we want recover)
- The number of equations, n, is less than the number of unknowns, m. Because we can observe only small part of a signal.
- The goal is to find the solution (recover signal x)

The solution of underdetermined linear systems is obviously non-unique, so a natural approach is to find the solution that is minimal in the certain sense:

$$\|x\| \to \min_x$$
$$\text{subject to } Ax = f$$

- Typical choice of $\|x\| = \|x\|_2$ leads to the **linear least squares problem** (and has been used for ages).
- The choice $\|x\| = \|x\|_1$ leads to the **compressed sensing**
- It typically yields the **sparsest solution**

Forward and backward stability

- Let x be an object (for example, a vector)
- Let $f(x)$ be the function (functional) you want to evaluate

You also have a **numerical algorithm** alg(x) that actually computes **approximation** to $f(x)$. The algorithm is called **forward stable**, if (The error is very small)

$$\|alg(x) - f(x)\| \leq \varepsilon$$

The algorithm is called **backward stable**, if for any x there is a close vector $x + \delta x$ such that

$$alg(x) = f(x + \delta x)$$

and $\|\delta x\|$ is small.

3. Matrix-matrix multiplication. Computer memory hierarchy and blocking. Standard libraries for linear algebra. Strassen algorithm (no need to remember exact formulas) and its complexity (with derivation).

Matrix-by-matrix product

Definition A product of an $n \times k$ matrix A and a $k \times m$ matrix B is a $n \times m$ matrix C with the elements

$$c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}, \quad i = 1, \dots, n, \quad j = 1, \dots, m$$

For $m = k = n$ complexity of a naïve algorithm is $2n^3 - n^2 = \mathcal{O}(n^3)$.

- All the dense NLA algorithms are reduced to a sequence of MM.
- Efficient implementation of MM reduces the complexity of numerical algorithms by the same factor.

We need an implementation of MM, which reaches flops, which are close to the peak flops of the corresponding device.

Computer memory hierarchy and blocking

A naïve algorithm is slow, since it doesn't use the benefits of fast memory (cache) and does not use parallelization ability.
Typically the data is stored in RAM and in order to make computations we need to transfer it to the processor back and forth. To speed up the process we should make use of cache: split the matrix into blocks. If A_{11}, B_{11} and their product fit into the cache memory, then we load them only once into the memory. The block size (n) should be chosen as $3(\frac{n}{q})^2 \sim$ cache size, where q $= \frac{matrix\_size}{cahce\_size}$.
It's worth to compute AB^T, not AB, in this case we access elements sequentially.

Standard libraries for linear algebra

The approach of using block version of algorithms is a core of BLAS.

- BLAS-1, operations like $c = a + b$: $\mathcal{O}(n)$ data, $\mathcal{O}(n)$ operations
- BLAS-2, operations like matrix-by-vector product: $\mathcal{O}(n^2)$ data, $\mathcal{O}(n^2)$ operations
- BLAS-3, matrix-by-matrix product: $\mathcal{O}(n^2)$ data, $\mathcal{O}(n^3)$ operations

Packages:

- ATLAS automatically adapts to a particular system architechture.
- LAPACK provides high-level linear algebra operations (e.g. matrix factorizations).
- Intel MKL provides re-implementation of BLAS and LAPACK, optimized for Intel processors. Available in Anaconda Python distribution. MATLAB uses Intel MKL by default.
- OpenBLAS is an optimized BLAS library based on GotoBLAS. (The only open source BLAS library now, but it's not really well developed)
- PyTorch: BLAS (MV and MM), LAPACK (solution of linear systems, decompositions)
- For GPU it was implemented special cuBLAS

Strassen algorithm $\mathcal{O}(n^{2.807\ldots})$

Naïve multiplication contains 8 multiplications and 4 additions, but Strassen used 18 additions and only 7 multiplications:
$c_{11} = f_1 + f_4 - f_5 + f_7$,
$c_{12} = f_3 + f_5$,
$c_{21} = f_2 + f_4$,
$c_{22} = f_1 - f_2 + f_3 + f_6$, where
$f_1 = (a_{11} + a_{22})(b_{11} + b_{22})$,
$f_2 = (a_{21} + a_{22})b_{11}$,
$f_3 = a_{11}(b_{12} - b_{22})$,
$f_4 = a_{22}(b_{21} - b_{11})$,
$f_5 = (a_{11} + a_{12})b_{22}$,
$f_6 = (a_{21} - a_{11})(b_{11} + b_{12})$,
$f_7 = (a_{12} - a_{22})(b_{21} + b_{22})$.

Reducing number of multiplications helps, since these formulas also hold when a_{11}, a_{12} and etc. are block matrices.
Strassen algorithm looks as follows.

- we split matrices A and B of sizes $n \times n$, $n = 2^d$ into 4 blocks of size $\frac{n}{2} \times \frac{n}{2}$
- we calculate multiplications in the described formulas recursively.

Complexity of the Strassen algorithm

Number of multiplications
$M(n)$ is number of multiplications used to multiply 2 matrices of sizes $n \times n$ using the divide and conquer concept. Then for naïve algorithm we have number of multiplications:

$$M_{naive}(n) = 8M_{naive}\left(\tfrac{n}{2}\right) = 8^2 M_{naive}\left(\tfrac{n}{4}\right) = \cdots = 8^{d-1}M(1) = 8^d = 8^{\log_2 n} = n^{\log_2 8} = n^3$$

So, even when using divide and coquer idea we can not be better than n^3.
For the Strassen algorithm we have:

$$M_{strassen}(n) = 7M_{strassen}\left(\tfrac{n}{2}\right) = 7^2 M_{strassen}\left(\tfrac{n}{4}\right) = \cdots = 7^{d-1}M(1) = 7^d = 7^{\log_2 n} = n^{\log_2 7}$$

Number of additions
For the Strassen algorithm: $A_{strassen}(n) = 7A_{strassen}\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$

Since on the first level we have to add $\frac{n}{2} \times \frac{n}{2}$ matrices 18 times and then go deeper for each of the 7 multiplications. Thus,

$$A_{strassen}(n) = 7A_{strassen}\left(\tfrac{n}{2}\right) + 18\left(\tfrac{n}{2}\right)^2 = 7\left(7A_{strassen}\left(\tfrac{n}{4}\right) + 18\left(\tfrac{n}{4}\right)^2\right) + 18\left(\tfrac{n}{2}\right)^2 = 7^2 A_{strassen}\left(\tfrac{n}{4}\right) + 7\cdot 18\left(\tfrac{n}{4}\right)^2 + 18\left(\tfrac{n}{2}\right)^2 = \cdots = 18\sum_{k=1}^{d} 7^{k-1}\left(\tfrac{n}{2^k}\right)^2 = \tfrac{18}{4}n^2 \sum_{k=1}^{d}\left(\tfrac{7}{4}\right)^{k-1} = \tfrac{18}{4}n^2 \frac{\left(\frac{7}{4}\right)^d - 1}{\frac{7}{4} - 1} = 6n^2\left(\left(\tfrac{7}{4}\right)^d - 1\right) \leqslant 6n^2\left(\tfrac{7}{4}\right)^d = 6n^{\log_2 7}$$ (since $4^d = n^2$ and $7^d = n^{\log_2 7}$).

Total complexity
$M_{strassen}(n) + A_{strassen}(n) = 7n^{\log_2 7}$. Strassen algorithm becomes faster when

$$2n^3 > 7n^{\log_2 7},$$
$$n > 667$$

4. Matrix norms. Example of a norm that is not a matrix norm (does not satisfy submultiplicative property). Scalar product. Cauchy-Schwarz-Bunyakovsky inequality.

Matrix norms

$\|\cdot\|$ is called a **matrix norm** if it is a vector norm on the vector space of $n \times m$ matrices:

- $\|A\| \geq 0$ and if $\|A\| = 0$ then $A = 0$
- $\|\alpha A\| = |\alpha| \|A\|$
- $\|A + B\| \leq \|A\| + \|B\|$ (triangle inequality)

Additionally some norms can satisfy the **submultiplicative** property

$$\|AB\| \leq \|A\| \|B\|$$

These norms are called **submultiplicative** norms. Example of a non-submultiplicative norm is **Chebyshev** norm

$$\|A\|_C = \max_{i,j} |a_{ij}|$$

Important matrix norms

Operator norms

- The most important class of the matrix norms is the class of **operator norms**. They are defined as

  $$\|A\|_{*,**} = \sup_{x \neq 0} \frac{\|Ax\|_*}{\|x\|_{**}},$$

  where $\|\cdot\|_*$ and $\|\cdot\|_{**}$ are **vector norms**.
- It is easy to check that operator norms are submultiplicative if $\|\cdot\|_* = \|\cdot\|_{**}$. Otherwise, it can be non-submultiplicative, think about example.
- **Frobenius norm** is a matrix norm, but not an operator norm, i.e. you can not find $\|\cdot\|_*$ and $\|\cdot\|_{**}$ that induce it.
- This is a nontrivial fact and the general criterion for matrix norm to be an operator norm can be found in [Theorem 6 and Corollary 4].

  For $\|\cdot\|_* = \|\cdot\|_{**}$ let us check on the blackboard!

Matrix p-norms

Important case of operator norms are matrix p-norms, which are defined for $\|\cdot\|_* = \|\cdot\|_{**} = \|\cdot\|_p$.

Among all p-norms three norms are the most common ones:

- $p = 1$, $\|A\|_1 = \max\limits_j \sum\limits_{i=1}^n |a_{ij}|$.

- $p = 2$, spectral norm, denoted by $\|A\|_2$.

- $p = \infty$, $\|A\|_\infty = \max\limits_i \sum\limits_{j=1}^m |a_{ij}|$.

Spectral norm

- Spectral norm, $\|A\|_2$ is one of the most used matrix norms (along with the Frobenius norm).
- It can not be computed directly from the entries using a simple formula, like the Frobenius norm, however, there are efficient algorithms to compute it.
- It is directly related to the **singular value decomposition** (SVD) of the matrix. It holds

  $$\|A\|_2 = \sigma_1(A) = \sqrt{\lambda_{\max}(A^* A)}$$

  where $\sigma_1(A)$ is the largest singular value of the matrix A and * is a *conjugate transpose* of the matrix.

Scalar product

While norm is a measure of distance, the **scalar product** takes angle into account. It is defined as

- **For vectors:**

  $$(x, y) = x^* y = \sum_{i=1}^{n} \overline{x}_i y_i,$$

  where $\overline{x}$ denotes the *complex conjugate* of x. The Euclidean norm is then

  $$\|x\|_2 = \sqrt{(x, x)},$$

  or it is said the norm is **induced** by the scalar product.

- **For matrices** (Frobenius scalar product):

  $$(A, B)_F = \sum_{i=1}^{n} \sum_{j=1}^{m} \overline{a}_{ij} b_{ij} \equiv \operatorname{trace}(A^* B),$$

  where trace(A) denotes the sum of diagonal elements of A. One can check that $\|A\|_F = \sqrt{(A, A)_F}$.

Remark. The angle between two vectors is defined as

$$\cos\phi = \frac{(x, y)}{\|x\|_2 \|y\|_2}.$$

Similar expression holds for matrices.

Cauchy-Schwarz-Bunyakovsky inequality

An important property of the scalar product is the **Cauchy-Schwarz-Bunyakovski inequality**:

$$|(x, y)| \leq \|x\|_2 \|y\|_2,$$

and thus the angle between two vectors is defined properly.

Examples

Several examples of optimization problems where matrix norms arise:

- $\min\limits_{\text{rank}(A_r)=r} \|A - A_r\|$ — finding best rank-r approximation.

  SVD helps to solve this problem for $\|\cdot\|_2$ and $\|\cdot\|_F$.

- $\min\limits_{B} \|P_\Omega \odot (A - B)\| + \text{rank}(B)$ — matrix completion.

$$(P_\Omega)_{ij} = \begin{cases} 1 & i, j \in \Omega \\ 0 & \text{otherwise}, \end{cases}$$

  where $\odot$ denotes Hadamard product (elementwise)

- $\min\limits_{B,C \geq 0} \|A - BC\|_F$ — nonnegative matrix factorization.

  Symbol $B \geq 0$ here means that all elements of B are nonnegative.

5. Unitary matrices and their properties. Examples: Fourier matrix, permutation matrix, Householder reflections, Givens rotations.

Why Unitary?

For stability it is really important that the error does not grow after we apply some transformations. We want to use transformations that don't increase (or even preserve) the error.

Suppose you are given $\widehat{x}$ — the approximation of x such that,

$$\frac{\|x-\widehat{x}\|}{\|x\|} \leq \varepsilon.$$

Q: For which kind of matrices the norm of the vector **will not change**, so that

$$\frac{\|U(x-\widehat{x})\|}{\|Ux\|} = \frac{\|x-\widehat{x}\|}{\|x\|}.$$

A: For the euclidean norm $\|\cdot\|_2$ the answer is **unitary** (or orthogonal in the real case) matrices.

Unitary matrix

Complex $n \times n$ square matrix is called **unitary** if

$$U^*U = I_n,$$

Properties:

- $\|Uz\|_2 = \|z\|_2$ for all z
- Columns and rows of unitary matrices both form orthonormal basis in C^n
- $U^*U = I_n$ — left unitary for $m > n$
- $UU^* = I_m$ — right unitary for $m < n$
- Important property: **a product of two unitary matrices is a unitary matrix:**

$$(UV)^*UV = V^*(U^*U)V = V^*V = I,$$

- Unitary matrices also do not change matrix norms $\|\cdot\|_2$ and $\|\cdot\|_F$, i.e. for any square A and unitary U,V:

$$\|UAV\|_2 = \|A\|_2 \qquad \|UAV\|_F = \|A\|_F.$$

Important examples

- Permutation matrix P whose rows (columns) are permutation of rows (columns) of the identity matrix.
- Fourier matrix

$$F_n = \frac{1}{\sqrt{n}} \left\{ e^{-i\frac{2\pi kl}{n}} \right\}_{k,l=0}^{n-1}$$

- Householder matrices
- Givens (Jacobi) matrices

  Householder and Givens are two important classes of unitary matrices, using composition of which we can construct any unitary matrix

Householder matrix

Householder matrix is the matrix of the form

$$H \equiv H(v) = I - 2vv^*,$$

where v is an $n \times 1$ column and $v^*v = 1$.

Properties:

- H is unitary and Hermitian ($H^* = H$)
- $Hx = x - 2(v^*x)v$
- Householder transformation can zero all elements of a vector except first one:

$$H \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix} = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Here $H = I - 2vv^*$ and $v = x - \alpha e_1, \alpha = \|x\|, e_1 = (1, 0, \cdots 0)^T$

Actually, there is a mistake/misprint above, and $v = \dfrac{x - x_1 e_1}{\|x - x_1 e_1\|}$.

Using this property we can make arbitrary matrix A upper triangular:

$$H_4 H_3 H_2 H_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and as a corollary we can do QR decomposition via Householder algorithm.

QR decomposition: Every $A \in \mathrm{C}^{n \times m}$ can be represented as $A = QR$, where Q is unitary and R is upper triangular.

Givens matrix

Givens matrix is a rotation matrix:

$$G = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix},$$

- For a general case, we select two (i, j) planes and rotate vector x

$$x' = Gx,$$

only in the i-th and j-th positions:

$$x'_i = x_i \cos\alpha - x_j \sin\alpha, \quad x'_j = x_i \sin\alpha + x_j \cos\alpha,$$

with all other x_i remain unchanged.

- Therefore, we can make elements in the j-th position (x'_j) zero by choosing α such that

$$\cos\alpha = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad \sin\alpha = -\frac{x_j}{\sqrt{x_i^2 + x_j^2}}$$

Similarly we can make matrix upper-triangular (QR-decomposition) using Givens rotations:

$$\begin{bmatrix} \times & \times & \times \\ * & \times & \times \\ * & \times & \times \end{bmatrix} \to \begin{bmatrix} * & \times & \times \\ * & \times & \times \\ 0 & \times & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times \\ 0 & * & \times \\ 0 & * & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}$$

Explanation: for example, we select the first column and change two elements in it (rotate the vector) so that one of them becomes zero, and so on

Givens vs. Householder transformations

– Householder reflections are useful for dense matrices (complexity is aproc twice smaller than for Jacobi) and we need to zero large number of elements.

– Givens rotations are more suitable for sparse matrices or parallel machines as they act locally on elements.

6. Matrix rank. Skeleton decomposition (with proof). Low-rank matrix storage and matrix-vector product. Instability of matrix rank.

Support info:

$A = [a_1, \dots, a_m]$, where $a_m \in \mathbb{C}^{n\times 1}$.
$y = Ax \iff y = a_1x_1 + a_2x_2 + \ldots + a_mx_m$.

Definition1. Vectors a_i are called **linearly dependent**, if there exist simultaneously non-zero coefficients x_i such that $\sum_i a_ix_i = 0$, or in the matrix form: $Ax = 0, \quad \|x\| \neq 0$.

In this case, we say that the matrix A has a non-trivial **nullspace** (or **kernel**) denoted by $N(A)$ (or $\ker(A)$). Vectors that are not linearly dependent are called **linearly independent**.

Definition2. A **linear space** spanned by vectors $\{a_1, \dots, a_m\}$ is defined as all possible vectors of the form

$$\mathcal{L}(a_1, \dots, a_m) = \left\{ y : y = \sum_{i=1}^{m} a_ix_i, \, \forall x_i, \, i = 1, \dots, n \right\},$$

In the matrix form, the linear space is a set of all y such that: $y = Ax$.

This set is also called the **range** (or **image**) of the matrix, denoted by $\mathrm{range}(A)$ (or $\mathrm{im}(A)$) respectively.

Definition3. The dimension of a linear space $\mathrm{im}(A)$ denoted by $\dim \mathrm{im}(A)$ is the minimal number of vectors required to represent each vector from $\mathrm{im}(A)$.

Definition4. A matrix $A \in \mathbb{R}^{m\times n}$ is called of **full-rank**, if $\mathrm{rank}(A) = \min(m, n)$.

Matrix rank

- Rank of a matrix A is a maximal number of linearly independent *columns* in a matrix A, or the **dimension of its column space** $= \dim \mathrm{im}(A)$.

- You can also use linear combination of *rows* to define the rank, i.e. formally there are two ranks: column rank and row rank of a matrix.

Theorem: The dimension of the column space of the matrix is equal to the dimension of its row space.

- In the matrix form this fact can be written as $\dim\ \mathrm{im}(A) = \dim\ \mathrm{im}(A^\top)$.
- Thus, there is a single rank!

Skeleton decomposition

A very useful representation for computation of the matrix rank is the **skeleton decomposition** and is closely related to the rank. This decompositions explains, why and how matrices of low rank can be compressed.
It can be graphically represented in the matrix form:

$$A = C\widehat{A}^{-1}R,$$

where C are some $r = \mathrm{rank}(A)$ columns of A, R are some r rows of A and $\widehat{A}$ is the **nonsingular** submatrix on the intersection.

Proof for the skeleton decomposition
- Let $C \in \mathbb{C}^{n \times r}$ be the r columns based on the nonsingular submatrix $\widehat{A}$. Therefore they are linearly independent.
- Take any other column a_i of A. Then a_i can be represented as a linear combination of the columns of C, i.e. $a_i = Cx$.
- $a_i = Cx_i$ are n equations. We take r equations of those corresponding to the rows that contain $\widehat{A}$ and get the equation

$$\widehat{r}_i = \widehat{A}x_i \quad \Longrightarrow \quad x_i = \widehat{A}^{-1}\widehat{r}_i$$

Thus, $a_i = C\widehat{A}^{-1}\widehat{r}_i$ for every i and

$$A = [a_1, \dots, a_m] = C\widehat{A}^{-1}R.$$

A closer look on the skeleton decomposition

- Any rank-r matrix can be written in the form

$$A = C\widehat{A}^{-1}R,$$

where C is $n \times r$, R is $r \times m$ and $\widehat{A}$ is $r \times r$, or

$$A = UV,$$

where U and V are not unique, e.g. $U = C\widehat{A}^{-1}$, $V = R$.
- The form $A = UV$ is standard for skeleton decomposition.
- Thus, every rank-r matrix can be written as a product of a "skinny" ("tall") matrix U by a "fat" ("short") matrix V.

Storage and matrix-vector product

It is interesting to note, that for the rank-r matrix

$$A = UV$$

only U and V can be stored, which gives us $(n+m)r$ parameters, so it can be used for compression. We can also compute matrix-by-vector Ax product much faster:

- Multiplication $y = Vx$ costs $\mathcal{O}(mr)$ flops.

- Multiplication $z = Uy$ costs $\mathcal{O}(nr)$ flops.

The same works for addition, elementwise multiplication, etc. For addition:

$$A_1 + A_2 = U_1V_1 + U_2V_2 = [U_1|U_2][V_1^\top|V_2^\top]^\top$$

Instability of the matrix rank

For any rank-r matrix A with $r < \min(m, n)$ there is a matrix B such that its rank is equal to $\min(m, n)$ and

$$\|A - B\| = \epsilon.$$

Q: So, does this mean that numerically matrix rank has no meaning? (I.e., small perturbations lead to full rank!)

A: No. We should find a matrix B such that $\|A-B\|=\epsilon$ and B has minimal rank. So we can only compute rank with given accuracy ϵ.

7. Singular value decomposition (SVD). Proof of its existence. Null space and image of A and A^* in terms of singular vectors. Interpretation of SVD as a separation of variables.

Theorem 1. Any matrix $A \in \mathbb{C}^{n\times m}$ can be written as a product of 3 matrices:

$$A = U\Sigma V^*,$$

where

- U is an $n \times K$ unitary matrix,
- V is an $m \times K$ unitary matrix, $K = \min(m, n)$,
- Σ is a diagonal matrix with non-negative elements $\sigma_1 \geq \ldots, \geq \sigma_K$ on the diagonal.
- Moreover, if $\operatorname{rank}(A) = r$, then $\sigma_{r+1} = \cdots = \sigma_K = 0$.

Proof.

- Apparently, matrix A^*A (*Gram matrix*) is Hermitian, hence it can be diagonalized by unitary transformation (that means, there exist such unitary matrix V and diagonal matrix Σ, that $A^*A = V\Sigma^2V^*$). It follows from the fact (lecture 6, applications of the Schur theorem), that A^*A is a *normal* matrix.
- $A^*A \geq 0$. It follows from simple reasoning: $(A^*Ax, x) = (Ax, Ax) = \|Ax\|^2 \geq 0$. So, A^*A is non-negative definite, which means that its eigenvalues are non-negative. Therefore, there exists unitary matrix $V = [v_1, \ldots, v_n]$ such that

  $$V^*A^*AV = \operatorname{diag}(\sigma_1^2, \ldots, \sigma_n^2), \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n.$$

  Let $\sigma_i = 0$ for $i > r$, where r is some integer. If $r \leq n$, we can leave only r first columns of V and $V_r = [v_1, \ldots, v_r]$, $\Sigma_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$. Then

  $$V_r^*A^*AV_r = \Sigma_r^2 \quad \Longrightarrow \quad (\Sigma_r^{-1}V_r^*A^*)(AV_r\Sigma_r^{-1}) = I.$$

  As a result, matrix $U_r = AV_r\Sigma_r^{-1}$ satisfies $U_r^*U_r = I$ and hence has orthogonal columns. Let us add to U_r any orthogonal columns that are orthogonal to columns in U_r (it can be done, for example, with Householder transformations) and denote this matrix as U . Then

  $$AV = U\begin{bmatrix}\Sigma_r & 0\\ 0 & 0\end{bmatrix} \quad \Longrightarrow \quad U^*AV = \begin{bmatrix}\Sigma_r & 0\\ 0 & 0\end{bmatrix}.$$

  Since multiplication by non-singular matrices does not change rank of A, we have $r = \operatorname{rank}(\Sigma_r) = \operatorname{rank}(A)$.

□

Null space and image of A

$$\ker(A) = \mathcal{L}\{v_{r+1}, \ldots, v_n\}, \quad \operatorname{im}(A) = \mathcal{L}\{u_1, \ldots, u_r\}$$

$$\ker(A^*) = \mathcal{L}\{u_{r+1}, \ldots, u_n\}, \quad \operatorname{im}(A^*) = \mathcal{L}\{v_1, \ldots, v_r\},$$

where $\mathcal{L}\{e_1, \ldots, e_n\}$ is a linear space, spanned by $\{e_1, \ldots, e_n\}$

Proof. 1. Proof for $\ker(A)$. Let $x = \sum\limits_{i=r+1}^{n} w_i v_i$, where $w_i \in \mathbb{C}$. Then

$$Ax = \sum_{i=r+1}^{n} w_i(Av_i) = \sum_{i=r+1}^{n} w_i U\Sigma V^* v_i = \sum_{i=r+1}^{n} w_i U\Sigma e_i = 0,$$

because $\sigma_{r+1} = \cdots = \sigma n = 0$.

□

Separation of variables

We can use SVD to compute approximations of function-related matrices, i.e. the matrices of the form

$$a_{ij} = f(x_i, y_j),$$

where f is a certain function, and $x_i, \quad i = 1, \dots, n$ and $y_j, \quad j = 1, \dots, m$ are some one-dimensional grids. These matrices are often of low-rank. So finding rank-1 approximation of a function-related matrix via SVD-decomposition can be regarded as a separation of variables in the corresponding function.

8. Eckart-Young theorem (proof for the spectral norm). Applications of SVD: latent semantic analysis, collaborative filtering in recommender systems, dense matrix compression.

Formulation

Best rank k approximation for a matrix with SVD $A = U\Sigma V^T$ is $A_k = \sum_{i=1}^{k} u_i \sigma_i v_i^T$, where v_i, u_i are column of U, V. I.e.

$$\text{argmin}_{B_k:\text{rank}(B_k)=k} \|A - B_k\|_2 = \sum_{i=1}^{k} u_i \sigma_i v_i^T = A_k$$

Proof: First, calculate minimal norm:

$$\|A - A_k\|_2 = \| \sum_{i=k+1}^{N} u_i \sigma_i v_i^T \|_2 = \sigma_{k+1}$$

Next, compare with a matrix B of rank k, for which is true $B = XY^T$, where X and Y have k columns:

$$\|A - B\|_2 = \|\sum_{i=1}^{N} u_i \sigma_i v_i^T - XY^T\|_2$$

Since Y has k columns, there exists a vector y such that $y = \sum_{i=1}^{k+1} \alpha_i v_i$ and $Y^t y = 0$ (i.e. y that lies in given $k+1$-dimentional space and which is orthogonal to k given vectors). After normalizing this vector:

$$\|A - B\|_2 = \|\sum_{i=1}^{N} u_i \sigma_i v_i^T - XY^T\|_2 \geq \|(\sum_{i=1}^{N} u_i \sigma_i v_i^T - XY^T)y\|_2 = \|\sum_{i=1}^{N} u_i \sigma_i v_i^T y - XY^T y\|_2 = \|\sum_{i=1}^{k+1} u_i \sigma_i \alpha_i\|_2 = \sqrt{\sum_{i=1}^{k+1} \sigma_i^2 \alpha_i^2} \geq \sqrt{\sum_{i=1}^{k} \sigma_i^2 \alpha_i^2} \geq \sigma_{k+1} = \|A - A_k\|_2$$

Applications

1. Latent semantic analysis: to find "similarity" metric for large number of documents. Steps:
   (a) Count words in each document, construct matrix $C_{w,d}$
   (b) Use SVD to reduce its rank, effectively reducing number of relevant words: $C = WD^T$
   (c) Any two documents can be "compared" by taking inner product of appropriate columns in matrix D. Values close to 1 mean similar documents.

2. Collaborative filtering in recommender systems: give estimation of user's rating for given item based on his history of ratings. Same idea:
   (a) Construct matrix $C_{u,i}$ of all users' ratings on all items
   (b) Use SVD to reduce its rank, effectively reducing number of relevant items: $C = WD^T$
   (c) Now we get all reduced items' vectors which a particular user had rated before. We know all the ratings he had given to these columns. So, we can reconstruct such a vector in reduced user's space, that it minimizes error in ratings (product between user and item).
   (d) Having user's vector and item's vector from reduced space, inner product gives rating estimation.

3. By the theorem above, we can construct best possible k-rank (i.e. using $2nk$ space for storage) matrix using SVD decomposition. We can also estimate error of such representation.

9. LU-decomposition. Connection with Gaussian elimination. Complexity of decomposing and solving linear system. The concept of pivoting.

There are linear systems out there. Like this one:

$$\begin{gathered} 2x_1 + 3x_2 + x_3 = 11 \\ 4x_1 - x_2 + x_3 = 5 \\ x_1 - x_2 + 2x_3 = 4 \end{gathered}$$

They can be represented in a matrix form:

$$\begin{pmatrix} 2 & 3 & 1 \\ 4 & -1 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 2 \\ 4 \end{pmatrix}$$

or $Ax = b$

One way to solve such a linear system is Gaussian elimination. In Gaussian elimination we perform 2 steps: forward and backward. In forward step we try to bring a matrix into upper triangular form: that is very convenient, since in that case the have something like

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{13} \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

And can immediately infer values of x going from bottom to top: x_3 is equal to b_3, and then x_2 is obtained from $a_{22}x_2 + a_{23}b_3 = b_2$ and so on. This is the backward step.

Let's concentrate on the forward step, though - on bringing matrix A to an upper triangular form. In classic Gaussian elimination you do it as follows: go from top to bottom and subtract the upper row from the lower with an appropriate coefficient, that would convert first non-zero element in the lower row to zero. The thing is that you can represent these operations as matrix multiplications. Let's take a first step of elimination on our example matrix - we'll need to subtract 2*first row from the second row. In matrix form it would be equal to multiplication by identity, but with -2 in the position [0,1]:

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 \\ 4 & -1 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 8 \\ 2 \\ 4 \end{pmatrix}$$

which is the same thing as (result of the first step of Gaussian elimination)

$$\begin{pmatrix} 2 & 3 & 1 \\ 0 & -7 & -1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ -6 \\ 4 \end{pmatrix}$$

In this fasion we can represent the whole Gaussian elimination process as sequential multiplication by 3 matrices:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{14} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 \\ 4 & -1 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} =$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{14} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 8 \\ 2 \\ 4 \end{pmatrix}$$

Which will bring the left-hand side to an upper-triangular form, thus completing the forward step of Gaussian elimination. Let's have a look at the left-hand side, though. We effectively have shown that multiplying our matrix A by a bunch of matrices (M_1, M_2, M_3) we get an upper triangular matrix U:

$$M_3 M_2 M_1 A = U$$

And we're just one step away from LU decomposition! We sequentially multiply both sides by the inverses of M_i and get

$$A = M_1^{-1} M_2^{-1} M_3^{-1} U$$

Notice that inverses for matrices M_i are calculated extremely easily: if we subtracted some row with some coefficient, to get an inverse, we need to add this row back with the same coefficient. So, for example, an inverse for matrix M_1 would look like

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Also notice, that all the matrices M_i^{-1} are lower triangular. So, their product will also be lower triangular. And that's how we get to LU decomposition:

$$A = M_1^{-1} M_2^{-1} M_3^{-1} U = LU$$

Now, a problem along the way may arise in Gaussian elimination. Let's say, we had a different matrix:

$$A = \begin{pmatrix} 1 & 3 & 2 \\ 2 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix}$$

After eliminating the first variable from the first column we get

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & 2 \\ 2 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 \\ 0 & 0 & 1 \\ 0 & -10 & -7 \end{pmatrix}$$

How do we eliminate -10 now? The problem is that there is a 0 in the middle, and we can't subtract the second row from the third with any coefficient and zero out -10. The solution is to use a permutation matrix to swap the second row and the 3d one:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & 2 \\ 2 & 6 & 5 \\ 4 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 \\ 0 & -10 & -7 \\ 0 & 0 & 1 \end{pmatrix}$$

And we get our upper triangular form. The rest of decomposition is the same:

$$A = M_1^{-1} M_2^{-1} P_1^{-1} U = L^{'} P_1^{-1} U = P_1^{-1} P_1 L^{'} P_1^{-1} U = P_1^{-1} L U = PLU$$

Here I used that $PLP^{-1} = L'$ is still lower triangular (more detailed example here) This is PLU decomposition, or LU with pivoting. The only thing there is left to do is to discuss complexity. Complexity of naive LU through Gaussian elimination is as follows. We need to perform n elimination steps (on each step we get zeros in one more "lower triangular column", and we want to introduce zeros in all the lower triangle, containing n columns. More precisely, n-1, but who cares). During each of the elimination steps we need to perform n^2 subtractions - because we effectively modify every cell in our matrix. Well, as I understand, this number is also an overestimation - because on the first step we will modify the whole matrix except for the first row, that's $n(n-1)$ entries, at the second step we'll modify $(n-1)(n-2)$ entries and so on. But the order is n^2. As I said, overall we do n steps, so

the resulting complexity is $O(n^3)$.
Side notes:

- There is also Block LU:

  $$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}$$

  Where the Gaussian elimination can be performed recursively block-wise and the aggregated in the final result:

  $$\begin{aligned} A_{11} &= L_{11}U_{11} \\ U_{12} &= L_{11}^{-1}A_{12} \\ L_{21} &= A_{21}U_{11}^{-1} \\ L_{22}U_{22} &= A_{22} - L_{21}U_{11} \end{aligned}$$

  The advantage of such an approach is that one can apply the Strassen algorithm for matrix multiplication for off-diagonal blocks.

- LU decomposition is not unique - there may be different numbers on the diagonal of L and U. The convention is that the diagonal of L always consists of ones, and the diagonal of U is then defined unambiguously.

- For strictly regular matrices there always is LU decomposition (that is, you don't need pivoting - A=PLU). A strictly regular matrix is something not very intuitive (for me) - it's a matrix in which all the principal minors are non-singular. Minors are those thingies you take when computing determinant - submatrices without a specific row and a specific column. And principle minors are those submatrices, which you get removing from initial matrix rows and columns with the same numbers (like, row 1, col 1 or row 2 col 2, etc.).

10. Cholesky factorization (with proof of its existence).

Positive definite matrices and Cholesky factorization

If the matrix is Hermitian positive definite, i.e.

$$A = A^*, \quad (Ax, x) > 0, \quad x \neq 0,$$

then it can be factored as

$$A = RR^*,$$

where R is lower triangular.

Proof of existence of Cholesky factorization:

- Fistly, we prove that the matrix is non-singular ($det(A) > 0$): there is no vector x,s.t. $\|x\| \neq 0$,
- As matrix A is positive definite, we can prove by contradiction that there is no such vector x such that Ax=0:

  Then for such vector x,that Ax=0 $(Ax, x) = (Ax)^* x = x^* A^* x = x^* Ax = 0$ This is a contradiction to the definition of the positive definite matrix.

  Thus, the matrix is strictly regular. .
- As the matrix is non-singular, then $det(A) \neq 0$ and all leading priciple minors are above zero because matrix is positive definite. Thus, LU decomposition exists for such matrix generated in a procedure of Gaussian elimination. Thus, we can constructively prove the existence of $A = RR^*$.
- The idea is to take the first row and eliminate it in Gaussian elimination and scale the element by the square root from the Hermitian matrix.
- Here is an example of the construction of Cholesky decomposition using Gaussian elimination for 3×3 matrix: $A = LL^*$,where $L = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix}$.

  A Hermitian matrix A

$$A = \begin{pmatrix} A_{11} & A_{21} & A_{31} \\ A_{21} & A_{22} & A_{32} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = LL^* = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix} =$$

$$\begin{pmatrix} L_{11}^2 & L_{21}L_{11} & L_{31}L_{11} \\ L_{21}L_{11} & L_{21}^2 + L_{22}^2 & L_{31}L_{21} + L_{32}L_{22} \\ L_{31}L_{11} & L_{31}L_{21} + L_{32}L_{22}^2 & L_{31}^2 + L_{32} + L_{33}^2 \end{pmatrix}$$

  Thus, as we have the same number of variables in L as in A we can find the formulas for elements of factorization matrixes using elements of matrix A: The exact formula for 3×3 case is:

$$L = \begin{pmatrix} \sqrt{A_{11}} & 0 & 0 \\ A_{21}/L_{11} & \sqrt{A_{22} - L_{21}^2} & 0 \\ L_{31}/L_{11} & \frac{A_{32} - L_{31}L_{21}}{L_{22}} & \sqrt{A_{33} - L_{31}^2 - L_{32}^2} \end{pmatrix}$$

  Then from Gaussian elimination algorithm we get formulas for elements of matrix L :

  $L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk} L_{jk}^*}$, $L_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{jk} L_{jk}^*}{L_{jj}}$ for $i > j$

  Thus, by construction using Gaussian elimination the Cholesky decomposition exists for Hermitian positive definite matrix

Applications of Cholesky decomposition

- **Proof of existence of QR decomposition using Cholesky decomposition**

  If we have the representation of the form

  $$A = QR,$$

  then $A^*A = (QR)^*(QR) = R^*(Q^*Q)R = R^*R$, the matrix A^*A is called Gram matrix, and its elements are scalar products of the columns of A.

  1. Proof using Cholesky decomposition (full column rank)

     Assume that A has full column rank. Then, it is simple to show that A^*A is positive definite:

     $$(A^*Ay, y) = (Ay, Ay) = \|Ay\|^2 > 0, y \neq= 0.$$

     Therefore, for positive definite matrix A^*A the Cholesky decomposition: $A^*A = R^*R$ always exists.

     Then the matrix $Q = AR^{-1}$ is unitary:

     $$(AR^{-1})^*(AR^{-1}) = R^{-*}A^*AR^{-1} = R^{-*}R^*RR^{-1} = I.$$

  2. Proof using Cholesky decomposition (rank-deficient case)

     - When an $n \times m$ matrix does not have full column rank, it is said to be rank-deficient. The QR decomposition, however, also exists.
     - For any rank-deficient matrix there is a sequence of full-column rank matrices A_k such that $A_k \to A$ (because in the any vicinity of the degenerate matrix A there exists a perturbation ε_k, such that the matrix $A + \varepsilon_k$ is a full-rank matrix). By decomposing matrix using SVD decomposition:

       $$A = U\Sigma V^*$$

       , we can replace all zero entrees in matrix Σ by $\epsilon = \frac{1}{k}$,which converges to zero in the limit. In this manner we construct a sequence of full rank matrix A_k converging to not full rank A
     - Each A_k can be decomposed as $A_k = Q_kR_k$ as we proved QR decomposition for full-rank matrix.
     - The set of all unitary matrices of all unitary matrix is closed under multiplication and bounded as $Q_kQ_k^* = I$, sum of squares of each element is less than 1.
     - The set of all unitary matrices is compact (closed and bounded in finite dimensional space of matrices), thus there exists a converging subsequence $Q_{n_k} \to Q$, and $Q^*A_k \to Q^*A = R$, which is triangular.

11. Neumann series. Bounds for $\|(I-A)^{-1}\|$ and $\frac{\|(A+\Delta A)^{-1}-A^{-1}\|}{\|A^{-1}\|}$ (both with proofs). Estimate of a perturbation of a solution in terms of condition number (with proof). Linear systems with "consistent" and "inconsistent" right-hand sides and behavior of solution error for large condition numbers.

Generally, in this ticket we want to understand the reasons why solving linear systems sometimes can lead to huge errors. The main point will be *conditional number*, but let's first start with helpful for provents thing:

Neumann series.

If a matrix F is such that $\|F\|< 1$ holds, then the matrix $(I-F)$ is invertible and

$$(I-F)^{-1} = I + F + F^2 + F^3 + \ldots = \sum_{k=0}^{\infty} F^k.$$

Note that it is a matrix version of the geometric progression:

$$\frac{1}{1-q} = \sum_{k=0}^{\infty} q^k$$

for $0 < |q|< 1$.

What norm is considered here? What is the "best possible" norm here? I'm not sure, but probably Euclidean or Spectral norms, or use submultiplicity property ($\|AB\|\leq \|A\|\|B\|$).

but $|\lambda_{\max}(F)| < 1$ is definitely a condition for convergence of given series (this is about spectral norm, which is for square matrices as and F (also square, condition number is also defined only for square matrices because of inverse) in Neumann series. Read more here). Let's prove that the series

$\sum_{k=0}^{\infty} F^k$ converges.

Like in the scalar case, we have

$$(I-F)\sum_{k=0}^{N} F^k = (I-F)(I+F+F^2+\cdots+F^N) = (I+F+F^2+\cdots+F^N)-(F+F^2+F^3+\cdots+F^{N+1}) = (I-F^{N+1})$$

$$\to I, \quad N\to+\infty$$

Indeed,

$$\|(I-F^{N+1})-I\|= \|F^{N+1}\|\leqslant \|F\|^{N+1}\to 0, //\text{because norm<1}// \quad N\to+\infty.$$

After it, we can also estimate the **norm of inverse**:

$$\left\|\sum_{k=0}^{N} F^k\right\| \leq |\text{submultiplicity property of norm}|\leq \sum_{k=0}^{N}\|F\|^k\|I\|\leq |\text{Neuman series}|\leq \frac{\|I\|}{1-\|F\|}$$

Linear systems and their perturbations

Suppose, we have linear system

$$Ax = f$$

If we have computed A^{-1}, we simply obtain precise solution $x = A^{-1}f$.

We can estimate, how the perturbation of the matrix influences the inverse matrix. We assume that the perturbation E is small in the sense that $\|A^{-1}E\|< 1$. Then:

$$(A+E)^{-1} = \sum_{k=0}^{\infty}(-A^{-1}E)^k \cdot A^{-1}$$

as $(A+E)^{-1} = (A(I+A^{-1}E))^{-1} = (I+A^{-1}E)^{-1}A^{-1} = \sum_{k=0}^{\infty}(-A^{-1}E)^k A^{-1}$
Finally, let's estimate

$$\|(A+E)^{-1} - A^{-1}\|= \|(\sum_{k=0}^{\infty}(-A^{-1}E)^k A^{-1} - A^{-1}\|= \|(\sum_{k=0}^{\infty}(-A^{-1}E)^k - I)A^{-1}\|$$

$$\leq \|(\sum_{k=0}^{\infty}(-A^{-1}E)^k - I)\|\|A^{-1}\|$$

I can say also, that

$$\|\sum_{k=0}^{\infty}(-A^{-1}E)^k \cdot A^{-1}\|\leq \frac{\|I\|\|A^{-1}\|}{1-\|A^{-1}E\|}$$

Honestly, 'I don't know how they got this. Let it be as is, mb I will finish it later

$$\frac{\|(A+E)^{-1} - A^{-1}\|}{\|A^{-1}\|} \leq \frac{\|A^{-1}\|\|E\|\|I\|}{1-\|A^{-1}E\|}.$$

Now let's consider perturbated system (e.g., due to estimations or measurement errors

$$(A+\Delta A)\widehat{x} = f + \Delta f.$$

Let's try to estimate the difference between true solution x of non-perturbated system and $\hat{x}$ - solution of perturbated system (*also you can good read proofs here, but in Russian and different names for norms*).

$$\hat{x} - x = (A+\Delta A)^{-1}(f+\Delta f) - A^{-1}f = ((A+\Delta A)^{-1} - A^{-1})f + (A+\Delta A)^{-1}\Delta f =$$

$$= ((A(1+A^{-1}\Delta A))^{-1} - A^{-1})f + (1+A^{-1}\Delta A)A^{-1}\Delta f = \left(\sum_{k=0}^{\infty}(-A^{-1}\Delta A)^k - 1\right) A^{-1}f + \sum_{k=0}^{\infty}(-A^{-1}\Delta A)^k A^{-1}\Delta f =$$

$$= \sum_{k=1}^{\infty}(-A^{-1}\Delta A)^k A^{-1}f + \sum_{k=0}^{\infty}(-A^{-1}\Delta A)^k A^{-1}\Delta f$$

$$\frac{\|\hat{x}-x\|}{\|x\|} = \frac{\|(A+\Delta A)^{-1}(f+\Delta f) - A^{-1}f\|}{\|A^{-1}f\|} = \frac{\|(f+\Delta f) - (A+\Delta A)A^{-1}f\|}{\|(A+\Delta A)A^{-1}f\|} =$$

$$= \frac{\|\Delta f - \Delta A A^{-1}f\|}{\|(1+\Delta A A^{-1})f\|} \leq \frac{1}{1-\|\Delta A A^{-1}\|}\frac{\|\Delta f\|}{\|f\|} + \frac{\|\Delta A A^{-1}\|}{1-\|\Delta A A^{-1}\|} \leq \frac{\|A\|\|A^{-1}\|}{1-\|\Delta A A^{-1}\|}\left(\frac{\|\Delta f\|}{\|f\|} + \frac{\|\Delta A\|\|A^{-1}\|}{\|A\|\|A^{-1}\|}\right) =$$

$$= \frac{\|A\|\|A^{-1}\|}{1-\|\Delta A\|\|A^{-1}\|}\left(\frac{\|\Delta f\|}{\|f\|} + \frac{\|\Delta A\|}{\|A\|}\right) = \frac{\|A\|\|A^{-1}\|}{1-\|A\|\|A^{-1}\|\frac{\|\Delta A\|}{\|A\|}}\left(\frac{\|\Delta f\|}{\|f\|} + \frac{\|\Delta A\|}{\|A\|}\right) =$$

$$= \frac{\kappa(A)}{1-\kappa(A)\frac{\|\Delta A\|}{\|A\|}}\left(\frac{\|\Delta f\|}{\|f\|} + \frac{\|\Delta A\|}{\|A\|}\right),$$

where $\kappa(A)$ or sometimes $\varkappa(A)$, $\mu(A)$, $\mathrm{cond}(A)$ is a **condtion number cond(A)** $= \|A\|\|A^{-1}\|$

- The larger the condition number, the less number of digits we can recover. Note that the condition number is different for different norms.
- Note that if $\Delta A = 0$, then $\frac{\|\widehat{x}-x\|}{\|x\|} \leq \text{cond}(A)\frac{\|\Delta f\|}{\|f\|}$
- The spectral norm of the matrix is equal to the **largest singular value**, and the singular values of the inverse matrix are equal to the inverses of the singular values.
- Thus, the condition number in spectral norm is equal to the ratio of the largest singular value and the smallest singular value.

  $\text{cond}_2(A) = \|A\|_2\|A^{-1}\|_2 = \sigma_{\max} \cdot \frac{1}{\sigma_{\min}}$

«The submultiplicative property is needed in many places, for example in the estimates for the error of solution of linear systems (we will cover this topic later). Example of a non-submultiplicative norm is Chebyshev norm».

Linear systems with "consistent" and "inconsistent" right-hand sides and behavior of solution error for large condition numbers.

Remember this guy?

$$H_{ij} = \frac{1}{i+j+1}$$

This is Hilbert matrix

A linear or nonlinear system of equations is called **consistent** if there is at least one set of values for the unknowns that satisfies each equation in the system—that is, when substituted into each of the equations, they make each equation hold true as an identity. In contrast, a linear or non linear equation system is called **inconsistent** if there is no set of values for the unknowns that satisfies all of the equations.

Upper bound ($\Delta A = 0$, then $\frac{\|\widehat{x}-x\|}{\|x\|} \leq \text{cond}(A)\frac{\|\Delta f\|}{\|f\|}$) doesn't work for example for Hilbert matrix, when we jnp.ones(n) right-hand side of the equation (f): conditional number is extreme, but relative error is very small. For random right-hand-side situation is a few worse in terms of error for small size of the system n.Both errors are increasing with increasing of n.

Summing up, this is due to the singular-values decompoisition(??)

$Ax = f$, $A = U\Sigma V^*$, $\Sigma z = U^* f, z = V^* x$ obtained linear system with the diagonal matrix and solution is $z_k = U^* f/\sigma_k$, σ_k are decaying exponentially, but if $U^* f$ also decaying, this might be stable. For $f = (1, 1, ..)$ $U^* f$ decays quite fast and z_k are bounded not so bad ($|z_k| < 10^9$), though for randon rhs f ($|z_k| < 10^15$), that is close to machine infinity.

If rhs f is well approximated by leading singular vectors, accuracy might be much better. However, random has random orientation ...blablabla (hz more, sorry).

What will be with the solution for system with large condition number? We can't say definitely. But if we pereturbate matrix A or rhs f, we can estimate difference $\|\hat{x} - x\|/\|x\|$ as it is shown above.

For example, for

$$\begin{cases} 100u + 99v = 199 \\ 99u + 98v = 197 \end{cases}$$

solution is $u = 1, v = 1$

but for

$$\begin{cases} 100u + 99v = 198,99 \\ 99u + 98v = 197,01 \end{cases}$$

solution is $\hat{u} = 2.97$, $\hat{v} = -0.99$.
This is due to the fact, that $\operatorname{cond}(A)$ is big, about $4 * 10^4$.

Condition number for any (not square, but rectangular, singular) matrix is $\operatorname{cond}(A) = \|A\| \left\|A^\dagger\right\|$

12. Least squares (LS) problem for over- and underdetermined linear systems. Methods of solving LS via QR, pseudoinverse, SVD, padding into a bigger system.

Linear system of equations is called overdetermined if there are more equations than unknown, or, in other words, the matrix of coefficients $A \in \mathbb{R}^{n\times m}$ is such that $n > m$. Conversely, it is called underdetermined if $n < m$.

In general, an overdetermined system does not have an exact solution, that os why the second norm of the error $\|Ax - b\|_2$ is minimized instead, and the problem becomes that of least squares. Underdetermined system, on the other hand, has an infinite number of solutions in general. To constrain the solution, one can minimize $\|Ax - b\|_2 + \|x\|_2$.

Let us have an overdetermined system $Ax = b$, i.e. $n > m$. Then if we have a pseudoinverse $A^\dagger$ of A such that $A^\dagger A = I$, then we must have $x = A^\dagger b$. When A has full column rank, then $A^\dagger = (A^*A)^{-1}A^*$ and $x = (A^*A)^{-1}A^*b$.

A canonical way to solve least squares is using QR decomposition. If we have a QR decomposition of $A = QR$, then

$$x = ((QR)^*QR)^{-1}(QR)^*b = (R^*Q^*QR)^{-1}R^*Q^*b = R^{-1}R^{-*}R^*Q^*b = R^{-1}Q^*b.$$

Thus, since R is upper triangular, the complexity of solving the system is $O(m(n + m))$ ($O(nm)$ to compute Q^*b and $O(m^2)$ to invert R).

We can also do similar things using SVD. We know that if $A = U\Sigma V^*$, then $A^\dagger = V\Sigma^\dagger U^*$, hence the solution can be rewritten as $x = V\Sigma^\dagger U^*b$, which can in turn be rewritten as $x = \sum_{i=1}^{r} \frac{u_i^\top b}{\sigma_i} v_i$. The cost of computation is, thus, $O(nmr)$.

Another way to solve least squares is by padding A into the bigger system. Let us define $r = Ax - b$, then we have a different system: $A^*r = 0$, or in the block form:

$$\begin{pmatrix} 0 & A^* \\ A & -I \end{pmatrix} \begin{pmatrix} x \\ r \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix},$$

which is a $(n + m) \times (n + m)$ matrix with the same condition number.

13. Eigenvalue problem. Eigendecomposition. Criteria of eigendecomposition existence via algebraic and ge- ometric multiplicities (no proof). Example of a nondiagonalizable matrix. Computing eigenvalues using characteristic equation. Is it a good idea? Gershgorin circles theorem (with proof).

Eigenvalue problem

A vector $x \neq 0$ is called an eigenvector of a square matrix A if there exists a number λ such that

$$Ax = \lambda x.$$

The number λ is called an eigenvalue.

Eigendecomposition

If matrix A of size $n \times n$ has n eigenvectors s_i, $i = 1, \dots, n$:

$$As_i = \lambda_i s_i,$$

then this can be written as

$$AS = S\Lambda, \quad \text{where} \quad S = (s_1, \dots, s_n), \quad \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n),$$

or equivalently (multiply by S^{-1} from the right)

$$A = S\Lambda S^{-1}.$$

This is called an **eigendecomposition** of a matrix. Matrices than can be represented by their eigendecomposition are called **diagonalizable**.

Criteria of eigendecomposition existence via algebraic and geometric multiplicities (no proof)

Matrix is diagonalizable iff **algebraic multiplicity** of each eigenvalue (mutiplicity of eigenvalue in the characteristic polynomial) is equal to its **geometric multiplicity** (dimension of eigensubspace). In other words, if eigenvalue λ has algebraic multiplicity n. There should be n not collinear nonzero vectors x, such that $Ax = \lambda x$. This equality will also hold for any linear combination of them.

One of the important class of diagonalizable matrices are **normal matrices**:

$$AA^* = A^*A.$$

Example of a nondiagonalizable matrix

You can simply check that, e.g. matrix

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

has one eigenvalue 1 of multiplicity 2 (since its characteristic polynomial is $p(\lambda) = (1 - \lambda)^2$), but only one eigenvector $\begin{pmatrix} c \\ 0 \end{pmatrix}$ and hence the matrix is not diagonalizable.

Computing eigenvalues using characteristic equation

The eigenvalue problem has the form

$$Ax = \lambda x,$$

or

$$(A - \lambda I)x = 0,$$

therefore matrix $A - \lambda I$ has non-trivial kernel and should be singular.

If x is an eigenvector with corresponding eigenvalue λ.

$$(A - \lambda I)x = \lambda x - \lambda x = 0$$

so x is in the kernel of $(A - \lambda I)$.

That means, that the **determinant**

$$p(\lambda) = \det(A - \lambda I) = 0.$$

- The equation is called **characteristic equations** and is a polynomial of order n.
- The n-degree polynomial has n complex roots!

Finding eigenvalues via solving a characteristic equation is not a very good idea, since polynomial rootfinding is a very ill-conditioned problem, because monomials are almost linearly dependent.

Gershgorin circles theorem (with proof).

There is a very interesting theorem that sometimes helps to localize the eigenvalues.

It is called **Gershgorin theorem**.

It states that all eigenvalues $\lambda_i, i = 1, \dots, n$ are located inside the union of **Gershgorin circles** C_i, where C_i is a disk on the complex plane with center a_{ii} and radius

$$r_i = \sum_{j \neq i} |a_{ij}|.$$

Moreover, if the circles do not intersect they contain only one eigenvalue per circle.

Proof

First, we need to show that if the matrix A is **strictly diagonally dominant**, i.e.

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|,$$

then such matrix is non-singular.

We separate the diagonal part and off-diagonal part, and get

$$A = D + S = D(I + D^{-1}S),$$

and $\|D^{-1}S\|_1 < 1$. Therefore, by using the **Neumann series**, the matrix $I + D^{-1}S$ is invertible and hence A is invertible.

Now the proof follows by contradiction:

if any of the eigenvalues lies outside all of the circles, the matrix $(A - \lambda I)$ is strictly diagonally dominant ($|a_{ii} - \lambda| > \sum_{j \neq i} |a_{ij}|$)

- thus it is invertible
- that means, that $(A - \lambda I)x = 0$ means $x = 0$.
- so λ is not an eigenvalue

14. Power method and its convergence (with bound derivation). Application in PageRank.

Power method is the simplest method for the **computation of the largest eigenvalue in modulus**. The eigenvalue problem

$$Ax = \lambda x, \quad \|x\|_2 = 1 \text{ for stability.}$$

can be rewritten as a **fixed-point iteration**. This iteration is called **power method** and finds the largest in modulus eigenvalue of A. Power method has the form

$$x_{k+1} = Ax_k, \quad x_{k+1} := \frac{x_{k+1}}{\|x_{k+1}\|_2}$$

and

$$x_{k+1} \to v_1,$$

where $Av_1 = \lambda_1 v_1$ and λ_1 is the largest eigenvalue and v_1 is the corresponding eigenvector. On the $(k+1)$-th iteration approximation to λ_1 can be found as

$$\lambda^{(k+1)} = (Ax_{k+1}, x_{k+1}),$$

Note that $\lambda^{(k+1)}$ is not required for the $(k+2)$-th iteration, but might be useful to measure error on each iteration: $\|Ax_{k+1} - \lambda^{(k+1)} x_{k+1}\|$. The convergence is geometric, but the convergence ratio is q^k, where $q = \left|\frac{\lambda_2}{\lambda_1}\right| < 1$, for $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n$ and k is the number of iteration. It means, the convergence can be artitrary small. To prove it, it is sufficient to consider a 2×2 diagonal matrix.

Things to remember about the power method

- Power method gives estimate of the largest eigenvalue in modulus or spectral radius of the given matrix
- One step requires one matrix-by-vector product. If the matrix allows for an $\mathcal{O}(n)$ matvec (for example, it is sparse), then power method is tractable for larger n.
- Convergence can be slow
- If only a rough estimate is needed, only a few iterations are sufficient
- The solution vector is in the Krylov subspace $\{x_0, Ax_0, \ldots, A^k x_0\}$ and has the form $\mu A^k x_0$, where μ is the normalization constant.

Application in PageRank. Google pagerank is one of the most famous eigenvectors computation. All we know about the web is which page refers to which. PageRank is defined by a recursive definition. Denote the importance of the i-th page by p_i. Then we define this importance as an average value of all importance of all pages that refer to the current page. It gives us a linear system

$$p_i = \sum_{j \in N(i)} \frac{p_j}{L(j)},$$

where $L(j)$ is the number of outgoing links on the j-th page, $N(i)$ are all the neighbours of the i-th page. It can be rewritten as

$$p = Gp, \quad G_{ij} = \frac{1}{L(j)}$$

which is an eigenvalue problem. The maximum eigenvalue for G is 1, thus, the power method explained above is useful in finding the importance of web pages.

15. Schur decomposition (with proof). Normal matrices and their diagonalizability. Properties of eigenvalues of Hermitian, unitary and skew-Hermitian matrices. Variational concept for eigenvalues: Rayleigh quotient.

Definitions:

- Schur decomposition: any square matrix A can be expressed as $A = UTU^*$, where U is unitary and T is upper triangular.
- Normal matrix: matrix A is normal, iff $A^*A = AA^*$.
- Rayleigh quotient: $R(A, x) \equiv R_A(x) = \frac{(x,Ax)}{(x,x)} = \frac{x^*Ax}{x^*x}$

Proof of Schur decomposition

Let q_1 be some eigenvector of$A$ of unit length: $Aq_1 = \lambda q_1, (q_1, q_1) = 1$. Then we can create a matrix $U_1 = (q_1|Q'_1)$ such that q is orthogonal to columns of Q'_1. Then

$$Q^*AQ = (q_1|Q_1)^*A(q_1|Q_1) = \left[\begin{array}{c|c} q_1^*Aq_1 & q_1^*AQ_1 \\ \hline Q_1^*Aq_1 & Q_1^*AQ_1 \end{array}\right] =$$

$$= \left[\begin{array}{c|c} \lambda q_1^*q_1 & q_1^*AQ_1 \\ \hline \lambda Q_1^*q_1 & Q_1^*AQ_1 \end{array}\right] = \left[\begin{array}{c|c} \lambda & q_1^*AQ_1 \\ \hline 0 & Q_1^*AQ_1 \end{array}\right]$$

Now we can continue this procedure by taking an eigenvector q_2 of the matrix $Q_1^*AQ_1$, forming the matrix $U_2 = \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & (q_2|Q_2) \end{array}\right]$ etc. Repeating this procedure n times, we obtain an upper triangular matrix T:

$$U_n^* ... U_1^* A U_1 ... U_n = T.$$

Therefore,

$$A = UTU^*, \text{ where } U = U_1 ... U_n,$$

and U is unitary as a product of unitary matrices.

Normal matrix is unitarily similar to a diagonal matrix - If A is a normal matrix, then it's Schur decomposition takes form $A = U^*DU$, such that U is unitary, and D is a diagonal matrix with eigenvalues of A on diagonal.

Proof

Let $A = U^*TU$ be Schur decomposition of A. Then

$$AA^* = A^*A \implies$$

$$AA^* = U^*TUU^*T^*U = U^*TT^*U = \\ = U^*T^*TU = U^*T^*UU^*TU = A^*A \implies T^*T = TT^* \quad (1)$$

Since we know that T is upper triangular, we this implies that T is diagonal (this can be noticed if we write down the products TT^* and T^*T explicitly.

Eigenvalue properties - If A is unitary, then all its eigenvalues satisfy $|\lambda|= 1$ - If A is hermitian, then all its eigenvalues are real - If A is skew-hermitian, then all its eigenvalues are purely imaginary

Proof

- Hermitian matrix

If the matrix A is Hermitian,i.e. $A = A^*$, and $Av = \lambda v$, then we can take a conjugate transpose:

$$v^*A^* = \lambda^*v^*.$$

Therefore, if $v \neq 0$

$$\lambda^*\|v\|_2^2 = \lambda^*v^*v = v^*A^*v = v^*Av = v^*\lambda v = \lambda\|v\|_2^2 \implies \lambda^* = \lambda$$

- Unitary matrix

If the matrix A is unitary, then $A^*A = I$. Let $Av = \lambda v \implies v^*A^* = \lambda^* v^*$. Then

$$\|v\|^2 = v^*v = v^*Iv = v^*A^*Av = \lambda^* v^* v \lambda = |\lambda|^2 \|v\|^2 \implies |\lambda|^2 = 1$$

- Skew-hermitian matrix

If the matrix A is skew-hermitian,i.e. $A^* = -A$, and $Av = \lambda v$, then we can take a conjugate transpose:

$$v^*A^* = \lambda^* v^*.$$

Therefore, if $v \neq 0$

$$\lambda^* \|v\|_2^2 = \lambda^* v^* v = v^* A^* v = -v^* Av = -v^* \lambda v = -\lambda \|v\|_2^2 \implies \lambda^* = -\lambda \implies \lambda \in i\mathbb{R}$$

Rayleigh quotient and eigenvalues

If A is an $n \times n$ hermitian matrix with eigenvalues $\lambda_1 \leq ... \leq \lambda_n$, then

$$\lambda_1 \leq R_A(x) \leq \lambda_n$$

16. QR decomposition. Proof of its existence. Connection with the Gram-Schmidt algorithm. QR via Cholesky decomposition.

QR decomposition of the matrix A is a such decomposition, that Q is an (n,m) ($n \geq m$)column orthogonal (column unitary) matrix ($Q^*Q = I$), and R is a (m,m) upper triangular square matrix. It is defined for any rectangular (n,m) matrix.

Uniqueness.

QR decomposition is unique if we require the diagonal elements of R to be positive. Otherwise, if A=QR is a QR decomposition, take S - a diagonal matrix with elements on the main diagonal equal to +-1. Q'=QS, R'=SR, then A=Q'R' is also a QR decomposition.

Applications.

QR decomposition is used as a preprocessing step for the SVD. If A is a rectangular matrix with a high dimension of rows and relatively small amount of columns, we can compute its QR decomposition and find the SVD for the resulting matrix R. Reduces the problem of finding the SVD of a rectangular matrix to finding an SVD of a square matrix (reduction in complexity).

QR decomposition is used in an algorithm for finding eigenvalues (change iteratively $A_i = Q_i R_i$ into $A_{i+1} = R_i Q_i$ and find the QR decomposition of A_{i+1}.

A fast and stable way to solve overdetermined linear systems. Typically QR decomposition is the method of choice.

QR decomposition is a canonical way to solve the linear least squares problem. The solution of the linear least squares problem is $x = A^\dagger b$. If A has a full column rank, we compute the QR decomposition to express:

$$x = A^\dagger b = (A^*A)^{-1}A^*b = ((QR)^*(QR))^{-1}(QR)^*b = (R^*Q^*QR)^{-1}R^*Q^*b = R^{-1}Q^*b.$$

QR decomposition is a more stable way of solving this problem than by using the pseudoinverse matrix explicitly.

Connection with the Gram-Schmidt algorithm.

Gram-Schmidt process is a process of orthogonalizing (orthonormalizing) a set of linearly independent vectors. Take $(v_1, v_2, \dots v_n)$ linearly independent vectors, and iteratively calculate the orthogonal basis $(u_1, \dots u_n)$ using this procedure: $u_i = v_i - \sum_{j=1}^{i-1} \frac{(u_j, v_i)}{(u_j, u_j} u_j$. $e_i = \frac{u_i}{norm(u_i)}$ to make it orthonormal.

Application of the Gram–Schmidt process to the column vectors of a full column rank matrix gives the QR decomposition. Matrix Q is made up from the resulting column vectors.

Existence.

A QR decomposition exists for any rectangular (n,m) matrix.

Proof using Cholesky factorization.

Suppose A has full column rank. Then A^*A is a Hermitian, positive definite matrix since $(A^*A)^* = A^*A$, $(A^*Ax, x) = (Ax, Ax) = ||Ax||^2 > 0 \ \forall x \neq 0$.

If $||Ax||^2 = 0$, then $Ax = 0$, and it means columns of A are linearly dependent with coefficients x_j, not all of which are zero.

For a Hermitian positive-definite matrix A^*A there exists its Cholesky factorization, $A^*A = LL^*$, where L is a lower triangular matrix with real and positive diagonal entries -> L is invertible. Denote $R = L^*$, then $A^*A = R^*R$, R is an upper triangular matrix (ofc R is also invertible). Then consider AR^{-1}. It exists since R is invertible. Show that AR^{-1} is unitary.

$$(AR^{-1})^*AR^{-1} = R^{-1*}A^*AR^{-1} = R^{-1*}R^*RR^{-1} = I$$

Then denote $Q = AR^{-1},\ A = QR$, q.e.d

If A does not have full column rank, then for any rank-deficient matrix A there exists a sequence of full-column rank matrices A_k that converges to A. $A_k \to A$ Decompose them. $A_k = Q_k R_k$.

The set of unitary matrices is compact since is is bounded and the all its limit point matrices are also unitary. Since it is compact, from our Q_i we can find a converging sequence $Q_{n_k} \to Q$. $R_{n_k} = Q^*_{n_k} A_{n_k}$, then since both Q_{n_k} and A_{n_k} converge, then R_{n_k} also converges, and since R_{n_k} are all upper triangular, R is also upper triangular.

Existence of QR decomposition can be also proved by using Gram-Schmidt algorithm and also Householder reflections (17).

17. QR via Householder reflections and Givens rotations (both with derivation). Comparison of these algorithms.

Householder method:

- $H = I - 2vv^*$

  1. Let $A = [v_1, v_2, ...v_n]$
  2. At step i compute $H(v_i')$, where $v_i' = v_i[n-i:n]$
  3. Fill the result with ones on diagonal, pad it with zeros to match the size of A, call it H_i.
  4. Compute H_iA.
  5. If $i! = n$ return to step 2.
  6. $H_nH_{n-1}...H_1 = Q^*$ and $Q^*A = R$

- In practice, better use $H = I - 2UU^*$, which is a block version of Householder.

Givens method:

- Given a vector $x = [x_1, x_2, ...x_n]^T$ we can use a product of zero rotations to zero out $n-i$ elements of x below entry x_i. To zero out x_{x+1}, compute $G(i, i+1, c_{i+1}, s_{i+1})x$.

- Consequently, the product $G(i, n, c_n, s_n)...G(i, i+2, c_{i+2}, s_{i+2})G(i, i+1, c_{i+1}, s_{i+1})x$ transforms x into form $[x_1, x_2, ...x_i, 0...0]^T$.

- The algorithm:

  1. Take a_{nn}. Calculate $\theta = \arctan \frac{a_{nn}}{a_{n-1n}}$.
  2. Calculate $c = -\cos\theta$ and $s = \sin\theta$
  3. Construct $G_i = G(a_{nn}, a_{n-1n}, s, c)$ (see question 5 on how).
  4. Calculate G_iA
  5. Repeat for every element below the main diagonal
  6. The consequent product of all G's at each step is $Q^* and Q^*A = R$

Comparison between algorithms:

- Householder algorithm is best for sequential architectures and dense matrices. $2mn^2 - (2/3)n^3$ flops.

- Givens algorithm is best for parallel architectures and sparse matrices, $3mn^2 - n^3$ flops.

18. QR algorithm. Convergence theorem (no proof). Accelerating convergence with shifts. Reducing complexity of QR algorithm from $O(n^4)$ down to $O(n^3)$

QR algorithm:

- Let A be a matrix of which we want to compute the eigenvalues,
- 1. $A_k = Q_k R_k$, Q_r is unitary R_k is upper triangular
  2. $A_{k+1} = R_k Q_k = Q_k^* A_k Q_k$. A_{k+1} is unitary similar to A_k, therefore this operation preserves eigenvalues.
  3. Repeat step 1 for given number of iterations.
  4. The approximate eigenvalues of A_n are situated on the main diagonal. The more iterations the better the precision of those.

Convergence of QR algorithm:

- If we have a decomposition: $A = X\Lambda X^{-1}$, $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$and $\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$ and there is a gap between Λ_1 and Λ_2, then the block A_{21} converges to zero with speed: $||A_{21}^{(k)}|| \leq Cq^k$, where $q = \frac{\lambda_{n+1}}{\lambda_m}$, where m is size of vector Λ_1.
- In other words, the magnitude gap between eigenvalues reduces the convergence speed of algorithm.

Accelerating convergence with shifts:

- $A_k = s_k I = Q_k R_k$, $A_{k+1} = R_k Q_k + s_k I$
- Convergence of this algorithm is: $|\frac{\lambda_{s+1} - s_k}{\lambda_m - s_k}|$, where λ_m is the spectral radius of A.
- The closer s_k is to eigenvector, the faster the convergence.

Reducing complexity of QR algorithm:

- It is faster to compute QR decomposition for matrices of Hessenberg (almost triangular) form.Therefore, before the iteration starts, we need to perform unitary similar reduction of the matrix to the Hessenberg form (this should preserve the eigenvalues). Applying Householder reflections we can reduce any matrix to the Hessenberg form: $U^*AU = H$.

  Note, tha the QR algorithm preserves the upper Hessenberg form:

  $$A_k = Q_k R_k \qquad A_{k+1} = R_k Q_k \Rightarrow A_{k+1} = R_k H_k R_k^{-1}$$

  Multiplication by upper triangular matrices cannot add nonzero elements below the diagonal.
- Now, given A_0 is in Hessenberg form, each iteration of the algorithm will const us $O(n^2)$.

19. Divide and conquer algorithm for symmetric eigenvalue problems (with derivation).

By using two-sided Householder transformation, any matrix can be reduced to a tridiagonal form. If we have a tridiagonal matrix T, let us split it into blocks:

$$T = \begin{bmatrix} T_1' & B \\ B^\top & T_2' \end{bmatrix}$$

We can write the matrix T as

$$T = \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m vv^*$$

Where $v = (0, \dots, 0, 1, 1, 0, \dots, 0)^T$.

Note that now T is almost a block-tridiagonal matrix, except for two elements, which is corrected by a rank 1 correction term $b_m vv^*$. Now, when T_1, T_2 are lower in dimension than T, suppose that we have already computed their decompositions: $T_1 = Q_1 \Lambda_1 Q_1^*$, $T_2 = Q_2 \Lambda_2 Q_2^*$

$$\begin{bmatrix} Q_1^* & 0 \\ 0 & Q_2^* \end{bmatrix} T \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} = D + \rho uu^*, \quad D = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}$$

Now the problem is reduced to finding the eigenvalues of a diagonal matrix + one-rank correction term.

$$det(D + puu^* - \lambda I) = det((D - \lambda I)(I + (D - \lambda I)^{-1} puu^*)) = det(D - \lambda I) det(I + (D - \lambda I)^{-1} puu^*)$$

$det(D - \lambda I)$ is just a product of the diagonal elements. Note that $(D - \lambda I)^{-1}$ is also a diagonal matrix. Denote $w = p(D - \lambda I)^{-1} u$.Now we have to find the determinant $det(I + wu^*)$, $C = I + wu^*$. Suppose x is an eigenvector, then $Cx = x + w(u^* x)$. So for all $x: \ (u^* x) = 0$, and the dimensionality of such a subspace is n-1 - 1 is an eigenvalue with multiplicity n-1. Find the remaining eigenvalue. The sum of the eigenvalues is equal to the trace of the matrix (use Jordan decomposition and cyclic property of trace to prove it). Trace is simply n+(w,u) here. Then

$$\det\bigl(I + \rho(D - \lambda I)^{-1} uu^*\bigr) = 1 + \rho \sum_{i=1}^{n} \frac{|u_i|^2}{d_i - \lambda} = 0$$

To compute it, $O(n^2)$ complexity is required. $O(nlogn)$ if the Fast Multipole Method is implemented.

This algorithm has been abandoned for a long time because of instability of computation of the eigenvectors (close eigenvalues lead to collinear vectors that have to be orthogonal). It was solved with the Loewner theorem, which for the case of $d_n < \alpha_n < \dots < d_{i+1} < \alpha_{i+1} \dots$ allows to change the matrix from D to $\widehat{D}$ with preservation of the set of the eigenvalues. $\widehat{D} = D + \widehat{u}\widehat{u}^*$. So now it is possible to compute the eigenvectors (after computing the eigenvalues first).

20. Jacobi method for eigenvalue problem. Its convergence (with proof).

Jacobi method

Recall, that Jacobi (Givens) rotation correspond to the 2×2 orthogonal matrix of the form:

$$G_{ij} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}$$

Where two variables i, j for row and column are selected.

Idea

Idea of the Jacobi method is to minimize the sum of off-diagonal elements by applying succesive Jacobi rotations U to zero off-diagonal elements:

$$\Gamma(A) = \mathrm{off}(U^* AU) \quad \mathrm{off}(X) = \sum_{i \neq j} |X_{ij}|^2 = \|X\|_F^2 - \sum_{i=1}^{n} x_{ii}^2$$

- When the **pivot** is chosen it is easy to eliminate it
- The question is the order in which to make the **sweeps** to make
- In case of the elimination of the largest off-diagonal elements each time the convergence is quadratic (but searching for the largest element on each iteration is prohibitevely expensive)
- So, in practice the cyclic order $(1, 2), (1, 3). \dots, (2, 3), \dots$ is used

Convergence

To show convergence, we firstly show that

$$\mathrm{off}^2(B) < \mathrm{off}^2(A)$$

For $B = U^* AU$. Note, that the Frobenius norm is preserved by unitarity transfromation. Let us apply Jacobi rotation for indices p and q.

$$\Gamma^2(A) = \mathrm{off}^2(B) = \|B\|_F^2 - \sum_{i=1}^{n} b_{ii}^2 = \|A\|_F^2 - \sum_{i \neq p,q} b_{ii}^2 - (b_{pp}^2 + b_{qq}^2) =$$

$$= \|A\|_F^2 - \sum_{i \neq p,q} a_{ii}^2 - (a_{pp}^2 + 2a_{pq}^2 + a_{qq}^2) = \|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2 - 2a_{pq}^2 = \mathrm{off}^2(A) - 2a_{pq}^2 < \mathrm{off}^2(A)$$

- Number of off-diagonal elements decreases after every Jacobi rotation
- In case the largest element is selected $|a_{pq}| = \gamma$, then :

  $$\mathrm{off}^2(A) \leq 2N\gamma^2$$

  Where $2N = n(n-1)$ is number of off-diagonal elements. Or rewrite it in the form:

  $$2\gamma^2 \geq \frac{\mathrm{off}^2(A)}{N}$$

  Therefore:

  $$\Gamma^2(A) = \mathrm{off}^2(A) - 2\gamma^2 \leq \mathrm{off}^2(A) - \frac{\mathrm{off}^2(A)}{N} \Rightarrow \Gamma(A) \leq \sqrt{1 - \frac{1}{N}} \mathrm{off}(A)$$

  After N steps, one has the factor:

  $$\Gamma(A) \leq \sqrt{1 - \frac{1}{N}}^{N} \leq (1 - \frac{1}{N})^{N/2} \simeq e^{-1/2}$$

  I.e a linear convergence (in fact locally quadratic - no proof here)

Summary

- Large constant (for the convergence rate)
- Very accurate (high relative error for small eigenvalues)
- Good parallel capabilities

21. Sparse matrix arithmectics: COO, LIL, CSR, CSC formats. Their comparison. Blocking and cache oblivious algorithms for increasing efficiency of sparse formats.

Let nnz be the number of nonzero elements in the matrix. Let the size of the compressed matrix be $n \times m$.

In COO (coordinate) format, we store a list of (row, column, value) tuples, $3 \times \text{nnz}$ elements in total, nnz of which are floats.

In LIL (list of lists) format, we store one list per row, with each entry in the list being the column index and the corresponding value, hence, $2 \times \text{nnz}$ elements, nnz of which are floats. Typically, these lists are sorted by the column index.

In CSR (compressed sparse row) format, we store three lists: ia, ja and sa. The lists ja and sa are of length that is equal to the number of nonzero elements in the matrix and contain the non-zero values and the column indices of those values respectively. The row index ia is of length $n+1$ and encodes the index in ja and sa where the given row starts. Its last element is nnz. For example, the matrix

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{pmatrix}$$

is represented in the CSR format as follows:

$$\text{ja} = (0,1,2,1) \quad \text{sa} = (5,8,3,6) \quad \text{ia} = (0,1,2,3,4)$$

.

The CSC (compressed sparse column) format is very similar to the CSR format. The difference is that in CSC, the roles of ia and ja are reversed. So, ja's elements now point to the start indices of new columns and ia contains the rows of the non-zero elements of the matrix.

The LIL format is the most efficient one in terms of memory consumption, the CSR and CSC formats are slightly less efficient (with an overhead of $n+1$ and $m+1$ integer values respectively. The COO format is the least efficient with an overhead of nnz integers.

These formats can be divided into two groups: those that support efficient modification and those that support efficient access and matrix operations (e.g. matvecs in $O(\text{nnz})$). The COO and LIL formats belong to the first group, while CSR and CSC belong to the second group. The drawbacks are obvious: COO and LIL formats are not suited for linear algebra, and CSR and CSC are bad for constructing matrices. Speaking of other advantages of LIL over COO, it supports slicing (even though column slicing is slow). The COO format's advantage over LIL is that it supports fast conversion to and from CSR/CSC (and other sparse) formats.

To improve the speed of matrix operations when using sparse matrix formats, one might utilize cache structure. For this, when matrix is in CSR form, ja needs to store store sequential elements. This way, when matvec is performed, the sequential elements of the vector will be moved to cache and number of cache misses will be reduced. In order to achieve this, one can, using matrices P and Q, permute rows and columns of the matrix A to obtain a matrix $A_1 = PAQ$, which has a separated block-diagonal form that is cache-oblivious.

Another way to speed up the computation is to use CSB (compressed sparse block) format, which stores block indices and indices of data inside each block. This format is much more suitable for parallel implementations, since blocks can be multiplied by their parts of a vector in parallel.

22. Sparse LU decomposition, connection with graphs. Dependence of fill-in on ordering of graph nodes (with example). Nested dissection and spectral bisection algorithms.

23. Richardson iteration. Optimal choice of parameter. Convergence estimate.

Richardson iteration

Richardson iteration is an iterative method for solving a system of linear equations. The simplest idea is the **"simple iteration method"** or **"Richardson iteration"**.

$$\begin{gathered} Ax = f \\ \tau(Ax - f) = 0 \\ x - \tau(Ax - f) = x \\ x_{k+1} = x_k - \tau\left(Ax_k - f\right) \end{gathered}$$

where τ is the **iteration parameter**, which can be always chosen such that the method **converges** (for the case of symmetric positive definite matrix).

Connection to ODEs

- The Richardson iteration has a deep connection to the Ordinary Differential Equations (ODE).
- Consider a time-dependent problem $A = A^* > 0$

$$\frac{dy}{dt} + Ay = f, \quad y(0) = y_0$$

- Then $y(t) \to A^{-1}f$ as $t \to \infty$, and the **Euler scheme** reads

$$\frac{y_{k+1} - y_k}{\tau} = -Ay_k + f$$

which leads to the Richardson iteration

$$y_{k+1} = y_k - \tau\left(Ay_k - f\right)$$

Convergence

We have $x_{k+1} = x_k - \tau\left(Ax_k - f\right)$

Substracting exact solution $x.$ from both sides:

$$e_{k+1} = x_{k+1} - x_* = x_k - x_* - \tau\left(A\left(x_k - x_* + x_*\right) - f\right) = e_k - \tau\left(Ae_k + Ax_* - f\right) = e_k - \tau Ae_k$$

- Let $x,$ be the solution; introduce an error $e_i = x_k - x_*$, then

$$e_{k+1} = (I - \tau A)e_k$$

therefore if $\|I - \tau A\| < 1$ in any norm, the iteration converges.

- For symmetric positive definite case it is always possible to select τ such that the method converges.

Optimal parameter choice (proof is below)

A is symmetric positive definite.

- The choice of r that minimizes $\|I - \tau A\|_2$ for $A = A^* > 0$ is (prove it!)

$$r_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\text{man}}}$$

where $\lambda_{\min}$ is the minimal eigenvalue, and $\lambda_{\max}$ is the maximal eigenvalue of the matrix A.

- So, to find optimal parameter, we need to know the **bounds of the spectrum** of the matrix A, and we can compute it by using **power method**.

  Notations:

  - $R_a = (I - \tau A)$
  - $\rho(A) = \max(|\lambda_1|, \ldots |\lambda_n|)$ is a spectral radius.

  Theorem. Assume that P is a nonsingular matrix and that P^{-1} A has positive real eigenvalues, ordered in such a way that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$. Then, the stationary Richardson method is convergent iff $0 < \alpha < 2/\lambda_1$. Moreover, letting

  $$\alpha_{opt} = \frac{2}{\lambda_1 + \lambda_n}$$

  the spectral radius of the iteration matrix $\mathrm{R_a}$ is minimum if $\alpha = \alpha_{\mathrm{opt}}$, with

  $$\rho_{opt} = \min_{\alpha}\left[\rho\left(\mathrm{R}_\alpha\right)\right] = \frac{\lambda_1 - \lambda_n}{\lambda_1 + \lambda_n}$$

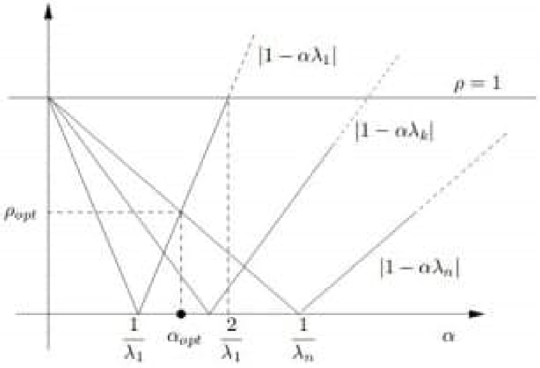


Figure 1: r

Proof. The eigenvalues of R_α are given by $\lambda_i\left(\mathrm{R}_\alpha\right) = 1 - \alpha\lambda_i$, so that **Richardson method** is convergent iff $\left|\lambda_i\left(\mathrm{R}_\alpha\right)\right| < 1$ for $i = 1, \ldots, n$, that is, if $0 < \alpha < 2/\lambda_1$. It follows (Figure) that $\rho\left(\mathrm{R}_\alpha\right)$ is minimum when $1 - \alpha\lambda_n = \alpha\lambda_1 - 1$, that is, for $\alpha = 2/(\lambda_1 + \lambda_n)$, which furnishes the desired value for α_{opt}. By substitution, the desired value of ρ_{opt} is obtained.

Condition number and convergence speed
Even with the optimal parameter choice, the error at the next step satisfies

$$\|e_{k+1}\|_2 \leq q\,\|e_k\|_2\,, \quad \rightarrow \quad \|e_k\|_2 \leq q^k\,\|e_0\|_2$$

where

$$\begin{array}{l} q = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} = \frac{\operatorname{cond}(A)-1}{\operatorname{cond}(A)+1} \\ \operatorname{cond}(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \quad \text{for} \quad A = A^* > 0 \end{array}$$

is the condition number of A.

24. Chebyshev iteration. Permutations of roots for computing parameters. Error bounds via Chebyshev polynomials.

Chebyshev iteration

Suppose we change τ at **every step**:

$$x_{k+1} = x_k - \tau_k \left(Ax_k - f\right)$$

Consider:

$$e_{k+1} = \left(I - \tau_k A\right) e_k = \left(I - \tau_k A\right) \left(I - \tau_{k-1} A\right) e_{k-1} = \ldots = p(A) e_0$$

where $p(A)$ is a **matrix polynomial** (simplest matrix function)

$$p(A) = \left(I - \tau_k A\right) \ldots \left(I - \tau_0 A\right)$$

and $p(0) = 1$
The error is written as

$$e_{k+1} = p(A) e_0$$

and hence

$$\|e_{k+1}\| \leq \|p(A)\| \|e_0\|_*$$

where $p(0) = 1$ and $p(A)$ is a **matrix polynomial**.
To get better **error reduction**, we need to minimize

$$\|p(A)\|$$

over all possible polynomials $p(x)$ of degree $k+1$ such that $p(0) = 1$. We will use $\|\cdot\|_2$.

Polynomials least deviating from zeros

Important special case: $A = A^* > 0$.
Then $A = U \Lambda U^*$
and

$$\|p(A)\|_2 = \left\|U p(\Lambda) U^*\right\|_2 = \|p(\Lambda)\|_2 = \max_i \left|p\left(\lambda_i\right)\right| \leq \max_{\lambda_{min} \leq \lambda \leq \lambda_{max}} |p(\lambda)|$$

The latter inequality is the only approximation. Here we make a **crucial assumption** that we do not want to benefit from distribution of spectra between $\lambda_{\min}$ and $\lambda_{\max}$

Thus, we need to find a polynomial such that $p(0) = 1$, that has the least possible deviation from 0 on $[\lambda_{\min}, \lambda_{\max}]$.

We can do the affine transformation of the interval $[\lambda_{\min}, \lambda_{\max}]$ to the interval $[-1, 1]$:

$$\zeta = \frac{\lambda_{\max} + \lambda_{\min} - \left(\lambda_{\min} - \lambda_{\max}\right) x}{2}, \quad x \in [-1, 1]$$

The problem is then reduced to the problem of finding the **polynomial least deviating from zero** on an interval $[-1, 1]$.

Exact solution: Chebyshev polynomials

The exact solution to this problem is given by the famous **Chebyshev polynomials** of the form

$$T_n(x) = \cos(n \arccos x)$$

What do you need to know about Chebyshev polynomials

1. This is a polynomial!

2. We can express T_n from T_{n-1} and T_{n-2} :

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad T_0(x) = 1, \quad T_1(x) = x$$

1. $|T_n(x)| \leq 1$ on $x \in [-1, 1]$

2. It has $(n + 1)$ **alternation points**, where the maximal absolute value is achieved (this is the sufficient and necessary condition for the **optimality**) (Chebyshev alternance theorem, no proof here).

3. The **roots are just**

$$n \arccos x_k = \frac{\pi}{2} + \pi k, \quad \rightarrow \quad x_k = \cos \frac{\pi(2k+1)}{2n}, k = 0, \ldots, n-1$$

Convergence of the Chebyshev-accelerated Richardson iteration

Note that $p(x) = (1 - \tau_n x) \ldots (1 - \tau_0 x)$, hence roots of $p(x)$ are $1/\tau_i$ and that we additionally need to map back from [-1,1] to $[\lambda_{\min}, \lambda_{\max}]$. This results into

$$\tau_i = \frac{2}{\lambda_{\max} + \lambda_{\min} - (\lambda_{\max} - \lambda_{\min}) x_i}, \quad x_i = \cos \frac{\pi(2i+1)}{2n} \quad i = 0, \ldots, n-1$$

The convergence (we only give the result without the proof) is now given by

$$e_{k+1} \leq Cq^k e_0, \quad q = \frac{\sqrt{\mathrm{cond}(A)} - 1}{\sqrt{\mathrm{cond}(A)} + 1}$$

which is better than in the Richardson iteration.

Permutations of roots for computing parameters

Permutation of roots of Chebyshev polynomial has crucial effect on convergence. There is a solution for the problem of ordering the parameters in a cyclical iterative method used for solving the equation $Au = f$, in such a way as to eliminate computational instability.

Initially, the roots are sorted from large to small eigenvalues, therefore acting in this order, one firstly increases the norm of the residual and in fact it leads to blow-up, the permutation arranges roots in a such way, that the large eigenvalues are interchanges with the small one, and the overall norm of the $I - \tau_k A$ is approximately constant.

25. Solving linear system as an optimization problem: minimization of residual and energy functional. Approximation of a solution of a linear system by a subspace, Galerkin projection. Krylov subspace and ill-posedness of natural Krylov basis.

Solving $Ax = b$ Using the Krylov Subspace $\mathcal{K}_k$ How do we solve $Ax = b$, given only the information available from k steps of either the Arnoldi or the Lanczos algorithm? since the only vectors we know are the columns of Q_k, the only place to "look" for an approximate solution is in the Krylov subspace $\mathcal{K}_k$ spanned by these vectors. In other words, we see the "best" approximate solution of the form

$$x_k = \sum_{j=1}^{k} z_k q_k = Q_k \cdot z, \qquad \text{where} \qquad z = [z_1, \ldots, z_k]^T$$

Now we have to define "best."

There are several natural but different definitions, leading to different algorithms. We let $x = A^{-1}b$ denote the true solution and $r_k = b - Ax_k$ denote the residual.

1. The "best" x_k minimizes $\|x_k - x\|_2$. Unfortunately, we do not have enough information in our Krylov subspace to compute this x_k.

2. The "best" x_k minimizes $\|r_k\|_2$. This is implementable, and the corresponding algorithms are called MINRES (for minimum residual) when A is symmetric and GMRES (for generalized minimum residual) when A is nonsymmetric

3. The "best" x_k makes $r_k \perp \mathcal{K}_k$, i.e., $Q_k^T r_k = 0$. This is sometimes called the orthogonal residual property, or a Galerkin condition

4. When A is symmetric and positive definite, it defines a norm $\|r\|_{A-1}= \left(r^T A^{-1} r\right)^{1/2}$ (see Lemma 1.3). We say the "best" x_k minimizes $\|r_k\|_{A^{-1}}$ This norm is the same as $\|x_k - x\|_A$. The algorithm is called the conjugate gradient algorithm

When A is symmetric positive definite, the last two definitions of "best" also turn out to be equivalent.

(for detailed info go to Demmel page 300)

Note: Krylov subspace can be used when we assume that A is accessible only via a "black-box" subroutine that returns $y = Az$ given any z (and perhaps $y = A^T z$ if A is nonsymmetric). In other words, no direct access or manipulation of matrix entries is used. F.e A may not be represented explicitly as a matrix but may be available only as a subroutine for computing Ax

26. Ill-posedness of natural Krylov basis. Orthogonal basis in Krylov subspace. Arnoldi relation and its derivation. Lanczos process.

Ill-posedness.

Natural Krylov basis is set of vectors $\{A^i r_0\}_{i=0}^{n-1}$. $A^i r_0 \to \lambda^i_{\max} v$. v is the eigenvector, corresponding to the maximal eigenvalue of A, i.e. k_i become more and more collinear for large i.

Orthogonal basis.

To improve stability, we orthogonalize the vectors from the Krylov subspace:

$$K_j = \begin{bmatrix} r_0 & Ar_0 & A^2 r_0 & \dots & A^{j-1} r_0 \end{bmatrix} = Q_j R_j,$$

and the solution will be approximated as

$$x \approx x_0 + Q_j c.$$

Arnoldi relation.

$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top$$

where H_j is upper Hessenberg, and $Q_{j+1} = [q_0, \dots, q_j]$ has orthogonal columns that spans columns of K_{j+1}.

Derivation.

We proved the relation for K_3 , but the idea is the same for arbitrary j. IMO 3 main ideas we need to remember for the proof are 1. $Ak_j = k_{j+1}$, 2. $K_j = Q_j R_j$, 3. $h_{j,j-1} q_j = k_j - \sum_{s=0}^{j-1} \alpha_s k_s$

$$A \begin{bmatrix} k_0 & k_1 & k_2 \end{bmatrix} = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix} = \begin{bmatrix} k_0 & k_1 & k_2 \end{bmatrix} \begin{bmatrix} 0 & 0 & \alpha_0 \\ 1 & 0 & \alpha_1 \\ 0 & 1 & \alpha_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & k_3 - \alpha_0 k_0 - \alpha_1 k_1 - \alpha_2 k_2 \end{bmatrix},$$

Denote $\widehat{k}_3 = k_3 - \alpha_0 k_0 - \alpha_1 k_1 - \alpha_2 k_2$. In the matrix form,

$$AK_3 = K_3 Z + \widehat{k}_3 e_2^\top,$$

where Z is the **lower shift** matrix with the last column $(\alpha_0, \alpha_1, \alpha_2)^T$, and e_2 is the last column of the identity matrix.

Let $K_3 = Q_3 R_3$ be the QR-factorization. Then,

$$AQ_3 R_3 = Q_3 R_3 Z + \widehat{k}_3 e_2^\top,$$
$$AQ_3 = Q_3 R_3 Z R_3^{-1} + \widehat{k}_3 e_2^\top R_3^{-1}.$$

Note that $e_2^\top R_3^{-1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix} = \gamma e_2^\top$,

and $R_3 Z R_3^{-1} = \begin{bmatrix} * & * & * \\ * & * & * \\ 0 & * & * \end{bmatrix} = H_3$, – **upper Hessenberg matrix**.

So, we obtained $AQ_j = Q_j H_j + \gamma \widehat{k}_j e_{j-1}^\top$, and $\widehat{k}_j = k_j - \sum_{s=0}^{j-1} \alpha_s k_s$. Now we select α_s in such a way that $Q_j^* \widehat{k}_j = 0$. Then, $\widehat{k}_j = h_{j,j-1} q_j$, where q_j is the last column of Q_{j+1}, and the final formula is

$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top.$$

Lanczos process.

$Q_j^* A Q_j = H_j$. If $A = A^*$, then H_j – hermitian matrix, – becomes **tridiagonal**: $H_j = T_j$.

$$AQ_j = Q_jT_j + t_{j,j-1}q_je_{j-1}^\top \Rightarrow$$

$$t_{j,j-1}q_j = (AQ_j - Q_jT_j)e_{j-1} = Aq_{j-1} - t_{j-1,j-1}q_{j-1} - t_{j-2,j-1}q_{j-2}.$$

It means that

1. we only need to compute the last column of T_j (i.e. $\alpha_j = t_{j-1,j-1}$ and $\beta_j = t_{j-2,j-1}$), and we can do it using orthogonality constraints $(q_j, q_{j-1}) = 0, (q_j, q_{j-2}) = 0$.

2. we only need to store two vectors to get the new one.

27. Direct Lanczos method for solving linear systems. Conjugate gradient method.

From direct Lanczos method to the conjugate gradient

We can now get from the Lanczos recurrence to the famous **conjugate gradient** method. For $A = A^* > 0$ we have a three-term recursion relationship:

$$AQ_j = Q_j T_j + T_{j,j-1} q_j.$$

On the j-th step the solution is spanned by the columns of Q_c, so the current approximation has the following form :

$$x_j = x_0 + Q_j c_j$$

Where c_j are the coefficients in the current Krylov subspace $\mathcal{K}_j$. Projection of the original system to the $\mathcal{K}_j$ gives:

$$Q_j^* A(x_0 + Q_j c_j) = Q_j^* A x_0 + T_j c_j = Q_j^* f$$

Here note, that $Q_j q_j = 0$. Then subtraction of the rhs from the lhs gives:

$$T_j c_j = Q_j^*(f - Ax_0) = Q_j^* r_0$$

Since r_0 spans the first Krylov subspace, the first column of Q is simply $r_0/\|r_0\|$, therefore:

$$T_j c_j = \|e_0\|^2 e_0 = \gamma e_0$$

Derivation of the following update formulas is not required on the oral exam!

Key idea - LU factorization of T with the subsequent solution of the aforewritten system.

Short term recurrences for x_j, p_j, ξ_j:

$$\begin{aligned} x_j &= P_j z_j = P_{j-1} z_{j-1} + \xi_j p_j = x_{j-1} + \xi_j p_j \\ p_j &= \frac{1}{d_j}(q_j - b_j p_{j-1}) \\ \xi_j &= -c_j \xi_{j-1} \end{aligned}$$

Direct Lanczos method

We have the direct Lanczos method, where we store p_{j-1}, q_j, x_{j-1} to get a new estimate of x_j.

The main problem is with q_j: we have the three-term recurrence, but in the floating point arithmetic the orthogonality is can be lost, leading to numerical errors.

Conjugate gradient method

One has three sets of vectors:

- x_k - approximate solutions
- r_k - residuals
- p_k - *conjugate* gradients

p_k are called gradients, because the single step of CG is a choice of scalar p_k, such that the residual norm $\|\|$is minimized

Instead of q_j (last vector in the modified Gram-Schmidt process), it is more convenient to work with the **residual** :

$$r_j = f - Ax_j.$$

The resulting recurrency has the form:

$$\begin{aligned}x_j &= x_{j-1} + \alpha_{j-1} p_{j-1}\\ r_j &= r_{j-1} - \alpha_{j-1} A p_{j-1}\\ p_j &= r_j + \beta_j p_{j-1}\end{aligned}$$

Hence the name conjugate gradient: to the gradient r_j we add a **conjugate direction** p_j. We have **orthogonality** of residuals:

$$(r_i, r_j) = 0, \quad i \neq j$$

and **A-orthogonality** of conjugate directions:

$$(Ap_i, p_j) = 0$$

which can be checked from the definition. The equations for α_j and β_j can be now defined explicitly from these two properties:

$$\alpha_{j-1} = \frac{(r_{j-1}, r_{j-1})}{(Ap_{j-1}, r_{j-1})} = \frac{(r_{j-1}, r_{j-1})}{(Ap_{j-1}, p_{j-1} - \beta_{j-1} p_{j-2})} = \frac{(r_{j-1}, r_{j-1})}{(Ap_{j-1}, p_{j-1})}$$

$$\beta_{j-1} = \frac{(r_j, r_j)}{(r_{j-1}, r_{j-1})}$$

CG derivation overview

- Want to find x_* in Krylov subspace
- But natural basis is ill-conditioned, therefore we need orthogonalization
- Derive recurrent equation for sequential orthogonalization of the Krylov subspace basis
- Arnoldi process for non-symmetric matrix
- Lanczos process for symmetrix matrix
- Clever re-writing of these formulas gives short recurrence

Properties of the CG method

- We need to store 3 vectors
- Since it generates A-orthogonal sequence $p_1, \ldots, p_N$, after n steps it should stop (i.e., $p_{N+1} = 0$)

28. Convergence theory of the conjugate gradient method.

A-optimality

Energy functional can be written as:

$$(Ax, x) - 2(f, x) = (A(x - x_*), (x - x_*)) - (Ax_*, x_*),$$

where $Ax_* = f$. Up to a constant factor

$$(A(x - x_*), (x - x_*)) = \|x - x_*\|_A^2$$

is the **A-norm** of the error.

Convergence

The CG method computes x_k that minimizes the energy functional over the Krylov subspace, i.e. $x_k = p(A)f$, where p is a polynomial of degree $k+1$, so:

$$\|x_k - x_*\|_A = \inf_p \|\left(p(A) - A^{-1}\right) f\|_A.$$

Using eigendecomposition of A we have :

$$A = U\Lambda U^*, \quad g = U^* f,$$

and

$$\|x - x_*\|_A^2 = \inf_p \|\left(p(\Lambda) - \Lambda^{-1}\right) g\|_\Lambda^2 = \inf_p \sum_{i=1}^n \frac{(\lambda_i p(\lambda_i) - 1)^2 g_i^2}{\lambda_i} = \inf_{q, q(0)=1} \sum_{i=1}^n \frac{q^2(\lambda_i) g_i^2}{\lambda_i}$$

Selection of the optimal q depends on the eigenvalue distribution

Absolute and relative error

We have

$$\|x - x_*\|_A^2 \leq \sum_{i=1}^n \frac{g_i^2}{\lambda_i} \inf_{q, q(0)=1} \max_j q(\lambda_j)^2$$

The first term is just

$$\sum_{i=1}^n \frac{g_i^2}{\lambda_i} = (A^{-1}f, f) = \|x_*\|_A^2$$

And we have relative error bound:

$$\frac{\|x - x_*\|_A}{\|x_*\|_A} \leq \inf_{q, q(0)=1} \max_j |q(\lambda_j)|$$

so if matrix has only 2 different eigenvalues, then there exists a polynomial of degree 2 such that $q(\lambda_1) = q(\lambda_2) = 0$, so in this case CG converges in 2 iterations.

- If eigenvalues are clustered and there are l outliers, then after first $\mathcal{O}(l)$ iterations CG will converge as if there are no outliers (and hence the effective condition number is smaller)
- The intuition behind this fact is that after $\mathcal{O}(l)$ iterations the polynomial has degree more than l and thus is able to zero l outliers.

The last term is just the same as for the Chebyshev acceleration, thus the same upper convergence bound holds:

$$\frac{\|x_k - x_*\|_A}{\|x_*\|_A} \leq \gamma \left(\frac{\sqrt{\mathrm{cond}(A)} - 1}{\sqrt{\mathrm{cond}(A)} + 1} \right)^k$$

Finite termination & clusters

1. If A has only m different eigenvalues, CG converges in m iterations (proof in the blackboard).
2. If A has m c̈lustersöf eigenvalues, CG converges cluster-by-cluster.
3. As a result, better convergence than Chebyshev acceleration, but slightly higher cost per iteration.

29. MINRES method. GMRES method and its connection to Anderson acceleration. Disadvantages of GMRES for nonsymmetric systems. Idea of BiCG and BiCGStab.

GMRES

CG works for symmetric positive definite systems. What happens if A is non-symmetric? We can use GMRES (generalized minimal residual method). Recall Arnoldi relation

$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top .$$

where A matrix from equation (generates Krylov subspace), $Q_j = (q_0, \dots, q_{j-1})$ – matrix with columns representing orthogonal basis in Krylov subspace, H_j – upper Hessenberg matrix (it is Garelkin projection of A, i.e. $Q_j^* A Q_j = H_j$), $h_{j,j-1}$ – just a number, q_j – new basis vector and $e_{j-1} = (0, \dots, 0, 1)^\top$.

Let us rewrite the latter expression as

$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top = Q_{j+1} \widetilde{H}_j, \quad \widetilde{H}_j = \begin{bmatrix} h_{0,0} & h_{0,1} & \dots & h_{0,j-2} & h_{0,j-1} \\ h_{1,0} & h_{1,1} & \dots & h_{1,j-2} & h_{1,j-1} \\ 0 & h_{2,2} & \dots & h_{2,j-2} & h_{2,j-1} \\ 0 & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & & h_{j,j-1} & h_{j-1,j-1} \\ 0 & 0 & \dots & 0 & h_{j,j-1} \end{bmatrix}$$

Then, if we need to minimize the residual over the Krylov subspace, we have

$$x_j = x_0 + Q_j c_j$$

and x_j has to be selected as

$$\|Ax_j - f\|_2 = \|AQ_j c_j - r_0\|_2 \to \min_{c_j} .$$

Using the Arnoldi recursion, we have

$$\|Q_{j+1} \widetilde{H}_j c_j - r_0\|_2 \to \min_{c_j} .$$

Using the orthogonal invariance under multiplication by unitary matrix, we get

$$\|\widetilde{H}_j c_j - \gamma e_0\|_2 \to \min_{c_j},$$

where we have used that $Q_{j+1}^* r_0 = \gamma e_0$. ($r_0$ is the first column in orthogonal Q_j).

1. This is just a linear least squares with $(j+1)$ equations and j unknowns (j supposed to be significantly smaller than matrix size, so its meaningful reduction).
2. The matrix is also upper Hessenberg, thus its QR factorization can be computed in a very cheap way ($\mathcal{O}(j^2)$ operations using Givens rotations).

Summary:

1. Minimizes the residual directly (not the energy functional as CG)
2. Memory grows with the number of iterations n as $\mathcal{O}(nj)$, this is the main disadvantage, so **restarts** typically implemented (just start GMRES from the new initial guess, this heuristic may diverge).

Relation to Andreson acceleration

We can apply the GMRES-like idea to speed up the convergence of a given fixed-point iteration

$$x_{k+1} = \Phi(x_k).$$

This was actually older than the GMRES, and known as **Anderson Acceleration**.

Idea: **use history** for the update

$$x_{k+1} = \Phi(x_k) + \sum_{s=1}^{m} \alpha_s (x_{k-s} - \Phi(x_{k-s})),$$

and the parameters α_s are selected to minimize the norm of the residual

$$\min_{\alpha} \left\| \sum_{s=1}^{m} \alpha_s (x_{k-s} - \Phi(x_{k-s})) \right\|_2, \quad \sum_{s=1}^{m} \alpha_s = 1$$

Although, we similarity is not quite clear, in fact it can be show that Anderson acceleration for linear system is nothing else than GMRES.

MINRES

The MINRES method is GMRES applied to a symmetric system. We minimize

$$\|AQ_j c_j - f\|_2 = \|Q_j H_j c_j + h_{j,j-1} q_j e_{j-1} c_j - f\|_2 = \|Q_{j+1} \widehat{H}_{j+1} c_j - f\|_2 \to \min$$

which is equivalent to a linear least squares with an **almost tridiagonal** matrix (with additional row)

$$\|\widehat{H}_{j+1} c_j - \gamma e_0\|_2 \to \min .$$

- In a similar to CG fashion, we can derive short-term recurrences.
- A careful implementation of MINRES requires at most 5 vectors to be stored.

BICG (BiConjugate Gradients), general idea

As we just saw, GMRES requires storage of full basis ($\mathcal{O}(nj)$ memory), which can be too costly for for large systems. We would like to have a short-term recurrences for general case as for symmetric one, that is the motivation for BiCG and Bi-CGSTAB.

Idea of BiCG method is to use the normal equations:

$$A^* A x = A^* f,$$

and apply the CG method to it (as now it is symmetric semi-positive definite).

There are two options:

1. Use $\mathcal{K}(A^*A, A^*f)$ to generate the subspace. That leads to square of condition number, therefore slow convergence.
2. Instead, the idea of BiCG is to use two Krylov subspaces $\mathcal{K}(A)$ and $\mathcal{K}(A^*)$ to generate two basises that are **biorthogonal** (so-called biorthogonal Lanczos).

The goal is to compute the Petrov-Galerkin projection

$$W^* A V \widehat{x} = W^* f$$

with columns W from the Krylov subspace of A^*, V from A. So we project we system to the space spanned by columns of W and then trying to solve resulting equation in space spanned by columns of V

This method can be still quite unstable, it order to improve stability we can use GMRES steps after each BiCG step, this method is called Bi-CGSTAB.

What methods to use

1. If a matrix is symmetric (Hermitian) positive definite, use CG method.
2. If a matrix is symmetric but indefinite, we can use MINRES method (GMRES applied to a symmetric system)
3. If a matrix is non-symmetric and not very big, use GMRES
4. If a matrix is non-symmetric and we can store limited amount of vectors, use either: GMRES with restarts, or BiCGStab (the latter of the product with $A^\top$ is also available).

30. Preconditioning concept. Right and left preconditioners. Jacobi, Gauss-Seidel and SOR preconditioners.

Preconditioner: general concept

The general concept of the preconditioner is simple:
Given a linear system

$$Ax = f,$$

we want to find the matrix P_R and/or P_L such that

1. Condition number of AP_R^{-1} (right preconditioner) or $P_L^{-1}A$ (left preconditioner) or $P_L^{-1}AP_R^{-1}$ is better than for A
2. We can easily solve $P_L y = g$ or $P_R y = g$ for any g (otherwise we could choose e.g. $P_L = A$)

Then we solve for (right preconditioner)

$$AP_R^{-1}y = f \quad \Rightarrow \quad P_R x = y$$

or (left preconditioner)

$$P_L^{-1}Ax = P_L^{-1}f,$$

or even both

$$P_L^{-1}AP_R^{-1}y = P_L^{-1}f \quad \Rightarrow \quad P_R x = y.$$

One of the ideas is to use other iterative methods (beside Krylov) as preconditioners.

Jacobi method

Consider the matrix with non-zero diagonal. To get the **Jacobi method** you express the diagonal element:

$$a_{ii}x_i = -\sum_{i \neq j} a_{ij}x_j + f_i$$

and use this to iteratively update x_i:

$$x_i^{(k+1)} = -\frac{1}{a_{ii}}\left(\sum_{i \neq j} a_{ij}x_j^{(k)} + f_i\right),$$

or in the matrix form

$$x^{(k+1)} = D^{-1}\Big((D - A)x^{(k)} + f\Big)$$

where $D = \mathrm{diag}(A)$ and finally

$$x^{(k+1)} = x^{(k)} - D^{-1}(Ax^{(k)} - f).$$

So, Jacobi method is nothing, but simple Richardson iteration with $\tau = 1$ and left preconditioner $P = D$ - diagonal of a matrix. Therefore we will refer to $P = \mathrm{diag}(A)$ as **Jacobi preconditioner**. Note that it can be used for any other method like Chebyshev or Krylov-type methods.

Jacobi preconditioner:

1. Very easy to compute and apply
2. Works well for diagonally dominant matrices (remember the Gershgorin circle theorem!)

3. Useless if all diagonal entries are the same (proportional to the identity matrix)
4. Works better if one considers block diagonal matrix instead of diagonal (still easy to inverse if blocks are small)

Gauss-Seidel method

Another well-known method is Gauss-Seidel method.

Its canonical form is very similar to the Jacobi method, with a small difference. When we update x_i as

$$x_i^{(k+1)} := -\frac{1}{a_{ii}} \left(\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} - f_i \right)$$

we **use it in the later updates**. In the Jacobi method we use the full vector from the previous iteration.

Its matrix form is more complicated. But given $A = A^* > 0$ we have

$$A = L + D + L^*,$$

where D is the diagonal of A, L is lower-triangular part with zero on the diagonal.

One iteration of the GS method reads

$$x^{(k+1)} = x^{(k)} - (L + D)^{-1}(Ax^{(k)} - f).$$

and we refer to the preconditioner $P = L + D$ as **Gauss-Seidel preconditioner**.

Good news: $\rho(I - (L + D)^{-1}A) < 1$, where $\rho(\cdot)$ is the spectral radius, i.e. for a positive definite matrix GS-method always converges.

Note: convergence of Gauss-Seidel method can be improved by choosing "good" order of variables (e.g. such that the norm of lower triangular part is large).

Successive overrelaxation

We can even introduce a parameter ω into the GS-method preconditioner, giving a **successive overrelaxation** (SOR(ω)) method:

$$x^{(k+1)} = x^{(k)} - \omega(D + \omega L)^{-1}(Ax^{(k)} - f).$$

$$P = \frac{1}{\omega}(D + \omega L).$$

1. Converges for $0 < \omega < 2$ and $\rho(J) = \rho(I - D^{-1}A) < 1$ (i.e. spectral radius of Jacobi matrix less than 1 – Jacobi method converges).
2. Optimal selection of ω is **not trivial**. If the Jacobi method converges, then

$$\omega^* = \frac{2}{1 + \sqrt{1 - \rho_J^2}},$$

   where $\rho(J)$ is spectral radius of Jacobi iterations
3. Note that $\omega = 1$ gives us a Gauss-Seidel preconditioner.

31. Incomplete LU for preconditioning. ILU(τ), ILU(k), second-order ILU (ILU2).

Incomplete LU, general idea

1. Decompose the matrix A in the form $A = P_1 L U P_2^\top$, where P_1 and P_2 are certain permutation matrices (which do the pivoting).
2. We would like to have sparse L and U.
3. However, an honest LU is not possible without fill-in growth for example for matrices, coming from 2D/3D Partial Differential equations (PDEs).
4. But we can use some nice strategies in order to artificially make L and U sparse (of course it is now an approximation to the honest LU, so the methods are adequate for preconditioning only)

Incomplete LU(k)

1. Suppose you want to eliminate a variable x_1, and the equations have the form
$$5x_1 + x_4 + x_{10} = 1, \quad 3x_1 + x_4 + x_8 = 0, \ldots,$$
and in all other equations x_1 are not present.
2. After the elimination, only x_{10} will enter additionally to the second equation (new fill-in).
$$x_4 + x_8 + 3(1 - x_4 - x_{10})/5 = 0$$
3. In the Incomplete LU case (actually, ILU(0)) we just throw away the **new fill-in**.
4. Suppose now you have an $n \times n$ matrix A and a corresponding adjacency graph (each edge correspond to an element in the matrix).
5. When we eliminate one variable (vertex), the new edges can appear only between vertices that had common neighbour (the vertex that we have just eliminated): it means, that they are second-order neigbours. *Note:* resulting matrix have the same sparsity pattern as the matrix A^2.
6. The **ILU(k)** idea is to leave only the elements in L and U that are k-order neighbours in the original graph (k is the number of intermediate nodes in shortest path).
7. The ILU(2) is very efficient in practice.

ILU Thresholded (ILUT, ILU(τ))

A much more popular approach is based on the so-called **thresholded LU**.
You do the standard Gaussian elimination with fill-ins, but either:

1. Throw away elements that are smaller than threshold, and/or control the amount of non-zeros you are allowed to store.
2. The smaller is the threshold, the better is the preconditioner, but more memory it takes.

Note: In the symmtric-positive-definite case, instead of incomplete LU you should use Incomplete Cholesky, which is twice faster and consumes twice less memory.

Second-order LU

1. There is a more efficient (but much less popular due to the limit of open-source implementations) **second-order** LU factorization

2. The idea (for symmetric matrices) is to approximate the matrix in the form

$$A \approx U_2 U_2^\top + U_2^\top R_2 + R_2^\top U_2,$$

which is just the expansion of the $UU^\top$ with respect to the perturbation of U.

3. U_1 and U_2 are upper-triangular and sparse, whereare R_2 is small with respect to the drop tolerance parameter.

4. We are able to solve linear system with such matrix in a fast way

Not much information about this method in the lecture, hope it will be enough.

32. Inverse and Rayleigh quotient iterations. Speed of convergence. Convergence behavior of inexact inverse iteration.

Idea: we want to find ANY eigenvalue.

Inverse iteration is just a power method for A^{-1}:

$$x_{i+1} = \frac{A^{-1}x_i}{\|A^{-1}x_i\|}.$$

To accelerate convergence strategy can be used:

$$x_{i+1} = \frac{(A-\sigma I)^{-1}x_i}{\|(A-\sigma I)^{-1}x_i\|},$$

where σ (called shift) should be close to the eigenvalue we want to find.

Its convergence speed is know from power iteration:

$$q = \frac{\|\lambda_{closest\_to\_\sigma} - \sigma\|}{\|\lambda_{2^{nd}\_closest\_to\_\sigma} - \sigma\|}$$

Rayleigh quotient iterations are similar, but shift depends upon iteration:

$$x_{i+1} = \frac{(A-R(x_i)I)^{-1}x_i}{\|(A-R(x_i)I)^{-1}x_i\|},$$

where $R(x_k) = \frac{(x_i,Ax_i)}{(x_i,x_i)}$ is Rayleigh quotient.

This method converges cubically for Hermitian matrices and quadratically for non-Hermitian case.

Problems may arise in 2 methods above if σ or $R(x_i)$ are close to eigenvalues since then matrices become ill-conditioned. One solution is to find x_{i+1} approximately, with tolerance τ_{i+1}. If accuracy of solution of systems increases from iteration to iteration, superlinear (quadratic according to paper linked in lectures) convergence for RQ iteration can still be present. Otherwise convergence will be linear.

33. Block power method.

Recall that the simplest method to find the largest eigenvalue is the **power method**

$$x_{i+1} = \frac{Ax_i}{\|Ax_i\|}$$

The convergence is linear with rate $q = \left|\frac{\lambda_1}{\lambda_2}\right|$.

Block power method

The block power method (also known as subspace iteration method or simultaneous vector iteration) is a natural generalization of the power method for several largest eigenvalues.
It looks as:

1. Y_0 is $N \times k$ matrix of rank k, $Y_0 = X_0 R_0$ (QR-decomposition)
2. $Y_i = AX_{i-1}$
3. $Y_i = X_i R_i$ (QR-decomposition)

QR-decomposition plays role of normalization in the standard power method.

Moreover, orthogonalization prevents the columns of the X_i from converging all to the eigenvector corresponding to the largest modulus eigenvalue.

Accelerating convergence of the block power method

- For Hermitian matrices convergence of the j-column is **linear** as for the power method with $q = \frac{|\lambda_j|}{|\lambda_{j+1}|}$.
- Hence, applying the block power method to the matrix $(A - \sigma I)^{-1}$ will accelerate convergence (shift-and-invert strategy).
- You can also accelerate the convergence by applying the **Rayleigh-Ritz procedure**.

34. Ritz approximation: Ritz values and vectors and their properties. Rayleigh-Ritz method.

Ritz approximation

Given subspace spanned by columns of unitary matrix Q_k of size $N \times k$ we consider the projected matrix $Q_k^* A Q_k$.

Let $\Theta_k = \operatorname{diag}(\theta_1, \dots, \theta_k)$ and $S_k = \begin{bmatrix} s_1 & \dots & s_k \end{bmatrix}$ be matrices of eigenvalues and eigenvectors of $(Q_k^* A Q_k)$:

$$(Q_k^* A Q_k) S_k = S_k \Theta_k$$

then $\{\theta_i\}$ are called **Ritz values** and $y_i = Q_k s_i$ - **Ritz vectors**.

Properties of the Ritz approximation

- Note that they are not eigenpairs of the initial matrix $AY_k \neq Y_k \Theta_k$, but the following equality holds:

  $$Q_k^*(AY_k - Y_k\Theta_k) = Q_k^*(AQ_kS_k - Q_kS_k\Theta_k) = 0,$$

  so the residual for the Ritz approximation is **orthogonal** to the subspace spanned by columns of Q_k.

- $\lambda_{\min}(A) \leq \theta_{\min} \leq \theta_{\max} \leq \lambda_{\max}(A)$. Indeed, using Rayleigh quotient:

  $$\theta_{\min} = \lambda_{\min}(Q_k^* A Q_k) = \min_{x \neq 0} \frac{x^*(Q_k^* A Q_k)x}{x^* x} = \min_{y \neq 0 : y = Q_k x} \frac{y^* A y}{y^* y} \geq \min_{y \neq 0} \frac{y^* A y}{y^* y} = \lambda_{\min}(A).$$

  Obviously, $\lambda_{\min}(Q_k^* A Q_k) = \lambda_{\min}(A)$ if $k = N$, but we want to construct a basis $k \ll N$ such that $\lambda_{\min}(Q_k^* A Q_k) \approx \lambda_{\min}(A)$.

  Similarly, $\theta_{\max} \leq \lambda_{\max}(A)$.

Rayleigh-Ritz method

Thus, if a subspace V approximates first k eigenvectors, then one can use the **Rayleigh-Ritz method**:

1. Find orthonormal basis Q_k in V (e.g. by using QR decomposition)
2. Calculate $Q_k^* A Q_k$
3. Compute Ritz values and vectors
4. Note that alternatevly one could use V with no orthogonalization, but then generalized eigenvalue problem $(V^* A V) s_i = \theta_i (V^* V) s_i$ has to be solved.

The question is how to find a good subspace V. And it is answered in the next question (it's Krylov subspace).

35. Arnoldi relation (with derivation). Lanczos and Arnoldi methods for solving partial eigenvalue problem. Their advantages and disadvantages.

Arnoldi relation and Lanczos procedure can be seen in Question 26. For Hermitian matrices from the Arnoldi relation we have

$$Q_k^* A Q_k = T_k,$$

where Q_k is orthogonal basis in the Krylov subspace generated by the Lanczos procedure and T_k is triangular matrix. According to the Rayleigh-Ritz method we expect that eigenvalues of T_k approximate eigenvalues of A. This method is called the **Lanczos method**. For nonsymmetric matrices it is called the **Arnoldi method** and instead of tridiagonal T_k we would get upper Hessenberg matrix. In the lecture we proved that the maximum eigenvalue $\theta_{\max}$ of $Q_k^* A Q_k$ is close to the maximum eigenvalue A: $\theta_{\max} \approx \lambda_{\max}$ - see next question for derivation .

Advantages

1. Any number of eigenvalues can be found depending on the dimension of the constructed Krylov subspace.
2. Convergence rate is faster than for ordinary power-method, which in addition finds only the maximum eigenvalue.

Disadvantages

1. The Lanczos vectors may loose orthogonality during the process due to floating-point errors, thus all practical implementations of it use restarts.
2. Applying Lanczos directly to A may result into a very slow convergence if $\lambda_i \approx \lambda_{i+1}$ (typically holds for smallest eigenvalues that are not well-separated)

36. Convergence bound for λ_{max} in Lanczos method (with derivation).

Let us denote $\theta_1 \equiv \theta_{\max}$ and $\lambda_1 \equiv \lambda_{\max}$. Then:

$$\theta_1 = \max_{y \in \mathcal{K}_i, y \neq 0} \frac{(y, Ay)}{(y, y)} = \max_{p_{i-1}} \frac{(p_{i-1}(A)x_0, Ap_{i-1}(A)x_0)}{(p_{i-1}(A)x_0, p_{i-1}(A)x_0)}$$

where p_{i-1} is a polynomial of degree not greater than $i-1$ such that $p_{i-1}(A)x_0 \neq 0$. Since $\theta_1 \leq \lambda_1$ we get

$$\lambda_1 - \theta_1 \leq \lambda_1 - \frac{(p_{i-1}(A)x_0, Ap_{i-1}(A)x_0)}{(p_{i-1}(A)x_0, p_{i-1}(A)x_0)}$$

for any polynomial p_{i-1}. Exapnding $x_0 = \sum_{j=1}^{N} c_j v_j$, where v_j are eigenvectors of A forming orthonormal basis:

$$\lambda_1 - \theta_1 \leq \lambda_1 - \frac{\sum_{k=1}^{N} \lambda_k |p_{i-1}(\lambda_k)|^2 |c_k|^2}{\sum_{k=1}^{N} |p_{i-1}(\lambda_k)|^2 |c_k|^2} =$$
$$= \frac{\sum_{k=2}^{N} (\lambda_1 - \lambda_k) |p_{i-1}(\lambda_k)|^2 |c_k|^2}{|p_{i-1}(\lambda_1)|^2 |c_1|^2 + \sum_{k=2}^{N} |p_{i-1}(\lambda_k)|^2 |c_k|^2} \leq (\lambda_1 - \lambda_n) \frac{\max_{2 \leq k \leq N} |p_{i-1}(\lambda_k)|^2}{|p_{i-1}(\lambda_1)|^2} \gamma$$

where $\gamma = \frac{\sum_{k=2}^{N} |c_k|^2}{|c_1|^2}$. Here, in the first equality we vanished the first term in the sum and combined all others in differences, and in the final equality used $\lambda_1 - \lambda_n \geq \lambda_1 - \lambda_k,\ \forall k$. So in oreder to make the difference $\lambda_1 - \theta_1$ as small as possible we need to minimize the ratio of the polynomials, i.e.

$$|p_{i-1}(\lambda_1)| \gg \max_{2 \leq k \leq N} |p_{i-1}(\lambda_k)|$$

This holds for the Chebyshev polynomial T_{i-1} on the $[\lambda_n, \lambda_2]$, hence

$$\lambda_1 - \theta_1 \leq \frac{\lambda_1 - \lambda_n}{T_{i-1}^2(1 + 2\mu)} \gamma, \quad \mu = \frac{\lambda_1 - \lambda_2}{\lambda_2 - \lambda_n}$$

Due ti the fact that $\mu > 0$ the point $1 + 2\mu$ lies out of the interval $[0, 1]$, where Cehbyshev polynomials are growing very fast, thus the difference is small.

37. Fast Fourier Transform (FFT). Cooley-Tukey algorithm (with derivation).

To perform FFT, we first permute the rows of the Fourier matrix F_n so that the first $n/2$ rows of the new matrix have odd row numbers (in the F_n), and the last $n/2$ rows have even numbers. This matrix has the following form:

$$P_n = \left(\begin{array}{ccccccc}
1 & 0 & 0 & 0 & \dots & 0 & 0 \\
0 & 0 & 1 & 0 & \dots & 0 & 0 \\
\vdots & & & & & & \vdots \\
0 & 0 & 0 & 0 & \dots & 1 & 0 \\
\hline
0 & 1 & 0 & 0 & \dots & 0 & 0 \\
0 & 0 & 0 & 1 & \dots & 0 & 0 \\
\vdots & & & & & & \vdots \\
0 & 0 & 0 & 0 & \dots & 0 & 1
\end{array}\right).$$

The result of the permutation can be written as:

$$P_nF_n = \left(\begin{array}{ccccc}
1 & 1 & 1 & \dots & 1 \\
1 & w_n^{2\cdot 1} & w_n^{2\cdot 2} & \dots & w_n^{2\cdot (n-1)} \\
1 & w_n^{4\cdot 1} & w_n^{4\cdot 2} & \dots & w_n^{4\cdot (n-1)} \\
\vdots & & & & \vdots \\
1 & w_n^{(n-2)\cdot 1} & w_n^{(n-2)\cdot 2} & \dots & w_n^{(n-2)\cdot (n-1)} \\
\hline
1 & w_n^{1\cdot 1} & w_n^{1\cdot 2} & \dots & w_n^{1\cdot (n-1)} \\
1 & w_n^{3\cdot 1} & w_n^{3\cdot 2} & \dots & w_n^{3\cdot (n-1)} \\
\vdots & & & & \vdots \\
1 & w_n^{(n-1)\cdot 1} & w_n^{(n-1)\cdot 2} & \dots & w_n^{(n-1)\cdot (n-1)}
\end{array}\right) = \begin{pmatrix}
\left\{w_n^{2kl}\right\} & \left\{w_n^{2k\left(\frac{n}{2}+l\right)}\right\} \\
\left\{w_n^{(2k+1)l}\right\} & \left\{w_n^{(2k+1)\left(\frac{n}{2}+l\right)}\right\}
\end{pmatrix}, \quad k,l = 0,\dots,\frac{n}{2}-1.$$

If we look closer at the first block, we will notice that

$$w_n^{2kl} = e^{-2kl\frac{2\pi i}{n}} = e^{-kl\frac{2\pi i}{n/2}} = w_{n/2}^{kl},$$

hence, this block is exactly the Fourier matrix $F_{n/2}$ that is smaller by a factor of 2.

The block $\left\{w_n^{(2k+1)l}\right\}$ can be written as

$$w_n^{(2k+1)l} = w_n^{2kl+l} = w_n^l w_n^{2kl} = w_n^l w_{n/2}^{kl},$$

which can be written as $W_{n/2}F_{n/2}$, where

$$W_{n/2} = \operatorname{diag}(1, w_n, w_n^2, \dots, w_n^{n/2-1}).$$

Doing the same tricks for the other blocks we will finally get (work it out!)

$$P_nF_n = \begin{pmatrix} F_{n/2} & F_{n/2} \\ F_{n/2}W_{n/2} & -F_{n/2}W_{n/2} \end{pmatrix} = \begin{pmatrix} F_{n/2} & 0 \\ 0 & F_{n/2} \end{pmatrix} \begin{pmatrix} I_{n/2} & I_{n/2} \\ W_{n/2} & -W_{n/2} \end{pmatrix}.$$

Thus, we reduced multiplication by F_n to 2 multiplications by $F_{n/2}$ and cheap multiplications by diagonal matrices. If we apply the obtained expressions recursively to $F_{n/2}$, we will get $\mathcal{O}(n \log n)$ complexity. This complexity comes from the fact that there are $\log n$ reductions of a larger DFT into smaller ones, and each such reduction costs $\mathcal{O}(n)$ operations.

38. Continuous and discrete convolution. Eigendecomposition for circulant matrices (with proof).

Convolution is a mathematical operation on two functions that produces a third function $f * g$ that expresses how the shape of one is modified by the other. Definition:

$f * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$.

Similarly, discrete convolution is

$(f * g)_n = \sum_{-\infty}^{\infty} f_i g_{n-i}$

Lastly, when a function g_T is periodic with period T, then for any function f for which convolution exists, it is called circular, is also periodic and equal to:

$f * g_T(t) = \int_{t_0}^{T+t_0} [\sum_{-\infty}^{\infty} f(\tau + kT)] g_T(t-\tau)d\tau$

, where t_0 is an arbitrary parameter.

Circular discrete convolution, similarly, is

$(f * g)_n \equiv \sum_{m=0}^{N-1} [\sum_{\infty}^{\infty} f_{m+kN}] g_{n-m}$

, where N is the period for periodic series g.

discrete convolution can be represented by a matrix-vector multiplication with a special matrix, which is called circulant matrix. In such a matrix every row is the same as the previous row, just shifted to the right by 1 (wrapping around "cyclically" at the edges):

$f * g = Cg$, where $C_{ij} = f_{i-j \mod N}$

Eigendecomposition for circulant matrices

Theorem

Let C be a circulant matrix of size $n \times n$ and let c be it' s first column, then:

$$C = \frac{1}{n} F_n^* \text{diag}(F_n c) F_n$$

Proof:

Consider a number

$$\lambda(\omega) = c_0 + \omega c_1 + \ldots + \omega^{n-1} c_{n-1}$$

Multiply $\lambda(\omega)$ by powers of $\omega^k, k = 0 \ldots n-1$:

$$\begin{aligned} \lambda\omega &= c_{n-1} + \omega c_0 + \ldots + \omega^{n-1} c_{n-2} \\ \lambda\omega^2 &= c_{n-2} + \omega c_{n-1} + \ldots + \omega^{n-1} c_{n-3} \\ &\ldots \\ \lambda\omega^{n-1} &= c_1 + \omega c_2 + \ldots + \omega^{n-1} c_0 \end{aligned}$$

$$\lambda(\omega) \begin{pmatrix} 1 & \omega & \ldots & \omega^{n-1} \end{pmatrix} = \begin{pmatrix} 1 & \omega & \ldots & \omega^{n-1} \end{pmatrix} C$$

Writing this for $\omega = 1, w_n, \ldots w_n^{n-1}$ one gets :

$$\Lambda F_n = F_n C$$

And finally:

$$C = \frac{1}{n} F_n^* C F_n \quad \Box$$

39. Product of Toeplitz matrix by vector via FFT. BTTB matrix-by-vector product via 2D FFT.

qWe can rapidly compute matvec product using FFT is we know that our input is Toeplitz matrix. Let firstly $C \in R^{n\times n}$ be a circular matrix and $c \in R^{n\times 1}$ - it's first column. Then there exists a formula

$$Cx = \frac{1}{n} F_n^* diag(F_n c) F_n x = \text{ifft}(\text{fft}(c) \circ \text{fft}(x))$$

Where $\odot$ is elementwise multiplication. So the only thing remains is to generalize it on any arbitrary Toeplitz matrix. Let we have matrix

$$T = \begin{pmatrix} t_0 & t_{-1} & t_{-2} \\ t_1 & t_0 & t_{-1} \\ t_2 & t_1 & t_0 \end{pmatrix}$$

Then we can present Tx product as

$$Tx = \left(\begin{pmatrix} t_0 & t_{-1} & t_{-2} & t_2 & t_1 \\ t_1 & t_0 & t_{-1} & t_{-2} & t_2 \\ t_2 & t_1 & t_0 & t_{-1} & t_{-2} \\ t_{-2} & t_2 & t_1 & t_0 & t_{-1} \\ t_{-1} & t_{-2} & t_2 & t_1 & t_0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ 0 \\ 0 \end{pmatrix} \right)_{(:3)} = \text{ifft}\left(\text{fft}\left(\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_{-2} \\ t_{-1} \end{pmatrix} \right) \circ \text{fft}\left(\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ 0 \\ 0 \end{pmatrix} \right) \right)_{(:3)}$$

where (: 3) means that we only need first 3 elements.

Multilevel Toeplitz matrices

The 2-dimensional convolution is defined as

$$y_{i_1 i_2} = \sum_{j_1, j_2 = 1}^{n} t_{i_1 - j_1, i_2 - j_2} x_{j_1 j_2}.$$

Note that x and y are 2-dimensional arrays and T is 4-dimensional. To reduce this expression to matrix-by-vector product we have to reshape x and y into long vectors:

$$\text{vec}(x) = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1n} \\ \hline \vdots \\ \hline x_{n1} \\ \vdots \\ x_{nn} \end{pmatrix}, \quad \text{vec}(y) = \begin{pmatrix} y_{11} \\ \vdots \\ y_{1n} \\ \hline \vdots \\ \hline y_{n1} \\ \vdots \\ y_{nn} \end{pmatrix}.$$

In this case matrix T is **block Toeplitz with Toeplitz blocks:** (BTTB)

$$T = \begin{pmatrix} T_0 & T_{-1} & T_{-2} & \dots & T_{1-n} \\ T_1 & T_0 & T_{-1} & \dots & T_{2-n} \\ T_2 & T_1 & T_0 & \dots & T_{3-n} \\ \dots & \dots & \dots & \dots & \dots \\ T_{n-1} & T_{n-2} & T_{n-3} & \dots & T_0 \end{pmatrix}, \quad \text{where} \quad T_k = t_{k, i_2 - j_2} \quad \text{are Toeplitz matrices}$$

Fast matvec with multilevel Toeplitz matrix

To get fast matvec we need to embed block Toeplitz matrix with Toeplitz blocks into the block circulant matrix with circulant blocks. The analog of

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \star \\ \star \end{pmatrix} = \text{ifft}\left(\text{fft}\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_{-2} \\ t_{-1} \end{pmatrix} \circ \text{fft}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \\ 0 \end{pmatrix}\right).$$

will look like

$$\begin{pmatrix} y_{11} & y_{12} & y_{13} & \star & \star \\ y_{21} & y_{22} & y_{23} & \star & \star \\ y_{31} & y_{32} & y_{33} & \star & \star \\ \star & \star & \star & \star & \star \\ \star & \star & \star & \star & \star \end{pmatrix} = \text{ifft2d}\left(\text{fft2d}\begin{pmatrix} t_{0,0} & t_{1,0} & t_{2,0} & t_{-2,0} & t_{-1,0} \\ t_{0,1} & t_{1,1} & t_{2,1} & t_{-2,1} & t_{-1,1} \\ t_{0,2} & t_{1,2} & t_{2,2} & t_{-2,2} & t_{-1,2} \\ t_{0,-2} & t_{1,-2} & t_{2,-2} & t_{-2,-2} & t_{-1,-2} \\ t_{0,-1} & t_{1,-1} & t_{2,-1} & t_{-2,-1} & t_{-1,-1} \end{pmatrix} \circ \text{fft2d}\begin{pmatrix} x_{11} & x_{12} & x_{13} & 0 & 0 \\ x_{21} & x_{22} & x_{23} & 0 & 0 \\ x_{31} & x_{32} & x_{33} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}\right)$$

where fft2d is 2-dimensional fft that consists of one-dimensional transforms, applied first to rows and and then to columns (or vice versa).

40. Matrix functions. Matrix exponential and its applications. Problem with evaluating matrix exponential and how to avoid it. Schur-Parlett algorithm. Matrix functions via Pade approximation.

- Matrix Function:

- We can use several ways to define a matrix function:

One way to define a matrix function $f(A)$ is to use **Jordan canonical form**.

A much more elegant way is to use **Cauchy integral representation**:

$$f(A) = \int_\Gamma f(z)(zI - A)^{-1} dz,$$

where $f(z)$ is analytic on and inside a closed contour Γ that encloses the spectrum of A.

One of the simplest matrix function is matrix **polynomial**:

$$P(A) = \sum_{k=0}^{n} c_k A^k.$$

Side_note: Hamilton-Cayley theorem states that $F(A) = 0$ where $F(\lambda) = \det(A - \lambda I)$, thus *all matrix polynomials have degree* $\leq n - 1$.

Matrix polynomials can be building blocks for more complex functions. Because we can define a function of the matrix by **Taylor series**:

$$f(A) = \sum_{k=0}^{\infty} c_k A^k.$$

The convergence is understood as the convergence in some **matrix norm**.

Example of such series is the **Neumann series:**

$$(I - F)^{-1} = \sum_{k=0}^{\infty} F^k,$$

which is well defined for $\rho(F) < 1$

Important matrix functions: - $\cos(A), \sin(A)$ used to solve wave equation $\frac{d^2y}{dt^2} + Ay = 0$. - Sign function, $\mathrm{sign}(A)$, used to compute spectral projections. - Inverse square root $A^{-1/2}$ used in many places, for example, to generate samples from a Gaussian distributions

- Matrix exponential and its applications:

The most well-known matrix function is **matrix exponential**. In the scalar case,

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots = \sum_{k=0}^{\infty} \frac{x^k}{k!},$$

and it directly translates to the matrix case:

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!},$$

the series that always converges, because the series

$$\sum_{k=0}^{\infty} \frac{\|A\|^k}{k!} = e^{\|A\|}.$$

Matrix exponential is important. For example a lot of practical problems are reduced to a system of linear ODEs of the form

$$\frac{dy}{dt} = Ay, \quad y(0) = y_0.$$

And the formal solution is given by $y(t) = e^{At}y_0$, so if we know e^{At} (or can compute matrix-by-vector product fast) there is a big gain over the time-stepping schemes. Indeed,

$$\frac{d}{dt}e^{At} = \frac{d}{dt}\sum_{k=0}^{\infty}\frac{t^kA^k}{k!} = \sum_{k=1}^{\infty}\frac{t^{k-1}A^k}{(k-1)!} = Ae^{At}.$$

+ It is more productive to use matrix exponential then to solve using Euler scheme:

$$\frac{dy}{dt} \approx \frac{y_{k+1} - y_k}{\tau} = Ay_k, \quad y_{k+1} = y_k + \tau Ay_k,$$

+ Also, for dense matrices matrix exponential also provides exact answer to the ODE for any t, compared to the approximation by time-stepping schemes.

- Problem with evaluating matrix exponential and how to avoid it: (strange question, on lecture some algorithms of matrix exponential were discussed with their problems, but the main problem with evaluating matrix exponential can be bad convergence when eigenvalues of matrix is large negative values)

* The simplest way to evaluate matrix exponential is to diagonalize the matrix:

$$A = S\Lambda S^{-1},$$

where the columns of S are eigenvectors of the matrix A, then

$$F(A) = SF(\Lambda)S^{-1}.$$

Problem: diagonalization can be unstable!(and not every matrix is diagonalizable)

To avoid: not use it, if you are not sure that matrix is diagonalizable.

* The evaluation by series is a bad idea, because the series convergence for the matrix exponential can be slow for big norm.

* We can use the idea of Krylov method: using the Arnoldi method, generate the orthogonal basis in the Krylov subspace, and compute (it can be used in general for any function)

$$f(A)v \approx f(QHQ^*)v = Qf(H)Q^*v,$$

where H is a small upper Hessenberg matrix, for which we can use, for example, the Schur-Parlett algorithm.

Problem: The convergence of the Krylov method can be quite slow: it is actually a polynomial approximation to a function.And convergence of polynomial approximation to the matrix function can be slow.

To avoid: Replace by rational approximation (Pade approximation)

* Scaling and squaring algorithm

The "canonical algorithm" for the computation of the matrix exponential also relies on scaling of the matrix A :

$$\exp(A) = \exp\left(A/2^k\right)^{(2^k)}.$$

The matrix then can have a small norm, thus:

Scale the matrix as $B := A/2^k$ to make it norm less than 1.

Compute exponent of $C = e^B$ by a Pade approximation

Square $e^A \approx C^{(2^k)}$ in k matrix-by-matrix products.

Problem: Large-scale matrices obviously do not allow for efficient scaling-and-squaring (need to work with dense matrices).

To avoid: Use rational Krylov subspace.
- Schur-Parlett algorithm:
Given a matrix A we want to compute $F(A)$, and we only can evaluate F at scalar points.
First, we reduce A to the **triangular form** as

$$A = UTU^*.$$

Therefore, $F(A) = UF(T)U^*$.
To compute the function of triangular matrices:

$$F_{ii} = F(T_{ii}),$$

Using the known values on the diagonal and the commutativity property $TF(T) = F(T)T$, we get the diagonals of the matrix one-by-one:

$$f_{ij} = t_{ij} \frac{f_{ii} - f_{jj}}{t_{ii} - t_{jj}} + \sum_{k=i+1}^{j-1} \frac{f_{ik}t_{kj} - t_{ik}f_{kj}}{t_{ii} - t_{jj}}.$$

- Pade approximation:
Matrix exponential is well approximated by rational function:

$$\exp(x) \approx \frac{p(x)}{q(x)},$$

where $p(x)$ and $q(x)$ are polynomials and computation of a rational function of a matrix is reduced to matrix-matrix products and matrix inversions.

The rational form is also very useful when only a product of a matrix exponential by vector is needed, since evaluation reduces to matrix-by-vector products and linear systems solvers.

41. Sylvester and Lyapunov equations, their applications. Solving Sylvester equation using Bartels-Stewart method.

Sylvester equation

$$AX + XB = C$$

Where A, B, C are given and X is the unknown.

Continious Lyapunov equation

$$AX + XA^T = C$$

Discrete Lyapunov equation

$$AXA^* - X = C$$

Applications of the Lyapunov equation

Lyapunov equation is very important for the stability of dynamical systems, and also for model order reduction.

- Lyapunov equation is very important for the stability of dynamical systems, and also for model order reduction $y(t) \to 0$ for $t \to \infty$. System is stable, iff for any $Q = Q^* > 0$ there exists a unique positive definite solution P of the Lyapunov equation

  $$AP + PA^* = Q$$

  The stability then can be checked without finding eigenvalues

- Model order reduction of linear time-invariant systems:

  $$\frac{dx}{dt} = Ax + Bu, \quad y = Cx,$$

  where x is **state**, u is control, and y is the observable. We want to approximate it by a smaller-dimensional linear system

  $$\frac{d\widehat{x}}{dt} = \widehat{A}\widehat{x} + \widehat{B}u, \quad y = \widehat{C}\widehat{x}$$

  in such a way that the output of the reduced system is close to the output of the original (big one). The optimal $\widehat{A}, \widehat{B}, \widehat{C}$ can be recovered from the solution of the auxiliary Lyapunov equations.

Solution of the Sylvester equation

$$AX + XB = C$$

- This is a system of linear equations for X

- It can be rewritten as a linear system using the **vec** and **Kronecker** product operations.

$$\mathrm{vec}(AX + XB) = (I \otimes A + B^\top \otimes I)\mathrm{vec}(X) = \mathrm{vec}(C).$$

The naive Gaussian elimination will take $\mathcal{O}(n^6)$ operations.
We can do it in $\mathcal{O}(n^3)$ operations!

Bartels-Stewart method

$$(I \otimes A + B^\top \otimes I)x = c.$$

Let us compute Schur decomposition of A and B:

$$A = Q_A T_A Q_A^*, \quad B^\top = Q_B T_B Q_B^*.$$

Then, we have:

$$(I \otimes A + B^\top \otimes I) = (I \otimes (Q_A T_A Q_A^*) + (Q_B T_B Q_B^* \otimes I) = (Q_B \otimes Q_A)(I \otimes T_A + T_B \otimes I)(Q_B^* \otimes Q_A^*).$$

Note, that:

$$(Q_B \otimes Q_A)^{-1} = Q_B^* \otimes Q_A^*,$$

thus we only need to solve an auxiliary linear system with the matrix

$$I \otimes T_A + T_B \otimes I.$$

Note, that if A and B are Hermitian, then T_A and T_B are diagonal, and this matrix is diagonal!
We have the system

$$(I \otimes T_A + T_B \otimes I)z = g,$$

in the matrix form:

$$T_A Z + Z T_B^\top = G.$$

Then we just write the equation elementwise and see that the equations are solved successively for $Z_{11}, Z_{21}, \ldots,$

Basic Questions. Basics questions of NLA (debilnik)

Question 1: Floating vs. fixed point representation of numbers

Fixed point representation

Numbers are represented as a number lying in the range $[-2^m, 2^m - 2^{-n}]$ with step equal to 2^{-n} and predefined fixed m and n for all numbers.

In other words, this range looks like $[-2^m, -2^m + 2^{-n}, -2^m + 2 \cdot 2^{-n}, \dots, 2^m - 2 \cdot 2^{-n}, 2^m - 2^{-n}]$.

Every real number is floored to the closest one in this range in order to be stored, i.e if $m = 3$ and $n = 2$, then $1.27 = -2^3 + 37 \cdot 2^{-2} = -8 + 9.25 = 1.25$

Floating point representation

Numbers are represented in the following format:

$number = significand \times base^{exponent}$

Significand is *integer*, *base* is positive integer and *exponent* is integer, i.e. $1.27 = 127 \cdot 10^{-2}$

Pros/Cons

- Fixed point represents numbers within specified range, so we control absolute accuracy

- Floating point represents number with relative accuracy, so it is suitable for computations with numbers varying in scale (i.e. 10^{-1} and 10^5)

Question 2

Definitions of vector and matrix norms. Basic norms: p-norms (vector and matrix), Frobenius norm.
Answer:
Vector norm is a measure of vector smallness (or largeness, if you wish). You just want to assign a small number to small vectors and a big number to large vectors. And of course you may use different rules assigning these numbers. These different rules correspond to different norms. However, there are some constraints the rule should satisfy in order to be called a norm. Here they are:

$$||x|| \geq 0$$

$$||x|| = 0 \implies x = 0$$

$$||\alpha x|| = |\alpha| ||x||$$

$||x+y|| \leq ||x|| + ||y||$ (triangle inequality). Despite different norms assign different numbers to the same vector, they all are equivalent in a sense that if the vector is small in one norm, it is also small in another. That is, for any x, holds $C_1||x||_* \leq ||x||_{**} \leq C_2||x||_*$.

I believe, we only discussed one family of vector norms, namely, p-norm:

$$||x||_p = \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}$$

p should be greater or equal to 1 (else triangle inequality doesn't hold and such measure is not considered to be a norm). In case of $p = 2$ it becomes a well-known Euclidian norm. Taking limit with $p \to \infty$ results in Chebyshev norm - that is, maximum element of a vector (that's because all the elements to very large power are negligible compared to the largest element to very large power). In case of $p = 1$ p-norm becomes L_1-norm (aka Manhattan distance):

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

Which is visualized as the distance you'd walk from the point a to the point b, if you were only allowed to move along the x or y axis (here's an image).
Now, about matrix norms. They are a little trickier, and don't have such straight-forward philosophy of "smallness of a matrix". The most naive way to define matrix norm is a Frobenius norm - we just treat a matrix as a vector, but for whatever reason written in 2D, and take 2-norm of this vector:

$$||A||_F = ||vec(A)||_2 = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2}$$

There is also Chebyshev norm of a matrix - analogously to a Chebyshev norm of a vector, it is

$$||A||_c = \max_{i,j} |a_{ij}|$$

But these two guys are just treating matrices as weirdly written vectors. The set of norms, specifically designed for matrices, is operator norms. Operator norm is defined for two vector norms ($||.||_*$ and $||.||_{**}$) as:

$$||A||_{*,**} = \sup_{x \neq 0} \frac{||Ax||_*}{||x||_{**}}$$

Operator norms have nice intuitive meaning. If we perceive a matrix as a transformation performed on a vector, then operator norm of a matrix is maximum magnitude by which a matrix can stretch

a vector (we have a length of a transformed vector in the numerator and a length of initial vector in the denominator, and sup is responsible for choosing the maximum value over all possible x vectors). Operator norm is defined with respect to two vector norms, but in my opinion it's quite poorly interpretable in terms of two different vector norms, so one can think of it just as if $||.||_*= ||.||_{**}$. Also, only such norms are further considered in the course.
Operator norms also have a nice property - submultiplicativity. That is, $||AB||\leq ||A||*||B||$. For example, Chebyshev norm for matrices doesn't have it, but Frobenius norm does. This property is important, because in linear algebra we often multiply a vector by matrix over and over again and we want to be able to impose an expected upper bound on the result. I also am not sure I understand this justification correctly, so feel free to edit.
The p-norms are also defined for matrices. These are operator norms, with $||.||_*= ||.||_{**}= ||.||_p$. Important special cases of such norms are with $p = 1, 2, \infty$. The case with $p = 2$ is called spectral norm. One cannot directly compute spectral norm via some explicit formula, but for $p = 1, \infty$ it is possible (I didn't delve into the reasons why it is true):

$$||A||_1= \max_j \sum_{i=1}^{n} |a_{ij|}$$

(maximum of the column sums by absolute values)

$$||A||_\infty= \max_i \sum_{j=1}^{n} |a_{ij|}$$

(same for the rows)

Question 3

Complexity of basic linear algebra operations: e.g. matrix-vector, matrix-matrix products.
Answer:

Matrix-by-vector multiplication (matvec) requires $2n^2 - n$ operations, $O(n^2)$ complexity. In general we cannot do better, however this is not good enough for some large scale problems like modeling galaxies ($n = 10^{11}$). The complexity can be improved when the matrix is sparse or structured:
Complexity linear in the number of non-zero elements $\implies O(n)$ for sparse matrices.
Constant matrix (rank-1) $\implies$ n operations.
The matrix by vector product with the **Fourier matrix** (DFT) can be done with Fast Fourier Transform (FFT) in $O(n \log(n))$.
Circulant matrix and **Toeplitz matrix** mat-vecs can be reduced to FFTs and result in $O(n \log(n))$ complexity as well.

Matrix-by-matrix multiplication requires $2n^3 - n^2$ operations, $O(n^3)$ complexity when using the naive algorithm $c_{ij} = \sum_k (a_{ik} b_{kj})$. When the matrices are not square, the naive complexity is $O(nmp)$ for matrices A that is $n \times m$ and $B : m \times p$.
To improve the time required for matrix-by-matrix multiplications one should use block-version of algorithms to make use of fast cache memory. Also speed up can be done by parallelization (particularly useful for GPU).
Faster asymptotically algorithm, sometimes used in practice is: Strassen $\implies O(n^{log_2(7)}) \approx O(n^{2.807})$.
An even faster algorithm asymptotically is based on Coppersmith-Winograd with $O(n^{2.372})$ but the constant is too large for the algorithm to be practical.

Question 4

Definition of eigenvalues and eigenvectors of a matrix. Characteristic equation.
Answer:

A vector $x \neq 0$ is called an eigenvector of a square matrix A if there exists number λ:

$$Ax = \lambda x$$

λ - eigenvalue
Since $A - \lambda I$ should have non-trivial Kernel, eigenvalues are the roots of the characteristic polynomial:

$$det(A - \lambda I) = 0$$

Characteristic equation:

$$p(\lambda) = det(A - I) = 0$$

Polynomial of order n
n - number of complex roots.

Question 5

What is singular value decomposition?
Answer:
Theorem Any matrix $A \in \mathbb{C}^{n\times m}$ can be written as a product of three matrices:

$$A = U\Sigma V^*,$$

U - $n \times K$ unitary matrix,
V- $m \times K$ unitary matrix, $K = \min(m, n)$,
Σ - a diagonal matrix with non-negative elements $\sigma_1 \geq \ldots, \geq \sigma_K$ on the diagonal.
Moreover, if $\operatorname{rank}(A) = r$, then $\sigma_{r+1} = \cdots = \sigma_K = 0$.

Question 6

Definition of Hermitian and unitary (symmetric, orthogonal) matrices. Properties of their eigenvalues
Answer:
Complex $n \times n$ square matrix U is called unitary if:

$$U^*U = UU^* = I_n$$

Columns and rows of unitary matrix both form orthonormal basis in C^n
In case of real matrix $U^* = U^T$ and matrix such that

$$U^TU = UU^T = I_n$$

called orthogonal.

Hermitian matrix - is a complex square matrix that is equal to its own conjugate transpose:

$$A = A^*$$

In case of real matrix $A^* = A^T$, and if $A = A^T$ - we have a symmetric matrix.
Every Hermitian matrix is a normal matrix ($A^*A = AA^* = AA$). We can represent any normal matrix as :

$$A = U\Lambda U^*$$

U - unitary matrix,
Λ - diagonal matrix which diagonal elements are eigenvalues of A.

$$A^* = U\Lambda^*U^* = U\Lambda U^* = A$$

$$\Rightarrow \Lambda^* = \Lambda$$

Hence, all eigenvalues of the Hermitian matrix A are real.
If we have a unitary matrix B:

$$B^*B = BB^* = I$$

B - normal matrix
$B = U\Lambda U^*$
Hence,

$$U\Lambda\Lambda^*U^* = U\Lambda^*\Lambda U^* = I$$

$$\Lambda\Lambda^* = \Lambda^*\Lambda = I$$

$$\Rightarrow |\lambda|^2 = 1 \Rightarrow |\lambda| = 1$$

Question 7

Definition of symmetric positive definite matrix
Answer:

Given a square matrix $A \in \mathbb{R}^{n \times n}$ and a vector $x \in \mathbb{R}^n$, the scalar value $x^T A x$ is called a quadratic form. Written explicitly, we see that

$$x^T A x = \sum_{i=1}^{n} x_i (Ax)_i = \sum_{i=1}^{n} x_i \left(\sum_{j=1}^{n} A_{ij} x_j \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j$$

We give the following definitions:
- A symmetric matrix $A \in \mathbb{S}^n$ is positive definite (PD) if for all non-zero vectors $x \in \mathbb{R}^n, x^T A x > 0$. This is usually denoted $A \succ 0$ (or just $A > 0$), and often times the set of all positive definite matrices is denoted $\mathbb{S}_{++}^n$
- A symmetric matrix $A \in \mathbb{S}^n$ is positive semidefinite (PSD) if for all vectors $x^T A x \geq 0$. This is written $A \succeq 0$ (or just $A \geq 0$), and the set of all positive semidefinite matrices is often denoted S_+^n
- Likewise, a symmetric matrix $A \in \mathrm{S}^n$ is negative definite (ND), denoted $A \prec 0$ (or just $A < 0$) if for all non-zero $x \in \mathbb{R}^n, x^T A x < 0$
- Similarly, a symmetric matrix $A \in \mathbb{S}^n$ is negative semidefinite (NSD), denoted $A \preceq 0$ (or just $A \leq 0$) if for all $x \in \mathbb{R}^n, x^T A x \leq 0$

Finally, a symmetric matrix $A \in \mathbb{S}^n$ is indefinite, if it is neither positive semidefinite nor negative semidefinite $-$ i.e., if there exists $x_1, x_2 \in \mathbb{R}^n$ such that $x_1^T A x_1 > 0$ and $x_2^T A x_2 < 0$ It should be obvious that if A is positive definite, then $-A$ is negative definite and vice versa. Likewise, if A is positive semidefinite then $-A$ is negative semidefinite and vice versa. If A is indefinite, then so is $-A$.

One important property of positive definite and negative definite matrices is that they are always full rank, and hence, invertible.

Question 8: Definition of normal matrices and their properties

Normal matrices - such matrices so

$$AA^* = A^*A$$

Theorem A - normal $\Leftrightarrow$ $A = U\Lambda U^*$ where Λ - diagonal and on the diagonal are eigenvalues and U - unitary and constructed by eigenvectors.

The main idea of the proof: first of all we can construct **Shur form** and then prove that non-diagonal elements of T are 0 (using the fact that $AA^* = A^*A$

Some properties

1. normal matrix is Hermitian $\Leftrightarrow$ its eigenvalues are real.
2. normal matrix is unitary $\Leftrightarrow$ its eigenvalues satisfy $|\lambda|= 1$.

proof

1. $\Rightarrow$ If normal matrix is hermitian $\rightarrow$ its eigenvalues are real. Hermitian matrix means: $A = A^*$ and normal - $AA^* = A^*A$ Let's consider decomposition of normal matrix A: $A = U\sum U^*$ and $A^* = (U\sum U^*)^* = U\sum^* U^* = U\sum U^* = A$ and it means, that $\sum = \sum^*$ and $\lambda_i = \lambda_i^* => \lambda_i$ are real.

   $\Leftarrow$ If normal matrix has real eigenvalues $\rightarrow$ it is hermitian.

   Let's consider the same decomposition as in 1): $A = U\sum U^*$ and $A^* = (U\sum U^*)^* = U\sum^* U^*$. As we know, that eigenvalues are real: $\sum = \sum^*$ and $A^* = (U\sum U^*)^* = U\sum^* U^* = U\sum U^* = A$. That's proof, that matrix is hermitian $A = A^*$

2. $\Rightarrow$ If normal matrix is unitary $\rightarrow$ its eigenvalues satisfy $|\lambda|= 1$

   As A - normal: $A = U\sum U^*$ and $AA^* = U\sum U^*U\sum^* U^* = U\sum\sum^* U^* = Udiag(\lambda_i^2)U^* = I \rightarrow U^*Udiag(\lambda_i^2)U^*U = U^*IU \rightarrow diag(\lambda_i^2) = I \rightarrow |\lambda|= 1$

   $\Leftarrow$ If normal matrix has eigenvalues satisfy $|\lambda|= 1$ $\rightarrow$ it is unitary

   As A - normal: $A = U\sum U^*$ and $AA^* = U\sum U^*U\sum^* U^* = U\sum\sum^* U^* = Udiag(\lambda_i^2)U^* = UIU* = I$

Question 9: Definition, existence and uniqueness of basic matrix decompositions: LU, QR, Cholesky, Schur, SVD, skeleton, eigendecomposition. Computational complexity of these decompositions.

1. **LU**

   Let's consider Gaussian elimination for $Ax = f$: $x_1 = f - (a_2x_2 - \ldots)/a_{11}$ and !!! $a_{11} \neq 0$ We have 2 steps:

   (a) forward step → coming to upper-triangular matrix. $Ly = f$

   (b) backward step $Ux = y$

   So actually we have $A = LU$ decomposition where L - low-triangular matrices and U - is upper-triangular. LU - is not unique, but we can take ones on L diagonal and win!

   For each elimination we have complexity $O(n^2)$ and n steps, so total complexity $O(n^3)$.

   The LU-factorization can be improved by using block LU-decomposition.

   $$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

   And make this decomposition recursively (A_11 - block LU decomposition and so on).

   There are two basic operations: compute LU-factorization of half-matrices + matrix-by-matrix product.

   The LU-decomposition algorithm does not fail if we do not divide by zero at every step of the Gaussian elimination.

   So we need **strictly regular matrices**

   The concept of pivoting

   We can also make some permutations of rows or columns and at each step, select the index that is maximal in modulus, and put it onto the diagonal.

2. **Cholesky**

   !!!Hermitian positive definite matrices

   For such matrices we have $A = RR^*$ where R - low-triangular matrix.

   The main idea of the proof: first of all there is exist LU-decomposition. Then considering Gaussian elimination we can construct exectly $A = RR^*$

3. **SVD**

   for any $A \in C^{nm}$ we can consider $A = U\Sigma V^*$ where U and V - are unitary and Σ - diagonal matrices with non-negative $\sigma_0 \geq \sigma_1 \geq \ldots \geq \sigma_k$. And if $rankA = r$ then $\sigma_{r+1} = \ldots = \sigma_k = 0$

   To **SVD** is connected **Eckart-Young theorem**

   Best rank k approximation for a matrix with SVD $A = U\Sigma V^T$ is $A_k = \sum_{i=1}^k u_i \sigma_i v_i^T$, where v_i, u_i are column of U, V. I.e.

   ${}_{A_k:\text{rank}(B_k)=k}\|A - B_k\|_2 = \sum_{i=1}^k u_i \sigma_i v_i^T = A_k$

4. **Schur**

   For any $A \in C^{n\times m}$ $A = UTU^*$ where U - unitary matrix and T - upper-triangular.

   The main idea of the proof:

1) first of all there is exists non-zero eigenvector v_1. Lets construct $V_1 = \begin{bmatrix} v_1 & \dots & v_n \end{bmatrix}$ where v_i $i \neq 1$ are orthogonal to v_1. Then we can achieve block-upper-triangular matrix $V_1^* A V_1 = \begin{bmatrix} \lambda_1 & * \\ 0 & A_2 \end{bmatrix}$

2) Then doing by induction we can construct A_2 and so on.

5. **QR**

For every $A \in C^{n \times m}$ matrix $A = QR$ were Q - (pseudo???) unitary and R - trapezoidal form.

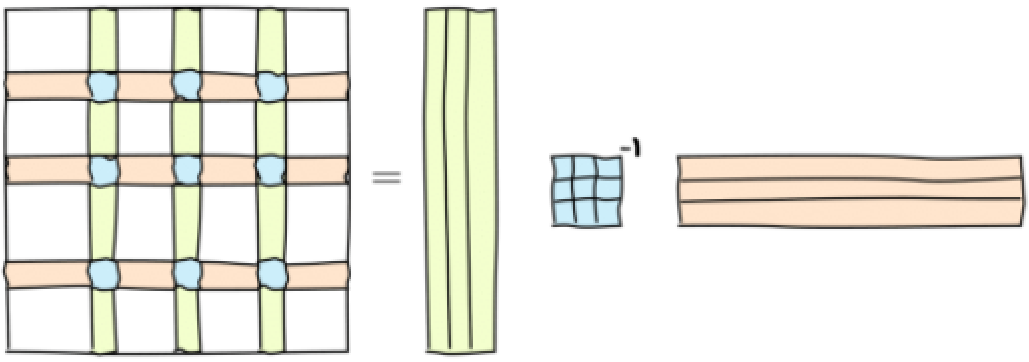
There are several different proof for existence. The full version i hope, you can find in "exam questions" but here we discus the main ideas only.

1) First of all we can prove the existence from Schur form (I believe it has to be Choletsky Decomposition). There are considered 2 different cases: A - full-rank matrix and not. The first situation is not difficult, in the second we consider convergence of full-rank matrices A_k to A and some properties of set of U as compact.

2) Second is using Gram-Schmidt algorithm

3) Also we can use a Householder reflections and Givens rotations

6. **skeleton**



for any matrices A with shape $n \times m$ we can $A = C\hat{A}^{-1}R = UV$ where C is $n \times r$, $\hat{A}$ - $r \times r$ and R - $r \times m$

7. **eigendecomposition**

The eigendecomposition is considered to **normal** matrices A:

A - normal $\Leftrightarrow$ $A = U\Lambda U^*$ where Λ - diagonal and on the diagonal are eigenvalues and U - unitary and constructed by eigenvectors.

The main idea of the proof: first of all we can construct **Shur form** and then prove that non-diagonal elements of T are 0 (using the fact that $AA^* = A^*A$

Question 10:Definition of condition number

Firstly we met the condition number in Neumann series. So if you need full information - you can find it in exam questions. But the mein idea is following. We have $Ax = f$ and a small perturbation $A + \Delta A$) and $f + \Delta f$ And we want to

Question 11: Defnitions of key matrices: Fourier, permutation, Householder, Givens, Hessenberg, triangular, Toeplitz, circulant.

Examples of **unitary** matrices:

Permutation matrix P whose rows (columns) are permutation of rows (columns) of the identity matrix.

Fourier matrix:

$$F_n = \frac{1}{\sqrt{n}} \left\{ e^{-i\frac{2\pi kl}{n}} \right\}_{k,l=0}^{n-1}.$$

Two important **classes of unitary matrices**, using composition of which we can construct any unitary matrix:

Householder matrix:

$$H \equiv H(v) = I - 2vv^*,$$

where v is an $n \times 1$ column and $v^*v = 1$. <br> - H is unitary and Hermitian ($H^* = H$). - It is also a reflection:

$$Hx = x - 2(v^*x)v.$$

<img src="householder.jpeg" width=500>

- A nice property of Householder transformation is that it can zero all elements of a vector except for the first one:

$$H \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix} = \begin{bmatrix} \times \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

- Householder algorithm for QR decomposition:

Using the obtained property we can make arbitrary matrix A lower triangular:

$$H_2 H_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix},$$

then finding $H_3 = \begin{bmatrix} I_2 & \\ & \widetilde{H}_3 \end{bmatrix}$ such that

$$\widetilde{H}_3 \begin{bmatrix} \times \\ \times \\ \times \end{bmatrix} = \begin{bmatrix} \times \\ 0 \\ 0 \end{bmatrix}$$

we get

$$H_3 H_2 H_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & \times \end{bmatrix}.$$

Finding H_4 by analogy we arrive at upper-triangular matrix.

Givens (Jacobi) matrix:

$$G = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix},$$

which is a rotation.

- For a general case, we select two (i, j) planes and rotate vector x

$$x' = Gx,$$

only in the i-th and j-th positions:

$$x'_i = x_i \cos\alpha - x_j \sin\alpha, \quad x'_j = x_i \sin\alpha + x_j \cos\alpha,$$

with all other x_i remain unchanged.

- Therefore, we can make elements in the j-th position zero by choosing α such that

$$\cos\alpha = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad \sin\alpha = -\frac{x_j}{\sqrt{x_i^2 + x_j^2}}.$$

- QR via Givens rotations:

Similarly we can make matrix upper-triangular using Givens rotations:

$$\begin{bmatrix} \times & \times & \times \\ * & \times & \times \\ * & \times & \times \end{bmatrix} \to \begin{bmatrix} * & \times & \times \\ * & \times & \times \\ 0 & \times & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times \\ 0 & * & \times \\ 0 & * & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}.$$

Householder vs. Givens transformations: - Householder reflections are useful for dense matrices (complexity is ≈ twice smaller than for Jacobi) and we need to zero large number of elements. - Givens rotations are more suitable for sparse matrice or parallel machines as they act locally on elements.

Hessenberg form:

The matrix A is said to be in the Hessenberg form, if

$$a_{ij} = 0, \quad \text{if } i \geq j + 2.$$

$$H = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}.$$

Reduction any matrix to Hessenberg form:

- By applying Householder reflections we can reduce any matrix to the Hessenberg form

$$U^* A U = H$$

- The only difference with Schur decomposition is that we have to map the first column to the vector with two non-zeros, and the first element is not changed.

- The computational cost of such reduction is $\mathcal{O}(n^3)$ operations.

- In a Hessenberg form, computation of one iteration of the QR algorithm costs $\mathcal{O}(n^2)$ operations (e.g. using Givens rotations), and the Hessenberg form is preserved by the QR iteration.

Toeplitz matrix:

A matrix is called **Toeplitz** if its elements are defined as

$$a_{ij} = t_{i-j}.$$

- A Toeplitz matrix is completely defined by its first column and first row (i.e., $2n - 1$ parameters).

- It is a **dense matrix**, however it is a **structured matrix** (i.e., defined by $\mathcal{O}(n)$ parameters).

- And the main operation in the discrete convolution is the product of Toeplitz matrix by vector.

Toeplitz and circulant matrix:

For a special class of Toeplitz matrices, named **circulant matrices** the fast matrix-by-vector product can be done.

A matrix C is called **circulant**, if

$$C_{ij} = c_{i-j \mod n},$$

i.e. it periodicaly wraps

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \\ c_3 & c_0 & c_1 & c_2 \\ c_2 & c_3 & c_0 & c_1 \\ c_1 & c_2 & c_3 & c_0 \end{bmatrix}.$$

- These matrices have the same **eigenvectors**, given by the Discrete Fourier Transform (DFT).

Spectral theorem for circulant matrices:

Any circulant matrix can be represented in the form

$$C = \frac{1}{n} F^* \Lambda F,$$

where F is the **Fourier matrix** with the elements

$$F_{kl} = w_n^{kl}, \quad k, l = 0, \ldots, n-1, \quad w_n = e^{-\frac{2\pi i}{n}},$$

and matrix $\Lambda = \text{diag}(\lambda)$ is the diagonal matrix and

$$\lambda = Fc,$$

where c is the first column of the circulant matrix C.

Question 12: Formulation of Eckart-Young theorem

The best low-rank approximation can be computed by SVD.

Theorem: Let $r < \operatorname{rank}(A)$, $A_r = U_r \Sigma_r V_r^*$. Then

$$\min_{\operatorname{rank}(B)=r} \|A - B\|_2 = \|A - A_r\|_2 = \sigma_{r+1}.$$

The same holds for $\|\cdot\|_F$, but $\|A - A_r\|_F = \sqrt{\sigma_{r+1}^2 + \dots + \sigma_{\min(n,m)}^2}$.

Proof: - Since $\operatorname{rank}(B) = r$, it holds $\dim \ker B = n - r$. - Hence there exists $z \neq 0$ such that $z \in \ker(B) \cap \mathcal{L}(v_1, \dots, v_{r+1})$ (as $\dim\{v_1, \dots, v_{r+1}\} = r + 1$). - Fix $\|z\| = 1$. Therefore,

$$\|A-B\|_2^2 \geq \|(A-B)z\|_2^2 = \|Az\|_2^2 = \|U\Sigma V^* z\|_2^2 = \|\Sigma V^* z\|_2^2 = \sum_{i=1}^{n} \sigma_i^2 (v_i^* z)^2 = \sum_{i=1}^{r+1} \sigma_i^2 (v_i^* z)^2 \geq \sigma_{r+1}^2 \sum_{i=1}^{r+1} (v_i^* z)^2 = \sigma_{r+1}^2$$

as $\sigma_1 \geq \dots \geq \sigma_{r+1}$ and

$$\sum_{i=1}^{r+1} (v_i^* z)^2 = \|Vz\|_2^2 = \|z\|_2^2 = 1.$$

Question 13: Formulation of the QR-algorithm

Schur form

- Recall that every matrix can be written in the **Schur form**

$$A = QTQ^*,$$

with upper triangular T and unitary Q and this decomposition gives eigenvalues of the matrix (they are on the diagonal of T).

- The first and the main algorithm for computing the Schur form is the **QR algorithm**.

QR algorithm

- The QR algorithm was independently proposed in 1961 by Kublanovskaya and Francis.
- <font color='red'> Do not **mix** QR algorithm and QR decomposition! </font>
- QR decomposition is the representation of a matrix, whereas QR algorithm uses QR decomposition to compute the eigenvalues!

Way to QR algorithm:

- Consider the equation

$$A = QTQ^*,$$

and rewrite it in the form

$$QT = AQ.$$

- On the left we can see QR factorization of the matrix AQ.
- We can use this to derive fixed-point iteration for the Schur form, also known as **QR algorithm.**

Derivation of the QR algorithm as fixed-point iteration We can write down the iterative process

$$Q_{k+1} R_{k+1} = A Q_k, \quad Q_{k+1}^* A = R_{k+1} Q_k^*$$

Introduce

$$A_k = Q_k^* A Q_k = Q_k^* Q_{k+1} R_{k+1} = \widehat{Q}_k R_{k+1}$$

and the new approximation reads

$$A_{k+1} = Q_{k+1}^* A Q_{k+1} = (Q_{k+1}^* A = R_{k+1} Q_k^*) = R_{k+1} \widehat{Q}_k.$$

So we arrive at the standard form of the QR algorithm.
The final formulas are then written in the classical **QRRQ**-form:
1. Start from $A_0 = A$. 2. Compute QR factorization of $A_k = Q_k R_k$. 3. Set $A_{k+1} = R_k Q_k$.
Iterate until A_k is **triangular enough** (e.g. norm of subdiagonal part is small enough).
What about the convergence and complexity
Statement
Matrices A_k are unitary similar to A

$$A_k = Q_{k-1}^* A_{k-1} Q_{k-1} = (Q_{k-1} \dots Q_1)^* A (Q_{k-1} \dots Q_1)$$

and the product of unitary matrices is a unitary matrix.
- Complexity of each step is $\mathcal{O}(n^3)$, if a general QR decomposition is done.
- Our hope is that A_k will be **very close to the triangular matrix** for suffiently large k.
Convergence and complexity of the QR algorithm
- The convergence of the QR algorithm is from the **largest eigenvalues** to the smallest.
- At least 2-3 iterations is needed for an eigenvalue.
- Each step is one QR factorization and one matrix-by-matrix product, as a result $\mathcal{O}(n^3)$ complexity.
Q: does it mean $\mathcal{O}(n^4)$ complexity totally?
A: fortunately, not.
- We can speedup the QR algorithm by using **shifts**, since $A_k - \lambda I$ has the same Schur vectors.
- We will discuss these details later
Does QR algorithm always convergence?
Answer: No Example: For a matrix
$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
we have $A_k = A$.
Connection to orthogonal iteration
In the previous lecture, we considered **power iteration**, which is $A^k v$ – approximation of the eigenvector.
The QR algorithm computes (implicitly) QR-factorization of the matrix A^k:

$$A^k = A \cdot \ldots \cdot A = Q_1 R_1 Q_1 R_1 \ldots = Q_1 Q_2 R_2 Q_2 R_2 \ldots (R_2 R_1) = \ldots (Q_1 Q_2 \ldots Q_k)(R_k \ldots R_1).$$

A few words about the SVD
- Last but not least: the **singular value decomposition** of matrix.

$$A = U \Sigma V^*.$$

- We can compute it via eigendecomposition of

$$A^* A = V^* \Sigma^2 V,$$

and/or

$$A A^* = U^* \Sigma^2 U$$

with QR algorithm, but it is a **bad idea** (c.f. Gram matrices).
- We will discuss methods for computing SVD later.
Schur form computation

- Recall that we are trying to avoid $\mathcal{O}(n^4)$ complexity. - The idea is to make a matrix have a simpler structure so that each step of QR algorithm becomes cheaper.

- In case of a general matrix we can use the **Hessenberg form**.

Hessenberg form

The matrix A is said to be in the Hessenberg form, if

$$a_{ij} = 0, \quad \text{if } i \geq j + 2.$$

$$H = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}.$$

Reduction any matrix to Hessenberg form

- By applying Householder reflections we can reduce any matrix to the Hessenberg form

$$U^* A U = H$$

- The only difference with Schur decomposition is that we have to map the first column to the vector with two non-zeros, and the first element is not changed.

- The computational cost of such reduction is $\mathcal{O}(n^3)$ operations.

- In a Hessenberg form, computation of one iteration of the QR algorithm costs $\mathcal{O}(n^2)$ operations (e.g. using Givens rotations, how?), and the Hessenberg form is preserved by the QR iteration (check why).

Symmetric (Hermitian) case

- In the symmetric case, we have $A = A^*$, then $H = H^*$ and the upper Hessenberg form becomes tridiagonal matrix.

- From now on we will talk about the case of symmetric tridiagonal form.

- Any symmetric (Hermitian) matrix can be reduced to the tridiagonal form by Householder reflections.

- **Key point** is that tridiagonal form is preserved by the QR algorithm, and the cost of one step can be reduced to $\mathcal{O}(n)$!

QR algorithm: iterations

- The iterations of the QR algorithm have the following form:

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k.$$

- If $A_0 = A$ is tridiagonal symmetric matrix, this form is preserved by the QR algorithm :

$$A_{k+1} = R_k A_k R_k^{-1}$$

Multiplication by triangular matrix, cannot add new elements below the first lower subdiagonal, and the next matrix is symmetric as well, therefore, no non-zeros above the first upper subdiagonal.

Tridiagonal form

- In the tridiagonal form, you do not have to compute the Q matrix: you only have to compute the **triadiagonal part** that appears after the iterations

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k,$$

in the case when $A_k = A_k^*$ and is also tridiagonal.

- Such matrix is defined by $\mathcal{O}(n)$ parameters; computation of the QR is more complicated, but it is possible to compute A_{k+1} directly without computing Q_k.

- This is called **implicit QR-step**.

Theorem on implicit QR iteration

All the implicit QR algorithms are based on the following **theorem**:

Let

$$Q^* A Q = H$$

be an irreducible upper Hessenberg matrix. Then, the first column of the matrix Q defines all of its other columns. It can be found from the equation

$$AQ = QH.$$

Convergence of the QR-algorithm

- The convergence of the QR-algorithm is a very **delicate issue** (see E. E. Tyrtyshnikov, "Brief introduction to numerical analysis" for details).

Summary. If we have a decomposition of the form

$$A = X \Lambda X^{-1}, \quad A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

and

$$\Lambda = \begin{bmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{bmatrix}, \quad \lambda(\Lambda_1) = \{\lambda_1, \dots, \lambda_m\}, \ \lambda(\Lambda_2) = \{\lambda_{m+1}, \dots, \lambda_r\},$$

and there is a **gap** between the eigenvalues of Λ_1 and Λ_2 ($|\lambda_1| \geq \dots \geq |\lambda_m| > |\lambda_{m+1}| \geq \dots \geq |\lambda_r| > 0$), then the $A^{(k)}_{21}$ block of A_k in the QR-iteration goes to zero with

$$\|A^{(k)}_{21}\| \leq Cq^k, \quad q = \left| \frac{\lambda_{m+1}}{\lambda_m} \right|,$$

where m is the size of Λ_1.

So we need to increase the gap! It can be done by the **QR algorithm with shifts**.

QR-algorithm with shifts

$$A_k - s_k I = Q_k R_k, \quad A_{k+1} = R_k Q_k + s_k I$$

The convergence rate for a shifted version is then

$$\left| \frac{\lambda_{m+1} - s_k}{\lambda_m - s_k} \right|,$$

where λ_m is the m-th largest eigenvalue of the matrix in modulus. - If the shift is close to the eigenvalue, then the convergence speed is better. - There are different strategies to choose shifts. - Introducing shifts is a general strategy to improve convergence of iterative methods of finding eigenvalues. - In next slides we will illustrate how to choose shift on a simpler algorithm.

Shifts and power method

- Remember the power method for the computation of the eigenvalues.

$$x_{k+1} := A x_k, \quad x_{k+1} := \frac{x_{k+1}}{\|x_{k+1}\|}.$$

- It converges to the eigenvector corresponding to the largest eigenvalue in modulus.
- The convergence can be very slow.
- Let us try to use shifting strategy. If we shift the matrix as

$$A := A - \lambda_k I$$

and the corresponding eigenvalue becomes small (but we need large). - That is not what we wanted!

Inverse iteration and Rayleigh quotient iteration

- To make a small eigenvalue large, we need to **invert the matrix**, and that gives us **inverse iteration**

$$x_{k+1} = (A - \lambda I)^{-1} x_k,$$

where λ is the shift which is approximation to the eigenvalue that we want.

- As it was for the power method, the convergence is linear.

- To accelerate convergence one can use the **Rayleigh quotient iteration** (inverse iteration with adaptive shifts) which is given by the selection of the **adaptive shift**:

$$x_{k+1} = (A - \lambda_k I)^{-1} x_k,$$

$$\lambda_k = \frac{(Ax_k, x_k)}{(x_k, x_k)}$$

- In the symmetric case $A = A^*$ the convergence is **locally cubic** and **locally quadratic** otherwise.

Small conclusion:

- QR algorithm: the "gold standard" of the eigenvalue computations - RQI-iteration: Rayleigh quotient iteration is implicitly performed at each step of the QR algorithm

15.Power method and how it converges

We are often interested in the computation of the part of the spectrum, like the largest eigenvalues or smallest eigenvalues.

Also it is interesting to note that for the Hermitian matrices ($A = A^*$) the eigenvalues are always real (prove it!). A power method is the simplest method for the computation of **the largest eigenvalue in modulus**.

It is also our first example of the **iterative method** and **Krylov method**.

Power method has the form

$$x_{k+1} = Ax_k, \quad x_{k+1} := \frac{x_{k+1}}{\|x_{k+1}\|_2}$$

and

$$x_{k+1} \to v_1,$$

where $Av_1 = \lambda_1 v_1$ and λ_1 is the largest eigenvalue and v_1 is the corresponding eigenvector.

Powever method finds the largest in modulus eigenvalue of A. On the $(k+1)$-th iteration approximation to λ_1 can be found as

$$\lambda^{(k+1)} = (Ax_{k+1}, x_{k+1}),$$

Note that $\lambda^{(k+1)}$ is not required for the $(k+2)$-th iteration, but might be useful to measure error on each iteration: $\|Ax_{k+1} - \lambda^{(k+1)} x_{k+1}\|$. The convergence is geometric, but the convergence ratio is q^k, where $q = \left|\frac{\lambda_2}{\lambda_1}\right| < 1$, for $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n$ and k is the number of iteration. It means, the convergence can be artitrary small. To prove it, it is sufficient to consider a 2×2 diagonal matrix.

Convergence analysis for $A = A^*$ Let's have a more precise look at the power method when A

is Hermitian. In two slides you will learn that every Hermitian matrix is diagonalizable. Therefore, there exists orthonormal basis of eigenvectors $v_1, \dots, v_n$ such that $Av_i = \lambda_i v_i$. Let us decompose x_0 into a sum of v_i with coefficients c_i:

$$x_0 = c_1 v_1 + \dots + c_n v_n.$$

Since v_i are eigenvectors, we have

$$\begin{aligned} x_1 &= \frac{Ax_0}{\|Ax_0\|} = \frac{c_1 \lambda_1 v_1 + \dots + c_n \lambda_n v_n}{\|c_1 \lambda_1 v_1 + \dots + c_n \lambda_n v_n\|} \\ &\vdots \\ x_k &= \frac{Ax_{k-1}}{\|Ax_{k-1}\|} = \frac{c_1 \lambda_1^k v_1 + \dots + c_n \lambda_n^k v_n}{\|c_1 \lambda_1^k v_1 + \dots + c_n \lambda_n^k v_n\|} \end{aligned}$$

Now you see, that

$$x_k = \frac{c_1}{|c_1|} \left(\frac{\lambda_1}{|\lambda_1|} \right)^k \frac{v_1 + \frac{c_2}{c_1} \frac{\lambda_2^k}{\lambda_1^k} v_2 + \dots + \frac{c_n}{c_1} \frac{\lambda_n^k}{\lambda_1^k} v_n}{\left\| v_1 + \frac{c_2}{c_1} \frac{\lambda_2^k}{\lambda_1^k} v_2 + \dots + \frac{c_n}{c_1} \frac{\lambda_n^k}{\lambda_1^k} v_n \right\|},$$

which converges to v_1 since $\left| \frac{c_1}{|c_1|} \left(\frac{\lambda_1}{|\lambda_1|} \right)^k \right| = 1$ and $\left(\frac{\lambda_2}{\lambda_1} \right)^k \to 0$ if $|\lambda_2| < |\lambda_1|$.

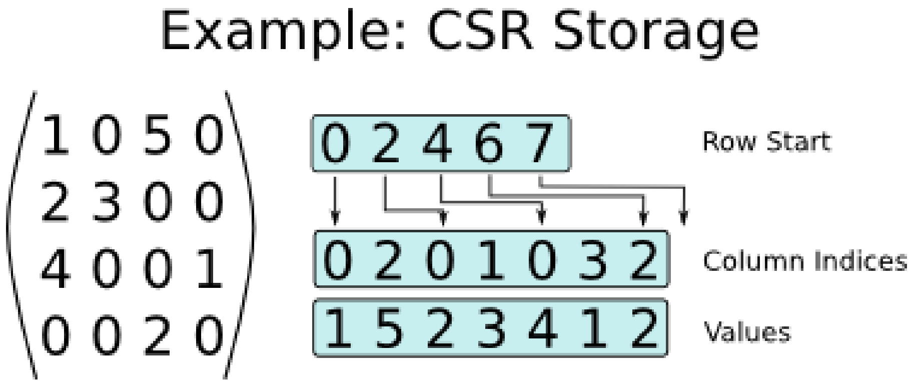
Things to remember about the power method

- Power method gives estimate of the largest eigenvalue in modulus or spectral radius of the given matrix
- One step requires one matrix-by-vector product. If the matrix allows for an $\mathcal{O}(n)$ matvec (for example, it is sparse), then power method is tractable for larger n.
- Convergence can be slow
- If only a rough estimate is needed, only a few iterations are sufficient
- The solution vector is in the **Krylov subspace** $\{x_0, Ax_0, \dots, A^k x_0\}$ and has the form $\mu A^k x_0$, where μ is the normalization constant.

16 CSR format for sparse matrix storing.

In the CSR format a matrix is stored as 3 different arrays:

- **ia** (row start) is an integer array of length $n + 1$
- **ja** (column indices) is an integer array of length**nnz**
- **sa** (values) is an real-value array of length **nnz**



So, we got $2 \cdot \mathbf{nnz} + n + 1$ elements.

Sparse matrices and efficiency

- Sparse matrices give complexity reduction.
- But they are**not very good** for parallel/GPU implementation.
- They do not give maximal efficiency due to **random data access**.
- Typically, peak efficiency of $10\% - 15\%$ is considered good.

18 Krylov subspace. Ideas of main Krylov methods: CG, MINRES, GMRES, bicgstab. Differences between these methods and when to apply them.

Introduction
Task: we have a linear system $Ax = f$, where A is symmetric positive definite. The idea to solve system is not to solve it but to minimise $(Ax, x) - 2(f, x)$ – the energy functional. The global minimum satisfies $Ax_* = f$. This will lead us to Galerkin projection ($Y^*AYc = Y^*(f - Ax_0) = Y^*r_0$,) and for this we actually have the following Krylov subspace:
Solution x_* lies in the Krylov subspace: $x_* \in \mathcal{K}_n(A, f)$. Here is an explanation of it:

1. According to Cayley–Hamilton theorem $p(A) = 0$, where $p(\lambda) = \det(A - \lambda I)$ is a characteristic polynomial.
2. $p(A)f = A^n f + a_1 A^{n-1} f + \ldots + a_{n-1} A f + a_n f = 0$
3. $A^{-1} p(A) f = A^{n-1} f + a_1 A^{n-2} f + \ldots + a_{n-1} f + a_n A^{-1} f = 0$
4. $x_* = A^{-1} f = -\frac{1}{a_n}(A^{n-1} f + a_1 A^{n-2} f + \ldots + a_{n-1} f)$
5. Thus, $x_* \in \mathcal{K}_n(A, f)$

So **the Krylov subpace** of the M-th order

$$\mathcal{K}_M(A, r_0) = \mathrm{Span}(r_0, Ar_0, \ldots, A^{M-1} r_0).$$

This is a subspace generated from a single vector $r_0 = f - Ax_0$:

$$y_0 \equiv k_0 = r_0, \quad y_1 \equiv k_1 = Ar_0, \quad y_2 \equiv k_2 = A^2 r_0, \ldots, \quad y_{M-1} \equiv k_{M-1} = A^{M-1} r_0.$$

The problem is that Krylov space is actually ill-conditioned (large condition values) because of the fact that power of matrix applied to a single vector converges to its eigenvalue. So it becomes more and

more colinear with each step. The solution to this problem is ORTHOGONALIZATION. However in different cases the different methods are appropriate for this.
Arnoldi relation (it is still introduction)
The first idea of orthogonalization is to use QR decomposition (Q - orthogonal, R - upper triangular) for iterative computation of Krylov subspace. (Each step will add a new dimension to the Krylov subspace). This solution will be stable (so it helps to solve our problem with ill-condition) however it can not suit us because we will need to start a computation of our orthogonal basis from scratch on each step. We need an iterative method (like Gram-Schmidt orthogonalization when the computation of a new vector is dependent on the previous ones)
Arnoldi relation is actually a modified Gram-Schmidt orthogonalization. It has the following formula (very important formula)

$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top,$$

where H_j is upper Hessenberg, and $Q_{j+1} = [q_0, \dots, q_j]$ has orthogonal columns that spans columns of K_{j+1}.
Lanczos relation (also introduction)
In the symmetric case the H_j from upper formula is actually not just upper Hessenberg, because it is also symmetric - it is tridiagonal. This leads us to significant improvement in possible computation - now we don't need to store the whole matrix to build a new vector! We only need 2 previous vectors:

$$t_{j,j-1} q_j = (AQ_j - Q_j T_j) e_{j-1} = Aq_{j-1} - t_{j-1,j-1} q_{j-1} - t_{j-2,j-1} q_{j-2}.$$

The coefficients $\alpha_j = t_{j-1,j-1}$ and $\beta_j = t_{j-2,j-1}$ can be recovered from orthogonality constraints. **Now we can move to the methods**

CG = Conjugate gradient method

symmetric positive case!
CG derivation overview

1. Want to find x_* in Krylov subspace
2. But natural basis is ill-conditioned, therefore we need orthogonalization
3. Derive recurrent equation for sequential orthogonalization of the Krylov subspace basis
   (a) Arnoldi process for non-symmetric matrix
   (b) Lanczos process for symmetric matrix (here we actually use residuals, not just final vectors in Gram-Schmidt. Hence the name conjugate gradient: to the gradient r_j we add a **conjugate direction** p_j. We have **orthogonality** of residuals: $(r_i, r_j) = 0, \quad i \neq j$ and **A-orthogonality** of conjugate directions: $(Ap_i, p_j) = 0$,)
4. Clever re-writing of these formulas gives short recurrence

Properties of the CG method

1. We need to store 3 vectors.
2. Since it generates A-orthogonal sequence $p_1, \dots, p_N$, after n steps it should stop (i.e., $p_{N+1} = 0$.)
3. Convergence
   (a) If A has only m different eigenvalues, CG converges in m iterations (proof in the blackboard).
   (b) If A has m "clusters" of eigenvalues, CG converges cluster-by-cluster.
4. $\mathcal{O}(n)$ memory

5. Square root of condition number in the estimates

6. Automatic ignoring of the outliers/clusters

7. No hyperparameters

8. A-optimality property (Energy functional can be written as $(Ax, x) - 2(f, x) = (A(x - x_*), (x - x_*)) - (Ax_*, x_*)$, $(A(x - x_*), (x - x_*)) = \|x - x_*\|_A^2$ is the A-norm of the error)

9. (note) Better convergence than Chebyshev acceleration, but slightly higher cost per iteration.

GMRES = the generalized minimal residual method

Non-symmetric case!
GMRES derivation overview

1. Want to find x_* in Krylov subspace but **now A is non-symmetric**

2. orthogonalize using Arnoldi process
$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top.$$

3. Let us rewrite the latter expression as
$$AQ_j = Q_j H_j + h_{j,j-1} q_j e_{j-1}^\top = Q_{j+1} \widetilde{H}_j$$
So each step we need to $\|\widetilde{H}_j c_j - \gamma e_0\|_2 \to \min_{c_j}$,

Summary of the GMRES

1. Minimizes the residual directly

2. No normal equations

3. Memory grows with the number of iterations as $\mathcal{O}(nj)$, so **restarts** typically implemented (just start GMRES from the new initial guess)

MINRES = symmetric GMRES

Symmetric case!
MINRES derivation overview

1. The difference to GMRES is that we minimize $\|AQ_j x_j - f\|_2 = \|Q_j \widehat{x}_j + h_{j,j-1} q_j \widehat{x}_j - f\|_2 = \|Q_{j+1} \widehat{H}_{j+1} \widehat{x}_j - f\|_2 \to \min$ which is equivalent to a linear least squares with an **almost tridiagonal** matrix
$$\|\widehat{H}_{j+1} x_j - \gamma e_0\|_2 \to \min.$$

Difference between MINRES and CG

1. MINRES minimizes $\|Ax_k - f\|_2$ over the Krylov subspace

2. CG minimize $(Ax, x) - 2(f, x)$ over the Krylov subspace

3. MINRES works for indefinite (i.e., non-positive definite) problems.

4. CG stores less vectors (3 instead of 5)

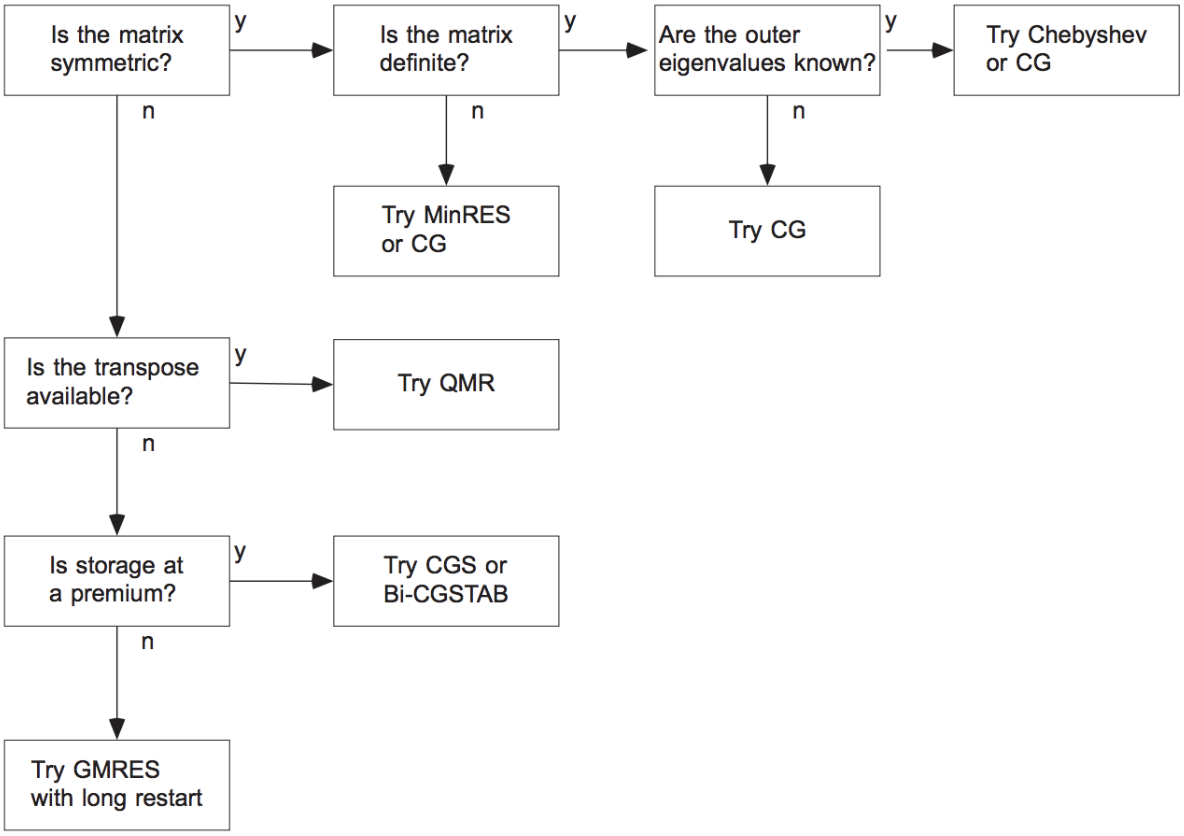
BiCGStab

Non-symmetric case!
BiCGStab derivation overview

1. Improvement (stabilized) of BiConjugate Gradients (BiCG method is using the normal equations $A^*Ax = A^*f$, and apply the CG method to it. It has squared condition number)

What methods to use

1. If a matrix is symmetric (Hermitian) positive definite, use CG method.
2. If a matrix is symmetric but indefinite, we can use MINRES method (GMRES applied to a symmetric system)
3. If a matrix is non-symmetric and not very big, use GMRES
4. If a matrix is non-symmetric and we can store limited amount of vectors, use either: GMRES with restarts, or BiCGStab (the latter of the product with $A^\top$ is also available)



19 Idea of ILU preconditioner

One step of LU factorization reduces matrix from size $n \times n$ to size $n-1 \times n-1$, but reduces sparsity: it adds coefficients in sparse matrices where previously was zero. ILU limits this influence. It can be better understood by looking sparse matrix A's corresponding adjacency graph (graph where 2 nodes are linked if and only if corresponding coefficient $A_{i,j}$ is not zero).

Then we eliminate one variable (vertex), new edges (called fill-in) appears between high-order neighbors. This new edge can appear only between vertices that had common neighbour: it means, that they are second-order neigbours.

The ILU(k) idea is to leave only the elements in L and U that are k-order neighbours in the original graph.

A much more popular approach is based on the so-called thresholded LU.

You do the standard Gaussian elimination with fill-ins, but either:

Throw away elements that are smaller than threshold, and/or control the amount of non-zeros you are allowed to store.

The smaller is the threshold, the better is the preconditioner, but more memory it takes.

It is denoted ILUT(τ).

20 Definition of Toeplitz matrix

A matrix is called Toeplitz if its elements are defined as

$$a_{ij} = t_{i-j}.$$

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & \vdots & \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & \ddots & a_1 & a_0 & a_{-1} & \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}.$$

- a Toeplitz matrix is completely defined by its first column and first row (i.e., $2n-1$ parameters).
- it is a dense matrix, however it is a structured matrix (i.e., defined by $\mathcal{O}(n)$ parameters).
- the main operation in the discrete convolution is the product of Toeplitz matrix by vector.

There is a special class of Toeplitz matrices, named circulant matrices for which fast matrix-by-vector product can be done (faster than $O(n^2)$).
A matrix C is called circulant, if

$$C_{ij} = c_{i-j \mod n},$$

i.e. it periodically wraps

$$C = \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \\ c_3 & c_0 & c_1 & c_2 \\ c_2 & c_3 & c_0 & c_1 \\ c_1 & c_2 & c_3 & c_0 \end{bmatrix}.$$

21 Fast Forier Transform and how it helps to multiply vector by Toeplitz matrix fast

Definition. The Fourier matrix is defined as:

$$F_n = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w_n^{1\cdot 1} & w_n^{1\cdot 2} & \dots & w_n^{1\cdot(n-1)} \\ 1 & w_n^2 \cdot 1 & w_n^2 \cdot 2 & \dots & w_n^2 \cdot (n-1) \\ 1 & w_n^{(n-1)} \cdot 1 & w_n^{(n-1)} \cdot 2 & \dots & w_n^{(n-1)\cdot(n-1)} \end{pmatrix}$$

or equivalently

$$F_n = \left\{ w_n^{kl} \right\}_{k,l=0}^{n-1}$$

where

$$w_n = e^{-\frac{2\pi i}{n}}$$

Properties:

- Symmetric (not Hermitian!)
- Unitary up to a scaling factor: $F_n^* F_n = F_n F_n^* = nI$ (check this fact). Therefore $F_n^{-1} = \frac{1}{n} F_n^*$
- Can be multiplied by a vector (called dicrete Fourier transform or DFT) with complexity $O(nlogn)$ (called FFT)

Let us have a circulant matrix C:

$$C = \begin{pmatrix} c_0 & c_{n-1} & c_{n-2} & \dots & c_1 \\ c_1 & c_0 & c_{n-1} & \dots & c_2 \\ c_2 & c_1 & c_0 & \dots & c_3 \\ c_{n-1} & c_{n-2} & c_{n-3} & \dots & \ddot{c}_0 \end{pmatrix}$$

Theorem. Let C be a circulant matrix of size $n \times n$ and let c be its first column, then it is represented in a form:

$$C = \frac{1}{n} F_n^* \operatorname{diag}(F_n c) F_n$$

where F is the Fourier matrix with the elements

$$F_{kl} = w_n^{kl}, \quad k, l = 0, \dots, n-1, \quad w_n = e^{-\frac{2\pi i}{n}}$$

and matrix $\Lambda = \operatorname{diag}(\lambda)$ is the diagonal matrix and

$$\lambda = Fc$$

where c is the first column of the circulant matrix C.

Proof. - Consider a number

$$\lambda(\omega) = c_0 + \omega c_1 + \ldots + \omega^{n-1} c_{n-1}$$

where ω is any number such that $\omega^n = 1$. - Lets multiply λ by $1, \omega, \dots, \omega^{n-1}$

$$\begin{aligned} \lambda &= c_0 + \omega c_1 && + \ldots + \omega^{n-1} c_{n-1} \\ \lambda\omega &= c_{n-1} + \omega c_0 && + \ldots + \omega^{n-1} c_{n-2} \\ \lambda\omega^2 &= c_{n-2} + \omega c_{n-1} && + \ldots + \omega^{n-1} c_{n-3} \\ & \ldots + \omega c_2 && + \ldots + \omega^{n-1} c_0 \end{aligned}$$

- Therefore,

$$\lambda(\omega) \cdot \begin{pmatrix} 1 & \omega & \dots & \omega^{n-1} \end{pmatrix} = \begin{pmatrix} 1 & \omega & \dots & \omega^{n-1} \end{pmatrix} \cdot C$$

- Writing this for $\omega = 1, w_n, \dots, w_n^{n-1}$ we get

$$\Lambda F_n = F_n C$$

and finally

$$C = \frac{1}{n} F_n^* \Lambda F_n, \qquad \text{where} \quad \Lambda = \operatorname{diag}(F_n c).$$

Representation $C = \frac{1}{n} F^* \operatorname{diag}(F_n c) F_n$ gives us an explicit way to multiply a vector x by C in $O(n \log n)$ operations. - Indeed,

$$Cx = \frac{1}{n} F_n^* \operatorname{diag}(F_n c) F_n x = \operatorname{ifft}(\operatorname{fft}(c) \circ \operatorname{fft}(x))$$

where $\circ$ denotes elementwise product (Hadamard product) of two vectors (since $\operatorname{diag}(a)b = a \circ b$) and ifft denotes inverse Fourier transform F_n^{-1}.

Basic Problems. Basics problems of NLA (debilnik)

Problem 1

Find the determinant of the following matrix: $A = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$
Solution:
The determinant is a scalar value that can be computed from the elements of a square matrix. In our case we have rectangular matrix 2×3. So, the determinant can not be computed.

Problem 2

Prove that the matrix xy^T has rank equals one
Solution:
The result matrix $A = xy^T$ is $n \times n$, where

$$a_1 = x_1 * y^T$$

$$a_2 = x_2 * y^T$$

$$...$$

$$a_n = x_n * y^T$$

where a_i - row A. So, we have n linear dependent rows and the rank of the matrix A is equal 1

Problem 3

Find at least one eigenpair of the matrix $A = \begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. How many eigenvalues are there?

Solution:
The matrix $A - \lambda I$, where λ is an eigenvalue of A, should have determinant equal to 0. We can use it to find λ.

$$\det\left(\begin{bmatrix} \sqrt{3}/2 - \lambda & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 - \lambda & 0 \\ 0 & 0 & 1 - \lambda \end{bmatrix}\right) = (\frac{\sqrt{3}}{2} - \lambda)^2(1 - \lambda) + \frac{1}{4}(1 - \lambda) = 0$$

.

$$((\frac{\sqrt{3}}{2} - \lambda)^2 + \frac{1}{4})(1 - \lambda) = 0$$

$$\lambda_1 = 1$$

$$(\frac{\sqrt{3}}{2} - \lambda)^2 + \frac{1}{4} = 0$$

$$\lambda^2 - \sqrt{3}\lambda + 1 = 0$$

$$D = 3 - 4 = -1$$

$$\lambda_{2,3} = \frac{\sqrt{3} \pm \sqrt{-1}}{2} = \frac{\sqrt{3}}{2} \pm \frac{1}{2}i$$

So, we have 3 eigenvalues, one real, two imaginary. Let's find an eigenvector for the real eigenvalue ($\lambda_1 = 1$). To do that we'll use $Av = \lambda v$:

$$\begin{bmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = 1 \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\begin{aligned}\frac{\sqrt{3}}{2}v_1 - \frac{1}{2}v_2 &= v_1\\ \frac{1}{2}v_1 + \frac{\sqrt{3}}{2}v_2 &= v_2\\ v_3 &= v_3\end{aligned}$$

Rearranging first two equations we get

$$\begin{aligned}v_1 &= \frac{1}{\sqrt{3}+2}v_2\\ v_1 &= (2-\sqrt{3})v_2\\ v_3 &= v_3\end{aligned}$$

Which is only possible if $v_1 = v_2 = 0$. So, we get an eigenvector

$$\begin{bmatrix}0\\0\\v_3\end{bmatrix}$$

Problem 4

Find pseudo-inverse of a scalar $(c)^\dagger$?

$$(c)^\dagger = \begin{cases} c^{-1} & \text{if c} \neq 0\\ 0 & \text{if c} = 0\end{cases} \tag{2}$$

Problem 5

Find pseudo-inverse of a vector $(a)^\dagger$?

$$(a)^\dagger = (a^*a)^\dagger a^* = \begin{cases} \frac{a^*}{a^*a} & \text{if a} \neq 0\\ 0 & \text{if a} = 0\end{cases} \tag{3}$$

Problem 6

If $a = 0$ or $b = 0$, then ab^T is a zero matrix, so anything could be a pseudoinverse. Otherwise $(a^Ta)^{-1}, (b^Tb)^{-1}$ exist.

$$\begin{aligned}(ab^\top)(ab^\top)^+(ab^\top) &= (ab^\top)(a^\top a)^+(b^\top b)^+ba^\top(ab^\top) &(4)\\ &= (a^\top a)^{-1}(b^\top b)^{-1}\, ab^\top ba^\top ab^\top &(5)\\ &= ab^\top. &(6)\end{aligned}$$

Problem 7

Find singular values of matrix $A = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$
Solution:
We use the fact that A^*A - Hermitian and non-negative define, so:

$$A^*AV = V\Sigma^2,$$

V - unitary matrix,
$\Sigma^2 = diag(\sigma_1^2, \sigma_2^2)$,
σ_1^2, σ_2^2 - eigenvalues of A^*A

$$A^*A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\Rightarrow \sigma_1^2 = \sigma_2^2 = 1$$

Hence, our singular values:

$$\sigma_1 = 1, \sigma_2 = 1$$

Problem 8

The given matrix is a rectangular. An eigenvector is defined only for square matrix, so it's impossible to find any eigenvector.

Problem 9

Householder matrix is:

$$P(v) = I - 2vv^* \tag{7}$$

Applied to the vector x, it's a reflection over the hyperplane that is perpendicular to the vector v. It is easy to understand the logic behind the formula by The Method of Gazing ©2:

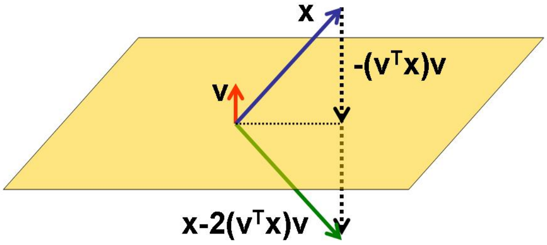


Figure 2: Householder reflection

Problem 10

Givens rotation matrix

In any multidimensional space we can choose 2 axes and rotate vector in the 2D plane spanned by these axes. Since we know the rotation matrix in 2D, it's easy to write Givens rotation matrix for the axes i, j for the angle θ:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{8}$$

Problem 11

Prove that $\left\|xy^\top\right\|_F = \left\|xy^\top\right\|_2 = \|x\|_2\|y\|_2$ for any $x, y \in \mathbb{C}^n$[1]

$$\|xy^T\|_F^2 = \mathrm{Tr}\big((xy^T)^* xy^T\big) = \mathrm{Tr}\big(\bar{y}x^* xy^T\big) = \mathrm{Tr}\big(x^* xy^T \bar{y}\big) = \|x\|_2^2\|y\|_2^2 \tag{9}$$

Essentially, action of matrix xy^T on vector v is just a scalar product (y, v) and multiplication on vector x. Since scalar product is maximal for parallel vectors (if len is fixed), supremum is reached for $v = \frac{y}{\|y\|}$

$$\|xy^T\|_2 = \sup_{\|v\|_2=1} \|xy^T v\|_2 = \|xy^T \frac{y}{\|y\|_2}\|_2 = \|x\|y\|_2\|_2 = \|x\|_2\|y\|_2 \tag{10}$$

Problem 12

Solve $\frac{\partial}{\partial a} x^\top a$ We use: $\frac{\partial x_j}{\partial x_i} = \delta_{ij}$

$$\left(\frac{\partial}{\partial a} x^T a\right)_i = \frac{\partial}{\partial a_i} x_j a_j = x_j \frac{\partial a_j}{\partial a_i} = x_j \delta_{ij} = x_i \to \frac{\partial}{\partial a} x^T a = x \tag{11}$$

Problem 13

Solve $\frac{\partial}{\partial A} \operatorname{tr}\left(A^\top B\right)$ We use the property: $\frac{\partial A_{kl}}{\partial A_{ij}} = \delta_{ik}\delta_{jl}$

$$\left(\frac{\partial}{\partial A} x^T a\right)_i = \frac{\partial}{\partial a_i} \sum_j x_j a_j = \sum_j x_j \frac{\partial a_j}{\partial a_i} = \sum_j x_j \delta_{ij} = x_i \to \frac{\partial}{\partial a} x^T a = x \tag{12}$$

$$\left(\frac{\partial}{\partial A} \mathrm{Tr}\, A^T B\right)_{ij} = \frac{\partial}{\partial A_{ij}} \sum_{kl} A_{kl} B_{kl} = \sum_{kl} \delta_{ik}\delta_{jl} B_{kl} = B_{ij} \to \frac{\partial}{\partial A} \mathrm{Tr}\, A^T B = B \tag{13}$$

Problem 14

Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \frac{1}{2}\|Ax - b\|_2^2$

$$(\nabla f(x))_i = \frac{\partial}{\partial x_i} \frac{1}{2} \|Ax - b\|_2 = \frac{\partial}{\partial x_i} \frac{1}{2} (Ax - b, Ax - b) = \frac{1}{2} \frac{\partial}{\partial x_i} ((Ax, Ax) - (Ax, b) - (Ax, b)^*) \tag{14}$$

$$\begin{aligned} \frac{\partial}{\partial x_i}(Ax, Ax) = \frac{\partial}{\partial x_i} \sum_{klm} A_{kl} x_l A_{km} x_m = \\ \sum_{klm} A_{kl} A_{km} (\delta_{il} x_m + \delta_{im} x_l) = \sum_{km} A_{ki} A_{km} x_m + \sum_{kl} A_{kl} A_{ki} x_l = 2(A^T A x)_i \end{aligned} \tag{15}$$

[1] It's easy to prove with use of rank-1 norms properties from hw1, but I don't think it would be allowed to use it.

$$\frac{\partial}{\partial x_i}(Ax, b) = \frac{\partial}{\partial x_i}\sum_{kl} A_{kl}x_l b_k = \sum_{kl} A_{kl}\delta_{il}b_k = \sum_k A_{ki}b_k = 2(A^T b)_i \tag{16}$$

For $A, b, x \in \mathbb{R}$:

$$\nabla f(x) = A^T(Ax - b) \tag{17}$$

For Hessian we are left only with the first term in 14

$$f''(x)_{ij} = \frac{\partial}{\partial x_j}(\nabla f(x))_i = \frac{\partial}{\partial x_j} =$$

$$\sum_{km} A_{ki}A_{km}\delta_{mj} + \sum_{kl} A_{kl}A_{ki}\delta_{lj} = \sum_k A_{ki}A_{kj} + \sum_k A_{kj}A_{ki} = 2(A^T A)_{ij} \tag{18}$$

$$f''(x) = 2A^T A \tag{19}$$

Problem 15

Show that if matrix is triangular and unitary, then it is diagonal.

Suppose A is, say, upper triangular

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ & a_{22} & \dots & a_{2m} \\ & & \ddots & \vdots \\ & & & a_{mm} \end{bmatrix}$$

The (i, j) -entry of A^*A is the inner product of the i th column of A with the jth column of A, and since A is unitary, the (i, j) -entry of A^*A is 1 if $i = j$ and 0 otherwise. Consider the first row of A^*A :

$$(A^*A)_{11} = |a_{11}|^2 = 1$$
$$(A^*A)_{12} = \bar{a}_{11}a_{12} = 0 \Rightarrow a_{12} = 0, \dots (A^*A)_{1m} = \bar{a}_{11}a_{1m} = 0 \Rightarrow a_{1m} = 0$$

We now know that A has the form

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ & a_{22} & \dots & a_{2m} \\ & & \ddots & \vdots \\ & & & a_{mm} \end{bmatrix}$$

Now consider the second row of A^*A :

$$(A^*A)_{22} = |a_{22}|^2 = 1$$
$$(A^*A)_{23} = \bar{a}_{22}a_{23} = 0 \Rightarrow a_{23} = 0, \dots (A^*A)_{2m} = \bar{a}_{22}a_{2m} = 0 \Rightarrow a_{2m} = 0$$

Proceeding like this, by rows, we see that only the diagonal entries of A are nonzero and they each have magnitude 1; that is, A has the form

$$A = \begin{bmatrix} e^{i\theta_1} & & \\ & \ddots & \\ & & e^{i\theta_m} \end{bmatrix}$$

for some $\theta_1, \dots, \theta_m$ If A is lower triangular,

$$A = \begin{bmatrix} a_{11} & & \\ \vdots & \ddots & \\ a_{m1} & \cdots & a_{mm} \end{bmatrix}$$

then we can do the same thing, establishing zeros below the diagonal in each row, starting at the bottom. since the last column of A has only one nonzero entry, which must satisfy $|a_{mm}|^2 = 1$ so that the (m, m) entry of A^*A will be 1, and the inner product of the last column with column j is $\bar{a}_{mm}a_{mj}$, we find $a_{mj} = 0$ for $j < m$. We then proceed to row $m - 1$, etc.