Dr. Christian Vollmann

Winter Term 2022

[February 15, 2023]

Numerical Linear Algebra

all exercises

Due date: –

Name:

Matriculation number:

Remarks

- Upload your solutions to the respective Homework directory on olat:

  Homework/`<sheetnumber>`/Participant Drop Box

- Follow these guidelines:

	Theory	**Programming**
Identifier	no gray tag "Python"	gray tag "Python"
Group work	no	yes (up to 3, but not mandatory!)
Upload format	.pdf	.ipynb
Filename	`sheetnumber_Name.pdf`	`sheetnumber_Name1Name2Name3.ipynb`

1 Math Basics

Ex 1 Math Basics

Binomial Coefficient The binomial coefficient is defined by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

where $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$. Pascal's triangle states the equation

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$$

for any $0 \leq k < n$. Please prove this equation.

Solution:

$$\binom{n}{k} + \binom{n}{k+1} = \frac{n!}{k!(n-k)!} + \frac{n!}{(k+1)!(n-(k+1))!} = \frac{n!(k+1) + n!(n-k)}{(k+1)!(n-k)!} = \frac{n!k + n! + n!\,n - n!\,k}{(k+1)!(n-k)!} = \frac{(n+1)!}{(k+1)!(n+1-1-k)!} = \binom{n+1}{k+1}$$

Ex 2 Math Basics

Complex Numbers
Let i be the imaginary unit, i.e., $i^2 = -1$. Please cast the following complex numbers into the format $z = x + iy$.

1. $(4+5i)(4-5i)$
2. $\frac{2+3i}{4+5i}$
3. $\sqrt{16b} + \sqrt{3a - 12a}$

Solution:

1. $(4+5i)(4-5i) = 4^2 - (5i)^2 = 16 + 25 = 41$
2. $\frac{2+3i}{4+5i} = \frac{(2+3i)(4-5i)}{41} = \frac{8-10i+12i-15i^2}{41} = \frac{23+2i}{41}$
3. $\sqrt{16b} + \sqrt{3a - 12a} = 4\sqrt{b} + \sqrt{-9a} = 4\sqrt{b} + i3\sqrt{a}$

Ex 3

Show by induction that for any $n \in \mathbb{N}$ it holds that

$$\sum_{k=0}^{n} (2k+1) = (n+1)^2.$$

Solution:

Show: $\sum_{k=0}^{n}(2k+1) = (n+1)^2$
Proof:
Induction Basis $(n=0)$

$$2\cdot 0+1 = 1 = (0+1)^2 \ \checkmark \quad (2P)$$

Induction Step $(n \mapsto n+1)$

$$\sum_{k=0}^{n+1}(2k+1) = 2(n+1)+1+\sum_{k=0}^{n}(2k+1) \overset{\text{[I.A.]}(2P)}{=} 2(n+1)+1+(n+1)^2$$
$$=2n+2+1+n^2+2n+1 = ((n+1)+1)^2 \ \checkmark \quad (4P)$$

Ex 4

Show by induction that for any $n \in \mathbb{N}$ it holds that

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

Solution:

Show: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
Proof:
Induction Basis $(n=1)$

$$1 = 2 = \frac{1(1+1)}{2} \ \checkmark \quad (2P)$$

Induction Step $(n \mapsto n+1)$

$$\sum_{k=1}^{n+1} k = (n+1)+\sum_{k=1}^{n} k \overset{\text{[I.A.]}(2P)}{=} (n+1)+\frac{n(n+1)}{2} = (n+1)+\left(\frac{n}{2}+1\right) = \frac{(n+1)(n+2)}{2} \ \checkmark \quad (4P)$$

Ex 5

Show by induction that for any $n \in \mathbb{N}$ it holds that

$$\sum_{k=1}^{n} k(k+1) = \frac{1}{3}n(n+1)(n+2).$$

Solution:

Show: $\sum_{k=1}^{n} k(k+1) = \frac{1}{3}n(n+1)(n+2)$
Proof:
Induction Basis $(n=1)$

$$1(1+1) = 2 = \frac{1}{3}1(1+1)(1+2) \ \checkmark \quad (2P)$$

Induction Step $(n \mapsto n+1)$

$$\sum_{k=1}^{n+1} k(k+1) = (n+1)(n+1+1) + \underbrace{\sum_{k=1}^{n} k(k+1)}_{\overset{\text{[I.A.]}(2P)}{=} \frac{1}{3}n(n+1)(n+2)}$$
$$= (n+1)(n+2)\underbrace{\left(1+\frac{1}{3}n\right)}_{=\frac{1}{3}(3+n)}$$
$$= \frac{1}{3}(n+1)((n+1)+1)((n+1)+2) \ \checkmark \quad (4P)$$

Ex 6

Proof via Induction
Please prove the following assertions by induction.

1. $\sum_{k=0}^{n}(2k+1) = (n+1)^2$
2. $2^n = \sum_{k=0}^{n} \binom{n}{k}$
3. Let $q \neq 1$ then

$$\sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}.$$

Hint: Use Exercise 12 for (ii).

Solution:

1. **Induction Basis** $(n=0)$

$$\sum_{k=0}^{0}(2k+1) = 1 = (0+1)^2.$$

Hence, *induction hypothesis* can be used.
Induction Step $(n \mapsto n+1)$

$$\sum_{k=0}^{n+1}(2k+1) = (2(n+1)+1) + \sum_{k=0}^{n}(2k+1) = (2(n+1)+1) + (n+1)^2 =$$
$$= 2n+2+1+n^2+2n+1 = n^2+4n+4 = (n+2)^2$$

$\square$

2. **Induction Basis** $(n=0)$

$$\sum_{k=0}^{0}\binom{n}{k} = \binom{0}{0} = 1 = 2^0.$$

Induction Step $(n \mapsto n+1)$

$$\begin{aligned}\sum_{k=0}^{n+1}\binom{n+1}{k} &= \sum_{k=0}^{n+1}\left[\binom{n}{k+1} + \binom{n}{k}\right] = \\ &= \frac{n!}{k!(n-k)!} + \frac{n!}{(k+1)!(n-(k+1))!} = \frac{n!(k+1)+n!(n-k)}{(k+1)!(n-k)!} \\ &= \frac{n!+n!n}{(k+1)!(n-k)!} = \frac{(n+1)!}{(k+1)!(n+1-(k+1))!} = \binom{n+1}{k+1}.\end{aligned}$$

3. **Induction Basis** $(n=0)$
$\sum_{k=0}^{0} q^k = 1 = \frac{q^1-1}{q-1}$.
Induction Step $(n \mapsto n+1)$

$$\sum_{k=0}^{n+1} q^k = \frac{q^{n+2}-1}{q-1} + \sum_{k=0}^{n} q^k = q^{n+1} + \frac{q^{n+1}-1}{q-1} = \frac{q^{n+2}-1}{q-1}.$$

Ex 7 Math Basics

Solution Sets
Please indicate the solution set of the following inequalities.

1. $\frac{3-x}{2} < 5$
2. $3 - (x+1)^2 \geq 2 + \frac{2}{3}x$
3. $5(x + 3(1-x) + 2) \geq 7$
4. $x + 2 \geq -\frac{1}{x}$

Solution:

1. $\mathbb{L} = \{x | -7 < x\}$
2. $\mathbb{L} = \{x | -\frac{8}{3} \leq x \leq 0\}$
3. $\mathbb{L} = \{x | x \leq -\frac{9}{5}\}$
4. $\mathbb{L} = \{x | x = -1 \vee x > 0\}$

Ex 8 Math Basics

Result for Injective Funtions
Let $f : X \to Y$ be an injective function and $A, B \subset X$. Please show that

$$f(A) \cap f(B) = f(A \cap B).$$

Hint: Split up the equality "$=$" into the parts "$\subset$" and "$\supset$". One direction is straightforward the other one requires, that f is injective.

Solution:

- $\underline{f(A \cap B) \subset f(A) \cap f(B):}$
  We have
  $$f(A \cap B) = f(A \cap B) \cap f(B \cap A) \subset f(A) \cap f(B).$$
- $\underline{f(A) \cap f(B) \subset f(A \cap B):}$
  Let $y \in f(A) \cap f(B)$,
  $\Rightarrow \quad \exists\, x_a \in A, x_b \in B : \; f(x_a) = y = f(x_b).$

  Since f is injective, we find $x_a = x_b$
  $\Rightarrow \quad x_a, x_b \in A \cap B$
  $\Rightarrow \quad y \in f(A \cap B).$

Ex 9 Math Basics

Irrational $\sqrt{2}$
Please show that $\sqrt{2}$ is not a rational number.

Solution:

Proof We prove the assumption by contradiction. Let us assume $\sqrt{2} \in \mathbb{Q}$. Therefore there are $a, b \in \mathbb{N}$ such that a, b are relatively prime (i.e. their greatest common divisor is 1) and $\sqrt{2} = \frac{b}{a}$.
We see that $2a^2 = b^2$ which implies that b^2, and therefore b, are even. That means there is a $c \in \mathbb{N}$ such that $b = 2c$. This implies that $2a^2 = 4c^2$ which tells us that $a^2 = 2c^2$. Hence, a^2 and therefore a are even. That contradicts the assumption that a, b are relatively prime. Therefore $\sqrt{2}$ is not an element of $\mathbb{Q}$. $\square$

Ex 10 Math Basics

Logical Negation
Please state the negation of the following statements.

1. It rains.
2. It rains and it is cold outside.
3. $A \wedge B$ (*Hint:* De Morgan)
4. $x = 3$ or $x = 1$
5. $A \vee B$
6. If it rains the street gets wet.
7. $A \Rightarrow B$

Solution:

1. It does not rain.
2. It either does not rain or it is not cold outside.
3. $\neg A \vee \neg B$
4. $x \neq 3$ and $x \neq 1$
5. $\neg A \wedge \neg B$
6. It rains and the street is not wet.
7. $A \wedge \neg B$

Ex 11 Math Basics

PQFormula
Let $a, b \in \mathbb{R}$ and $a \neq 0$. Please solve the following quadratic equations.

1. $x^2 + bx - 2b^2 = 0$
2. $3a^2x^2 + 4ax + 1 = 0$
3. $ax^2 + ax + x + 1 = 0$

Solution:

Formula

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

1.

$$x_{1,2} = -\frac{b}{2} \pm \sqrt{\left(\frac{b}{2}\right)^2 + 2b^2} =$$
$$-\frac{b}{2} \pm \sqrt{\frac{b^2 + 8b^2}{4}} = -\frac{b}{2} \pm b\frac{3}{2} = \begin{cases} b \\ -2b \end{cases}$$

2.

$$x_{1,2} = \begin{cases} -\frac{1}{3a} \\ -\frac{1}{a} \end{cases}$$

3.

$$x_{1,2} = \begin{cases} -\frac{1}{a} \\ -1 \end{cases}$$

Ex 12

Heron's algorithm
For a nonnegative number $a \geq 0$ compute the square root $\sqrt{a}$ up to an error of 10^{-10}. For this purpose, use the following iteration:

$$x_{n+1} = \frac{1}{2}\left(\frac{x_n^2 + a}{x_n}\right).$$

Choose different initial values x_0 and observe the convergence behavior by printing the error $|x_n - \sqrt{a}|$ in each iteration step.
Hint: Use a while-loop for the iteration (`while`) and the built-in function abs as well as the Python value `a ** 0.5` to compute the error $|x_n - \sqrt{a}|$ in each iteration step.

Solution:

```
def heron(a, x0=0.1, tol=10e-10, maxiter=1000):
    """
    applies the iteration rule according to the so-called "Heron method"
    """
    counter = 0
    # while the error is above our tolerance we do the iteration
    while abs(x0 - a**0.5) > tol:
        # print the current error
        print("Error =", abs(x0 - a**0.5))
        # perfom one step of the heron method:
        x0 = 0.5 * ((x0 ** 2 + a) / x0)
        # update the counter
        counter += 1
        if counter > maxiter:
            # stop the while loop if too many iterations
            break
    print("\n Result: sqrt(a) =", x0, "\n")
    return x0


if __name__ == "__main__":
    a = 2.

    # Choose different initial values
    x0 = [0.1, 1., 100000, 1.3]
    for x in x0:
        print("\n x0 =", x, "\n---------------\n")
        heron(a, x0=x)

    help(heron)
```

Ex 13 Math Basics

Set Operations
Let $M = \{\alpha, 3, \gamma\}, \ N = \{\square, \gamma, \star, \circ\}$ and $K = \{\alpha, \{\alpha\}, M\}$. Please indicate the sets

1. $M \cap N$,

2. $M \cap K$,
3. $\mathcal{P}(M) \cap K$,
4. $M \setminus K$ and
5. $N \cup K$.

Solution:

1. $M \cap N = \{\gamma\}$
2. $M \cap K = \{\alpha\}$
3. $\mathcal{P} \cap K = \{\{\alpha\}, M\}$
4. $M \setminus K = \{3, \gamma\}$
5. $N \cup K = \{\square, \gamma, \star, \circ, \alpha, \{\alpha\}, M\}$

Ex 14 Math Basics

Sets and Functions

Let A, B be finite sets which contain the same number of elements, i.e., $|A| = |B|$ and let $f : A \to B$ be a function. Please show that the following statements are equivalent.

1. f is injective
2. f is surjective
3. f is bijective

Hint: (iii) follows immediately from (i) $\Leftrightarrow$ (ii).

Solution:

- (i) $\Rightarrow$ (ii)

$$\begin{aligned}
i) &\Rightarrow \forall a_1, a_2 \in A : a_1 \neq a_2 \Rightarrow f(a_1) \neq f(a_2) \\
&\Rightarrow |f(A)| = |A| \quad \text{(note: } A = \{a | a \in A\}, f(A) = \{f(a) | a \in A\}) \\
&\Rightarrow |f(A)| = |B| \quad \text{(since } |A| = |B|) \\
&\Rightarrow f(A) = B \quad \text{(since } f(A) \subset B \text{ and } B \text{ finite)} \\
&\Rightarrow ii)
\end{aligned}$$

- (ii) $\Rightarrow$ (i)
  We know by surjectivity and $|A| = |B|$:

$$f(A) = B \quad \Rightarrow \quad |f(A)| = |B| \quad \Rightarrow \quad |f(A)| = |A| \quad (+)$$

  Now let $a_1 \neq a_2 \in A$ (to show $f(a_1) \neq f(a_2)$)
  Assumption: $f(a_1) = f(a_2)$, then

$$f(A) = \{f(a) \colon a \in A\} \subsetneq B \quad \Rightarrow \quad |f(A)| < |B| \quad \Rightarrow \quad |A| < |B| \quad \text{“contradiction to (+) !"}$$

- (iii) $\Leftrightarrow$ (i) (or (ii))
  By definition we have

$$(iii) \Leftrightarrow (i) \wedge (ii) \Leftrightarrow (i)$$

Ex 15 Math Basics

Subset Property

Please show that the following assertions are equivalent.

1. $A \subset B$
2. $A \cap B = A$
3. $A \cup B = B$

Hint: It suffices to show $(i) \Rightarrow (ii) \Rightarrow (iii) \Rightarrow (i)$.

Solution:

- (i) $\Rightarrow$ (ii)
  $A \cap B \subset A$ (No need for assumptions.)

$$\begin{aligned} & x \in A \cap B \\ \Rightarrow & x \in A \wedge x \in B \\ \Rightarrow & x \in A \end{aligned}$$

  $A \subset A \cap B$

$$\begin{aligned} & A \subset B \\ \Rightarrow & x \in A \Rightarrow x \in B \\ \Rightarrow & x \in A \Rightarrow x \in A \wedge x \in B \\ \Rightarrow & A \subset A \cap B \end{aligned}$$

- (ii) $\Rightarrow$ (iii)
  $B \subset A \cup B$ (Again, no need for assumptions.)

$$\begin{aligned} & x \in B \\ \Rightarrow & x \in A \vee x \in B \\ \Rightarrow & x \in A \cup B \end{aligned}$$

  $A \cup B \subset B$

$$\begin{aligned} & x \in A \cup B \\ \Rightarrow & x \in A \vee x \in B \\ \Rightarrow & (x \in A \wedge x \in B) \vee x \in B \quad \text{(since by ii): } A \subset A \cap B) \\ \Rightarrow & x \in (A \cap B) \cup B \\ \Rightarrow & x \in B \end{aligned}$$

- (iii) $\Rightarrow$ (i)

$$\begin{aligned} & x \in A \\ \Rightarrow & x \in A \vee x \in B \\ \Rightarrow & x \in A \cup B \\ \Rightarrow & x \in B, \quad \text{(since by iii): } A \cup B \subset B) \end{aligned}$$

Ex 16 Math Basics

Symmetric Difference
Let X be a set and $A, B \subset X$ be two subsets of X. The symmetric difference of A and B is then defined by

$$A \Delta B := (A \cap B^c) \cup (B \cap A^c).$$

Please show that

$$A \Delta B = (A \cup B) \cap (A \cap B)^c.$$

Hint: Use the De Morgan's rule and exploit the following distributive laws:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad \text{and} \quad A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

Solution:

$$\begin{aligned}
&(A\cap B^c)\cup(B\cap A^c)\\
=&((A\cap B^c)\cup B)\cap((A\cap B^c)\cup A^c)\\
=&\underbrace{((A\cup B)\cap(B^c\cup B))}_{A\cup B}\cap\underbrace{((A\cup A^c)\cap(B^c\cup A^c))}_{B^c\cup A^c}\\
=&(A\cup B)\cap(B^c\cup A^c)\\
=&(A\cup B)\cap(A\cap B)^c
\end{aligned}$$

Ex 17 Math Basics

Triangle Inequality
Let $x, y \in \mathbb{R}$. Please show the triangle inequality and the reverse triangle inequality.

1. $|x+y| \leq |x| + |y|$
2. $|x-y| \geq \big|\,|x| - |y|\,\big|$

Solution:

1. First we note that as

$$|x| = \begin{cases} x : x \geq 0 \\ -x : x < 0 \end{cases}$$

$x \leq |x|$.
Case 1 $(x = -y)$: $\surd$
Case 2 $(x > -y)$:

$$|x+y| = x + y \leq |x| + |y|.$$

Case 3 $(x < -y \Leftrightarrow -x > -(-y))$:

$$|x+y| = |-x-y| = -x-y \leq |-x| + |-y| = |x| + |y|.$$

2.

$$\begin{aligned}
|x| = |x-y+y| &\leq |x-y| + |y| \\
\Leftrightarrow |x| - |y| &\leq |x-y|
\end{aligned}$$

By interchanging x and y we hence obtain the assertion.

Ex 18 Math Basics

VennDiagrams
Please draw Venn-Diagrams which illustrate the following statements.

1. (Commutative property)
   $A \cup B = B \cup A$
   $A \cap B = B \cap A$
2. (Associative property)
   $(A \cup B) \cup C = A \cup (B \cup C)$
   $(A \cap B) \cap C = A \cap (B \cap C)$
3. (Distributive property)
   $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
   $(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$

Solution:

2 Linear Algebra

Ex 19 Linear Algebra

Copper, Silver and Gold Alloys
Let us assume we are given the alloys M_1, M_2 and M_3, which are built of copper, silver and gold with the following percental proportions:

	M_1	M_2	M_3
Copper	20	70	50
Silver	60	10	50
Gold	20	20	0

Is it possible to mix these alloys to form a new alloy which consists of 40% copper, 50% silver and 10% gold.

Hint: Cast the problem into a linear system $Ax = b$.

Solution:

We want to find $M_4 = x_1M_1 + x_2M_2 + x_3M_3$, so that $M_4 = \begin{pmatrix} 40 \\ 50 \\ 10 \end{pmatrix}$:

$$\left(\begin{array}{ccc|c} 20 & 70 & 50 & 40 \\ 60 & 10 & 50 & 50 \\ 20 & 20 & 0 & 10 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\overset{\text{(II)}'=\text{(II)}-3\text{(I)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} 20 & 70 & 50 & 40 \\ 3 & -200 & -100 & -70 \\ 20 & 20 & 0 & 10 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)}' \\ \text{(III)} \end{array} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\overset{\text{(III)}'=\text{(III)}-\text{(I)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} 20 & 70 & 50 & 40 \\ 3 & -200 & -100 & -70 \\ 1 & -50 & -50 & -30 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)}' \\ \text{(III)}' \end{array} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\overset{\text{(III)}''=\text{(III)}'-\frac{50}{200}\text{(II)}'}{\rightsquigarrow} \left(\begin{array}{ccc|c} 20 & 70 & 50 & 40 \\ 3 & -200 & -100 & -70 \\ 1 & \frac{1}{4} & -25 & -12,5 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)}' \\ \text{(III)}'' \end{array} \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$$

$$\text{(III)}'' \quad \Rightarrow \quad -25x_3 = -12,5 \quad \Rightarrow \quad x_3 = \frac{1}{2}$$

$$\text{(II)}' \quad \Rightarrow \quad -200x_2 - 100(\frac{1}{2}) = -70 \quad \Rightarrow \quad x_2 = -\frac{1}{200}(-70+50) = 0.1$$

$$\text{(I)} \quad \Rightarrow \quad 20x_1 + 70 \cdot 0,1 + 50\frac{1}{2} = 40 \quad \Rightarrow \quad x_1 = \frac{1}{20}8 = 0,4$$

Answer:Yes!, $x = \begin{pmatrix} 0,4 \\ 0,1 \\ 0,5 \end{pmatrix}$

Ex 20

Let $A \in \mathbb{R}^{n\times n}$ be matrix for which $A^\top = -A$ (*antisymmetric*). Show that the diagonal entries are zero, i.e., $a_{ii} = 0$ for all $1 \le i \le n$.

Solution:

From $A^\top = -A$ it follows that

$$a_{ii} = -a_{ii}$$

for all $1 \le i \le n$. This equation is only true for the number 0.

Ex 21

What is the determinant of

$$A = \begin{pmatrix} 0 & -e^{-i\pi} & i^2 & 0 & -\pi \\ e^{-i\pi} & 0 & -2 & 0 & 0 \\ 1 & 2 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -3 \\ \pi & 0 & 0 & 3 & 0 \end{pmatrix}?$$

Solution:

$$A = \begin{pmatrix} 0 & -e^{-i\pi} & i^2 & 0 & -\pi \\ e^{-i\pi} & 0 & -2 & 0 & 0 \\ 1 & 2 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -3 \\ \pi & 0 & 0 & 3 & 0 \end{pmatrix}$$

$$\begin{aligned} & \text{since } i^2 = -1,\ A^T = -A \quad (1P) \\ \Rightarrow\ & \det(A) = \det(A^T) = \det(-A) = (-1)^5 \det(A) \quad (1P) \\ \Rightarrow\ & \det(A) = 0 \quad (1P) \end{aligned}$$

Ex 22 Linear Algebra, Determinant

Compute Determinants

Compute the determinants of the following matrices and check whether they are invertible.

$$A = \begin{pmatrix} 1 & \pi & 2 & 12 \\ 0 & \frac{1}{5} & \frac{1}{\sqrt{2}} & 17 \\ 0 & 0 & 5 & \frac{1}{3} \\ 0 & 0 & 0 & 4 \end{pmatrix}, \qquad B = \begin{pmatrix} 5 & 3 \\ 1 & -2 \end{pmatrix}, \qquad C = \begin{pmatrix} 0 & \frac{1}{2} & 2 \\ -\frac{1}{2} & 0 & 7 \\ -2 & -7 & 0 \end{pmatrix}.$$

Solution:

1. $\det(A) \overset{[A \text{ is upper triangular}]}{=} 1 \cdot \frac{1}{5} \cdot 5 \cdot 4 \ = \ 4$
2. $\det(B) \ = \ 5 \cdot (-2) - 3 \cdot 1 \ = \ -13$
3.

$$\det(C) = \det\left(\begin{pmatrix} 0 & \frac{1}{2} & 2 \\ -\frac{1}{2} & 0 & 7 \\ -2 & -7 & 0 \end{pmatrix}\right) \overset{[\text{Sarrus' Rule}]}{=} (0 \cdot 0 \cdot 0) + \left(\frac{1}{2} \cdot 7 \cdot (-2)\right) + \left(2 \cdot \left(-\frac{1}{2}\right) \cdot (-7)\right)$$
$$- (-2 \cdot 0 \cdot 2) - (-7 \cdot 7 \cdot 0) - \left(0 \cdot \left(-\frac{1}{2}\right) \cdot \frac{1}{2}\right) \ = \ 0$$

Ex 23 Linear Algebra

Examples for (skew-)symmetric Matrices

Give examples for symmetric and skew-symmetric (exercise **??**) matrices $A \in \mathbb{R}^{2\times 2}$ and discuss their geometric behavior.

Solution:

Ex 24 Linear Algebra

Towards SVD: Properties of $A^T A$ and AA^T

Let $A \in \mathbb{R}^{m\times n}$ be any matrix. Please show:

1. $A^T A \in \mathbb{R}^{n\times n}$ and $AA^T \in \mathbb{R}^{m\times m}$ are symmetric.

2. A^TA and AA^T are positive semi-definite. (*Hint:* $\|x\|_2^2 = x^Tx \geq 0 \;\; \forall x \in \mathbb{R}^n$.)
   Remark: Thus, eigenvalues are nonnegative.

3. A^TA and AA^T have the same positive eigenvalues.

4. $\ker(A) = \ker(A^TA)$ and $\ker(A^T) = \ker(AA^T)$.
   Remark: Thus, $v \in \ker(A)$ is eigenvector of A^TA ($u \in \ker(A^T)$ is eigenvector of AA^T) to the eigenvalue $\lambda = 0$.

5. Name two sufficient conditions for the invertibility of A^TA and AA^T.

Solution:

1. *Recall: A matrix B is called symmetric, if $B^T = B$ holds.*
   Here, we find
   $$(A^TA)^T \;=\; A^T(A^T)^T \;=\; A^TA.$$
   and
   $$(AA^T)^T \;=\; (A^T)^TA^T \;=\; AA^T.$$

2. *Recall: A matrix B is called positive semi-definite, if $x^TBx \geq 0$ holds $\forall x \in \mathbb{R}^n$.*
   Here we find
   $$x^T(A^TA)x \;=\; (Ax)^TAx \;=\; \|Ax\|_2^2 \;\geq\; 0 \;\; \forall x \in \mathbb{R}^n.$$
   Next define $C := A^T$ and apply the latter result to C; note that $C^TC = AA^T$.

3. We show mutual subset relation:
   <u>**"$\subset$":**</u> We first show that all positive eigenvalues of A^TA are positive eigenvalues of AA^T:
   Let $\lambda > 0$ be an eigenvalue of A^TA with eigenvector $v \neq 0$, then we find
   $$A^TAv = \lambda v \;\overset{A\cdot|}{\Rightarrow}\; AA^T(Av) = \lambda(Av).$$
   In addition we find that $Av \neq 0$, since
   $$\|Av\|_2^2 = v^TA^TAv = \lambda v^Tv = \underbrace{\lambda}_{>0}\underbrace{\|v\|_2^2}_{>0} > 0.$$
   Thus we can conclude that λ is an eigenvalue of AA^T with eigenvector $Av =: u \neq 0$.

   <u>**"$\supset$":**</u> Next, we show that all positive eigenvalues of AA^T are also positive eigenvalues of A^TA:
   Let $\lambda > 0$ be an eigenvalue of AA^T with eigenvector $u \neq 0$, then we find
   $$AA^Tu = \lambda u \;\overset{A^T\cdot|}{\Rightarrow}\; A^TA(A^Tu) = \lambda(A^Tu).$$
   In addition we find that $A^Tu \neq 0$, since
   $$\|A^Tu\|_2^2 = u^T\underbrace{AA^Tu}_{\lambda u} = \underbrace{\lambda}_{>0}\underbrace{\|u\|_2^2}_{>0} > 0.$$
   Thus we can conclude that λ is an eigenvalue of A^TA with eigenvector $A^Tu =: v \neq 0$.

4. We show mutual subset relation:
   (a) $\ker(A) = \ker(A^TA)$:

   <u>"ker(A) $\subseteq$ ker(A^TA)":</u>
   Let $x \in$ ker(A) $\overset{\text{Def. ker}(A)}{\Rightarrow} Ax = 0 \Rightarrow A^TAx = 0 \overset{\text{Def. ker}(A^TA)}{\Rightarrow} x \in$ ker(A^TA).

   <u>"ker(A^TA) $\subseteq$ ker(A)":</u>
   Let $x \in$ ker(A^TA) $\overset{\text{Def.}}{\Rightarrow} A^TAx = 0 \Rightarrow \underbrace{x^TA^TAx}_{=\|Ax\|_2^2} = 0 \overset{\text{norm}\|\cdot\|_2^2 \text{ is definite}}{\Rightarrow} Ax = 0 \overset{\text{Def.}}{\Rightarrow} x \in$ ker(A).

   (b) $\ker(A^T) = \ker(AA^T)$:
   Define $C := A^T$ and apply result (a) to C; note that $C^TC = AA^T$.

5. For example:

i) Let A have independent columns (“full column rank”).
This is equivalent to $\underbrace{\text{ker}(A)}_{=\text{ker}(A^TA)} = \{0\}$, thus also the columns of A^TA are independent, which implies that A^TA is invertible.

ii) Let A^TA be positive definite.
Then its eigenvalues are strictly positive. Since A^TA is symmetric we can use its eigendecomposition to conclude that A^TA is invertible.

Ex 25

Let $A \in \mathbb{R}^{m\times n}$ be any matrix. Please show:

1. A^TA is symmetric.
2. A^TA is positive semi-definite.
3. $\ker(A) = \ker(A^TA)$.
   Hint: Show mutual subset relation.

Solution:

1. (2P) $(A^TA)^T \quad = \quad A^T(A^T)^T \quad = \quad A^TA$
2. (2P) $x^T(A^TA)x \quad = \quad (Ax)^TAx \quad = \quad \|Ax\|_2^2 \quad \geq \quad 0 \quad \forall x \in \mathbb{R}^n$
3. We show mutual subset relation:
   - (1P) "$\text{ker}(A) \subseteq \text{ker}(A^TA)$":
     Let $x \in \text{ker}(A) \overset{\text{Def. ker}(A)}{\Rightarrow} Ax = 0 \Rightarrow A^TAx = 0 \overset{\text{Def. ker}(A^TA)}{\Rightarrow} x \in \text{ker}(A^TA)$.
   - (1P) "$\text{ker}(A^TA) \subseteq \text{ker}(A)$":
     Let $x \in \text{ker}(A^TA) \overset{\text{Def.}}{\Rightarrow} A^TAx = 0 \Rightarrow \underbrace{x^TA^TAx}_{=\|Ax\|_2^2} = 0 \overset{\text{norm}\|\cdot\|_2^2 \text{ is definite}}{\Rightarrow} Ax = 0 \overset{\text{Def.}}{\Rightarrow} x \in \text{ker}(A)$.

Ex 26

Properties of A^TA and AA^T
Let $A \in \mathbb{R}^{m\times n}$ be any matrix. Please show: Name two sufficient conditions for the invertibility of A^TA and AA^T.

Solution:

1. For example:
   i) Let A have independent columns ("full column rank").
   This is equivalent to $\underbrace{\text{ker}(A)}_{=\text{ker}(A^TA)} = \{0\}$, thus also the columns of A^TA are independent, which implies that A^TA is invertible.

   ii) Let A^TA be positive definite.
   Then its eigenvalues are strictly positive. Since A^TA is symmetric we can use its eigendecomposition to conclude that A^TA is invertible.

Ex 27 Linear Algebra

The Subspaces Kernel and Image

1. Let $A \in \mathbb{F}^{m\times n}$. Show that $\ker(A)$ and $\text{Im}(A)$ are subspaces of $\mathbb{F}^n$ and $\mathbb{F}^m$, respectively.
2. Construct two example matrices and consider their kernel and image.

Solution:

1. a) To show: $\ker(A) \subset \mathbb{F}^n$ subspace

   Proof:

   i. $A \cdot 0 = 0$, so that $0 \in \ker(A)$, thus $\ker(A) \neq \emptyset$.

   ii. For $i = 1,2$ let $\lambda_i \in \mathbb{F},\ v_i \in \ker(A),$ then by linearity $A(\lambda_1 v_1 + \lambda_2 v_2) = \lambda_1 \underbrace{Av_1}_{=0} + \lambda_2 \underbrace{Av_2}_{=0} = 0$

   $\Rightarrow \quad \lambda_1 v_1 + \lambda_2 v_2 \in \ker(A)$

   b) To show: $\mathrm{Im}(A) \subset \mathbb{F}^m$ subspace

   Proof:

   i. $A \cdot 0 = 0 \in \mathrm{Im}(A)$, thus nonempty.

   ii. For $i = 1,2$ let $\lambda_i \in \mathbb{F},\ w_i \in \mathrm{Im}(A)$, then

$$\begin{aligned} & \exists v_1, v_2 \in \mathbb{F}^n : \ w_1 = Av_1,\ w_2 = Av_2 \\ \Rightarrow \quad & \lambda_1 w_1 + \lambda_2 w_2 = \lambda_1 Av_1 + \lambda_2 Av_2 = A(\lambda_1 v_1 + \lambda_2 v_2) \\ \Rightarrow \quad & \lambda_1 w_1 + \lambda_2 w_2 \in \mathrm{Im}(A) \end{aligned}$$

2. Examples:

   a) Consider the matrix composed of ones from the previous exercise.

   b) Let $A = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$. Then $Ax = 0$ implies $x = 0$, so that $\ker(A) = \{0\}$. In particular, the columns are linearly independent, so that they form a basis of $\mathbb{R}^2$, with other words: $\mathbb{R}^2 = \mathrm{span}(a_1, a_2) = \mathrm{Im}(A)$.

Ex 28 Linear Algebra

The Standard Inner Product

Show that the standard inner product $(\cdot,\cdot)\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, (x,y) \mapsto \sum_i^n x_i y_i$ satisfies

- (i) $\forall v, w \in \mathbb{R}^n : \ (v,w) = (w,v)$ (symmetric)
- (ii) $\forall v, w_1, w_2 \in \mathbb{R}^n : \ (v, w_1 + w_2) = (v, w_1) + (v, w_2)$
- $\forall v, w \in \mathbb{R}^n, r \in \mathbb{R} : \ (v, r \cdot w) = r \cdot (v,w)$ (linear in its second argument)
- (iii)$\forall v \in \mathbb{R}^n \backslash \{0\} : \ (v,v) > 0$ (positive definite)

Mappings $(\cdot,\cdot)\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ satisfying these three properties are called **inner products** or **symmetric bilinear forms** on $\mathbb{R}^n$.

Ex 29

Computing the Height of Objects

Let $a, b, c \in \mathbb{R}^2$ be vectors as illustrated in the figure below. We know from the lecture that the angle α between two vectors a, c is defined by

$$\cos(\alpha) = \frac{a^\top c}{\|a\|_2 \|c\|_2}.$$

Use this equality to compute the height $\|b\|_2$ of the tree in the figure below. The angle $\alpha := 36.87°$ and the distance to the tree $a := (4,0)^\top$ are given.

Solution:

Ex 30 Linear Algebra

The Four Fundamental Subspaces

1. Let $A \in \mathbb{R}^{m\times n}$. Show that
$$\forall y \in \text{im}(A), x \in \ker(A^T) \colon y^T x = 0$$
and
$$\forall y \in \text{im}(A^T), x \in \ker(A) \colon y^T x = 0.$$
2. Illuminate these findings on an example.

Remark: We say that $\ker(A^T)$ is the **orthogonal complement** of $\text{im}(A)$ and write

$$\ker(A^T)^\perp = \text{im}(A) \quad \text{or} \quad \ker(A^T) \perp \text{im}(A) \quad \text{or} \quad \ker(A^T) = \text{im}(A)^\perp.$$

Analogously $\ker(A)$ is the orthogonal complement of $\text{im}(A^T)$ and we write

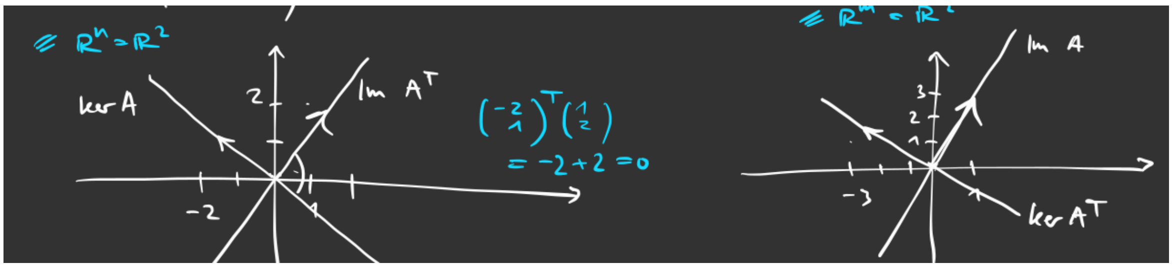
$$\ker(A)^\perp = \text{im}(A^T) \quad \text{or} \quad \ker(A) \perp \text{im}(A^T) \quad \text{or} \quad \ker(A) = \text{im}(A^T)^\perp.$$

Solution:

1. Let $y = Av \in \text{im}(A)$ and $x \in \ker(A^T)$. Then
$$y^T x = (Av)^T x = v^T A^T x = v^T (A^T x) = 0.$$
Apply this to $C = A^T$ to show the other results.
2. Let us consider
$$A = \begin{pmatrix} 1 & 2 \\ 3 & 6 \end{pmatrix}, \qquad A^\top = \begin{pmatrix} 1 & 3 \\ 2 & 6 \end{pmatrix}.$$
Then we find

$$\begin{aligned} \text{im}(A) &= \text{span} \begin{pmatrix} 1 \\ 3 \end{pmatrix} \\ \ker(A) &= \{x \in \mathbb{R}^2 : Ax = 0\} \\ &= \{x \in \mathbb{R}^2 : x_1 \begin{pmatrix} 1 \\ 3 \end{pmatrix} + x_2 \begin{pmatrix} 2 \\ 6 \end{pmatrix} = 0\} \\ &= \{x \in \mathbb{R}^2 : 1x_1 + 2x_2 = 0\} \\ &= \{x \in \mathbb{R}^2 : x_1 = -2x_2\} \\ &= \text{span} \begin{pmatrix} -2 \\ 1 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \text{im}(A^\top) &= \text{span} \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ \ker(A^\top) &= \{x \in \mathbb{R}^2 : Ax = 0\} \\ &= \{x \in \mathbb{R}^2 : 1x_1 + 3x_2 = 0\} \\ &= \{x \in \mathbb{R}^2 : x_1 = -3x_2\} \\ &= \text{span} \begin{pmatrix} -3 \\ 1 \end{pmatrix} \end{aligned}$$


$\subseteq \mathbb{R}^n = \mathbb{R}^2$
ker A
Im A^T
2
-2
1
$\begin{pmatrix} -2 \\ 1 \end{pmatrix}^T \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
$= -2 + 2 = 0$
$\subseteq \mathbb{R}^m = \mathbb{R}^2$
Im A
3
2
1
-3
ker A^T

Ex 31 Linear Algebra

Linear Dependence

1. Give an example where a nontrivial linear combination of three nonzero vectors $a_1, a_2, a_3 \in \mathbb{R}^4$ is the zero vector (nontrivial means that not all scaling coefficients are zero).
2. Write your example in the form $Ax = 0$.

Solution:

Take for example

$$A := [a_1, a_2, a_3] := \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \quad \text{and} \quad x := \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}.$$

Construction recipe: We take two random vectors x, y and form a simple linear combination $x + y$. We know these vectors are linearly dependent, so that we can take those as columns for the example matrix, i.e.,

$$x + y - (x + y) = 0 \quad \text{gives} \quad [x, y, (x + y)] \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = 0.$$

Ex 32 Linear Algebra

A matrix as a Linear Function
Let $A \in \mathbb{F}^{m\times n}$ be a matrix. Then consider the mapping $f_A \colon \mathbb{F}^n \to \mathbb{F}^m, x \mapsto Ax$.

1. Show that
$$f_A(\lambda x + y) = \lambda f_A(x) + f_A(y),$$
for all $x, y \in \mathbb{F}^n$ and $\lambda \in \mathbb{F}$.

   Hint: A vector is a matrix with just one column, so you can make use of the computation rules for matrices given in the lecture notes.

   Remark: Functions satisfying this property are called **linear**.

2. Use this fact to show the following equivalence:
$$\ker(A) := \{x \in \mathbb{F}^n \colon Ax = 0\} = \{0\} \quad \Leftrightarrow \quad f_A(x) = f_A(y) \text{ implies } x = y,$$
$$\text{(i.e., } f_A \text{ is an injective mapping).}$$

   Hint: Split up the equality $\Leftrightarrow$ into $\Rightarrow$ and $\Leftarrow$ and prove each of them separately.

Solution:

1. Let $x, y \in \mathbb{F}^n$ and $\lambda \in \mathbb{F}$. Then
$$f_A(\lambda x + y) = A(\lambda x + y) = A(\lambda x) + Ay = \lambda Ax + Ay = \lambda f_A(x) + f_A(y).$$

2. "$\Rightarrow$" Let $\ker(A) = \{0\}$
(To show: f_A is an injective mapping, i.e., $f_A(x) = f_A(y)$ implies $x = y$.)
Let $x, y \in \mathbb{F}^n$ with $f_A(x) = f_A(y)$, which implies by definition $Ax = Ay$ and thus by linearity $A(x - y) = 0$.
Thus, since $\ker(A) = \{0\}$, we conclude $x - y = 0$.

   "$\Leftarrow$" Let f_A be an injective mapping, i.e., $f_A(x) = f_A(y)$ implies $x = y$.
(To show:: $Ax = 0 \Leftrightarrow x = 0$ (here "$\Leftarrow$" is obvious).)
Let $Ax = 0$, then we find
$$f_A(0) = A0 = 0 = Ax = f_A(x).$$
Thus, since f_A is assumed to be injective, $x = 0$ (take "$y = 0$").

Ex 33 Linear Algebra

Matrix-Norm
Let $A \in \mathbb{R}^{n\times m}$. Please show that $\|A\|_1 := \max\{\|Ax\|_1 : \|x\|_1 = 1\} = \max_j \sum_{i=1}^n |a_{ij}|$.

Hint: As usual it is helpful to split up the equality sign into $\leq$ and $\geq$ and treat the parts separately.

Solution:

"$\geq$": Since $\{x : \|x\|_1 = 1\} \supset \{e_1, \dots, e_m\}$, we have

$$\max_{\|x\|_1=1} \|A\|_1 \geq \max_{x\in\{e_1,\dots,e_m\}} \|Ax\|_1.$$

$\leq$: Let $x \in \{x : \|x\|_1 = 1\},\ x \in \mathbb{R}^m$ then

$$\|\underbrace{Ax}_{\in\mathbb{R}^n}\|_1 = \sum_{i=1}^{n}\left|\sum_{j=1}^{m} a_{ij}x_j\right| \leq \sum_{j=1}^{m} \underbrace{\left[\sum_{i=1}^{n}|a_{ij}|\right]}_{\leq \max_j \sum_{i=1}^n |a_{ij}| =: m} |x_j|$$

$$\leq m \underbrace{\sum_{j=1}^{m}|x_j|}_{=\|x\|_1=1} = m$$

$$\overset{x \text{ arbitrary}}{\Rightarrow} \max_{\|x\|_1=1} \|Ax\|_1 \leq m.$$

□

Ex 34 Linear Algebra

Matrix Product as Sum of rank-1 Matrices

1. Let $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$. Show that

$$A \cdot B = \sum_{i=1}^{k} a_i b_i^\top = \sum_{i=1}^{k} \begin{pmatrix} a_{1i} \\ \vdots \\ a_{mi} \end{pmatrix} \cdot \begin{pmatrix} b_{i1} & \dots & b_{in} \end{pmatrix},$$

   where $a_i \in \mathbb{R}^{m\times 1}$ denotes the i-th *column* of A and $b_i^\top \in \mathbb{R}^{1\times n}$ denotes the i-th *row* of B.

2. Construct two examples with actual numbers.

Remark: Also see p. 10-11 in Gilbert Strang's "Linear Algebra and Learning from Data".

Solution:

Note that by definition of the matrix product we have that the entry at (μ, ν) of AB is given by

$$(AB)_{\mu\nu} = \sum_{i=1}^{k} a_{\mu i} b_{i\nu}.$$

Again, by definition of the matrix product, for the i-th column $a_i = (a_{1i}, \dots, a_{mi})^\top \in \mathbb{R}^{m\times 1}$ and i-th row $b_i^\top = (b_{i1}, \dots, b_{in}) \in \mathbb{R}^{1\times n}$, we find

$$(a_i b_i^\top)_{\mu\nu} = \sum_{j=1}^{1} (a_i)_{\mu j} (b_i^\top)_{j\nu} = (a_i)_{\mu 1} (b_i^\top)_{1\nu} = a_{\mu i} b_{i\nu}.$$

Thus

$$\left(\sum_{i=1}^{k} a_i b_i^\top\right)_{\mu\nu} = \sum_{i=1}^{k} (a_i b_i^\top)_{\mu\nu} = \sum_{i=1}^{k} a_{\mu i} b_{i\nu} = (AB)_{\mu\nu}.$$

Ex 35 Linear Algebra

Computation Rules for Matrices and Vectors

Below you find a collection of computation rules that are helpful when dealing with matrices and vectors. Feel free to prove them based on the definitions given in the lecture.

Compatibility properties of summing and scaling matrices
Let $A, B \in \mathbb{F}^{m\times n}$ and $r, s \in \mathbb{F}$. Then

$$\begin{array}{ll} i) & (r\cdot s)\cdot A = r\cdot(s\cdot A) \\ ii) & (r+s)\cdot A = r\cdot A + s\cdot A \\ & r\cdot(A+B) = r\cdot A + r\cdot B \\ iii) & 1\cdot A = A \end{array}$$

From which we can derive:

$$\begin{array}{ll} i) & 0\cdot A = 0 \\ ii) & r\cdot 0 = 0 \\ iii) & r\cdot A = 0 \quad \Rightarrow r = 0 \;\vee\; A = 0 \\ iv) & (-1)\cdot A = -A \end{array}$$

Compatibility properties of matrix sum and product
Let $A, \tilde{A} \in \mathbb{F}^{m\times n}$, $B, \tilde{B} \in \mathbb{F}^{n\times l}$, $C \in \mathbb{F}^{l\times t}$, $r \in \mathbb{F}$. Then

$$\begin{array}{rl} i) & (A\cdot B)\cdot C = A\cdot(B\cdot C) \\ ii) & (A+\tilde{A})B = AB + \tilde{A}B \\ iii) & A(B+\tilde{B}) = AB + A\tilde{B} \\ iv) & I_m A = A I_n = A \\ v) & (r\cdot A)\cdot B = r(A\cdot B) = A(r\cdot B) \\ vi) & 0A = A0 = 0 \end{array}$$

Group property of invertible matrices
For two invertible matrices $A, B \in \mathbb{F}^{n\times n}$ we find

$$\begin{array}{rl} i) & (AB)^{-1} = B^{-1}A^{-1} \\ ii) & \left(A^{-1}\right)^{-1} = A \end{array}$$

Transpose matrices
Let $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times m}$. Then

i) $(A^\top)^\top = A$,

ii) $(AB)^\top = B^\top A^\top$,

iii) $(A+B)^\top = A^\top + B^\top$,

iv) for $A \in GL(n,\mathbb{R})$ we have $(A^\top)^{-1} = (A^{-1})^\top$.

Solution:

Ex 36 Linear Algebra

Linear Modeling
Let us assume a car c_1 starts driving at time $t = 0h$ and runs with $50\frac{km}{h}$. A second car c_2 runs with $90\frac{km}{h}$ but starts 2 hours later at time $t = 2h$ at the same position as car c_1 did.

1. Describe the positions $c_1(t)$ and $c_2(t)$ of the cars as affine linear function of time, i.e., $c_i(t) = m_i t + b_i$ for some appropriate $m_i, b_i \in \mathbb{R}$, $i = 1, 2$.

2. At which time t do the cars meet?

Solution:

We know, that car 1 starts at $t = 0$ with $50\frac{\text{km}}{\text{h}}$ and car 2 starts at $t = 2$ with $90\frac{\text{km}}{\text{h}}$. Now we want to model the position of car i at time t with the function

$$c_i(t) := m_i t + b_i \quad [km].$$

1. We already know $m_1 = 50, m_2 = 90$ and

$$\begin{aligned} &1)\ 0 = c_1(0) = m_1 \cdot 0 + b_1 \quad \Rightarrow \quad b_1 = 0, \\ &2)\ 0 = c_2(2) = \underbrace{m_2}_{=90} \cdot 2 + b_2 \quad \Rightarrow \quad b_2 = -180. \end{aligned}$$

2. We want to find a $\hat{t}$, where the car meet:

$$\begin{aligned} &c_1(\hat{t}) = c_2(\hat{t}) \\ \Leftrightarrow &50 \cdot \hat{t} = 90 \cdot \hat{t} - 180 \\ \Leftrightarrow &\hat{t} = \frac{180}{40} = 4.5 \end{aligned}$$

So we conclude, that the cars will meet after 4.5 hours or 270 minutes.

Ex 37

Let $A \in \mathbb{R}^{n \times n}$ be a *negative* definite matrix, i.e., $x^\top Ax < 0$ for all $x \in \mathbb{R}^n \setminus \{0\}$. Show that the eigenvalues of A are strictly negative, i.e., $\lambda < 0$ for all $\lambda \in \sigma(A)$.

Solution:

Consider the unit vectors $e_i = (0, \dots, 1, \dots, 0)^\top \in \mathbb{R}^n$. Then

$$0 < e_i^\top A e_i = a_i i.$$

Ex 38 Linear Algebra

$(n+1)$ **vectors are always dependent**
Consider arbitrary $n+1$ vectors $v_1, \dots, v_{n+1} \in \mathbb{F}^n$. Then there are coefficients $\alpha_i \in \mathbb{F}, i = 1, \dots, n+1$, which are not all zero and solve the equation $\sum_{i=1}^{n+1} \alpha_i v_i = 0$.

Solution:

If all vectors v_i are zero, the situation is trivial. Thus, we assume that at least one of the v_i is nonzero. Use all vectors v_i as columns of the matrix $V = [v_1, \dots, v_{n+1}] \in \mathbb{R}^{n \times (n+1)}$. Then build the REF (cf. theorem **??**) of V, i.e., construct P, L, U with $L \cdot U = P \cdot V$. Then U is of the form

$$U = \begin{pmatrix} u_{1,1} & \dots & u_{1,n} & u_{1,n+1} \\ & \ddots & \vdots & \vdots \\ & & u_{n,n} & u_{n,n+1} \end{pmatrix}$$

Thus, the equation $\sum_{i=1}^{n+1} \alpha_i v_i = 0$ is equivalent to $U[\alpha_1, \dots, \alpha_{n+1}]^\top = 0$. We can choose a permutation of columns Q such that we have for $\tilde{U} := UQ$

$$\tilde{U} = \begin{pmatrix} \tilde{u}_{1,1} & \dots & \tilde{u}_{1,\ell} & \dots & \tilde{u}_{1,n} & \tilde{u}_{1,n+1} \\ & \ddots & \vdots & \vdots & & \\ & & \tilde{u}_{\ell,\ell} & \dots & \tilde{u}_{\ell,n} & \tilde{u}_{\ell,n+1} \\ 0 & \dots & 0 & \dots & 0 & 0 \\ \vdots & & \vdots & & \vdots & \vdots \\ 0 & \dots & 0 & \dots & 0 & 0 \end{pmatrix}, \quad \tilde{u}_{1,1}, \dots, \tilde{u}_{\ell,\ell} \neq 0$$

and we can choose the coefficients $\tilde{\alpha}_i$ in the following manner

$$\tilde{\alpha}_{n+1}, \dots, \tilde{\alpha}_{\ell+1} \neq 0 \text{ arbitrary}, \ \ \tilde{\alpha}_i = -\frac{1}{\tilde{u}_{ii}} \sum_{k=i+1}^{n+1} \tilde{u}_{i,k} \tilde{\alpha}_k \text{ for } i = \ell, \dots, 1 \ \text{ and then } \alpha := Q\tilde{\alpha}$$

Orthogonal Matrices

A matrix $Q \in \mathbb{R}^{n\times n}$ is called **isometric**, if

$$\|Qx\|_2 = \|x\|_2, \quad \forall x \in \mathbb{R}^n.$$

1. **Bonus** (You can get 4 bonus points for this subtask): Show the following equivalence:

$$Q \text{ is isometric} \quad \Leftrightarrow \quad Q \text{ is orthogonal.}$$

   Hint: To show "$\Rightarrow$", use the **polarization identity**

$$(x, y)_2 = \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right), \quad x, y \in \mathbb{R}^n$$

   and the observation

$$a_{ij} = (e_i, Ae_j)_2, \quad \text{for} \quad A = (a_{ij})_{ij} \in \mathbb{R}^{n\times n},$$

   where $e_j = (0, \dots, 1, \dots, 0)^T \in \mathbb{R}^n$ denote the standard basis vectors.

2. For an angle $\alpha \in [0, 2\pi]$, consider the *rotation matrix*

$$Q_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}.$$

   a) Show that Q_α is orthogonal for all α.
   Hint: Use the Pythagorian identity $\sin^2\alpha + \cos^2\alpha = 1$

   b) Draw the set $\{Q_\alpha x : \|x\|_1 = 1\}$ for $\alpha = \pi/4$.

Solution:

1. "$\Rightarrow$" Let Q be isometric, i.e., $\|Qx\|_2 = \|x\|_2$ for all $x \in \mathbb{R}^n$. (to show: $Q^\top Q = I$ or equivalently $(Q^\top Q)_{ij} = \delta_{ij}$, where δ_{ij} denotes the Kronecker delta) Following the hint we first observe that

$$(Q^\top Q)_{ij} = (e_i, Q^\top Qe_j)_2 = (Qe_i, Qe_j)_2.$$

   Second, by polarization formula and the isometry property we obtain

$$\begin{aligned}(Qx, Qy)_2 &= \frac{1}{4}\left(\|Qx+Qy\|^2 - \|Qx-Qy\|^2\right)\\ &= \frac{1}{4}\left(\|Q(x+y)\|^2 - \|Q(x-y)\|^2\right)\\ &= \frac{1}{4}\left(\|x+y\|^2 - \|x-y\|^2\right)\end{aligned}$$

   for all $x, y \in \mathbb{R}^n$. Now we insert the standard basis vectors $x = e_i, y = e_j$ and obtain

$$\begin{aligned}(Q^\top Q)_{ij} &= (Qe_i, Qe_j)_2\\ &= \frac{1}{4}\left(\|e_i+e_j\|^2 - \|e_i-e_j\|^2\right)\\ &= \delta_{ij},\end{aligned}$$

   where δ_{ij} denotes the Kronecker delta. Thus, Q is orthogonal.

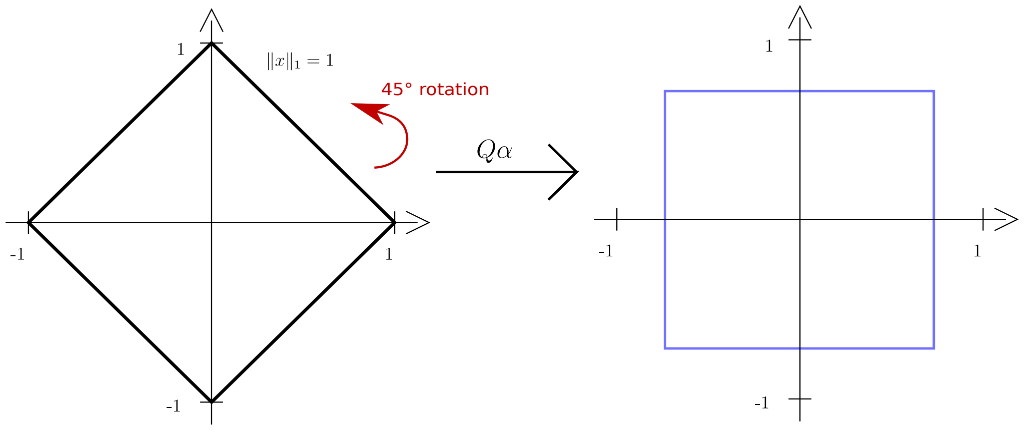
   "$\Leftarrow$" Let Q be orthogonal, i.e., $Q^\top Q = I$. Then $\|Qx\|_2 = \sqrt{x^\top Q^\top Qx} = \|x\|_2$.

2. a) By computing the matrix-matrix product we find

$$Q_\alpha^T Q_\alpha = \begin{pmatrix} \sin^2\alpha + \cos^2\alpha & -\cos\alpha\sin\alpha + \sin\alpha\cos\alpha \\ -\sin\alpha\cos\alpha + \cos\alpha\sin\alpha & \sin^2\alpha + \cos^2\alpha \end{pmatrix} = I.$$

   b) First note that

$$Q_{\pi/4} = \begin{pmatrix} \cos\pi/4 & -\sin\pi/4 \\ \sin\pi/4 & \cos\pi/4 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}.$$



Remark: We have rotated the unit $\|\cdot\|_1$-norm ball to a $\|\cdot\|_\infty$-norm ball of radius $\frac{1}{\sqrt{2}}$.

Ex 40 Linear Algebra

The p-Norm

A mapping $\|\cdot\| : \mathbb{R}^n \to [0,+\infty)$ is called **norm** on $\mathbb{R}^n$ if it satisfies the three properties

i) $\|x\| = 0 \;\Rightarrow x = 0$ *(positive definite/ point separating)*

ii) $\|r \cdot x\| = |r| \cdot \|x\|\,,\ \forall x \in \mathbb{R}^n, r \in \mathbb{R}$ *(absolutely homogeneous)*

iii) $\|x+y\| \leq \|x\| + \|y\|,\ \forall x,y \in \mathbb{R}^n$ *(subadditive/ triangle inequality)*

The most common example is given by the p-**norm** for $p \in [1,+\infty)$, defined by

$$\|x\|_p := \left(\sum_{i=1}^n |x_i|^p\right)^{\frac{1}{p}}$$

and the **supremum (or maximum) norm** defined by

$$\|x\|_\infty := \max_{i=1,\dots,n} |x_i|.$$

Remark: For a fixed $x \in \mathbb{R}^n$ one can show $\lim_{p\to\infty} \|x\|_p = \|x\|_\infty$.

Tasks:

1. Draw the sets $\{x \in \mathbb{R}^2 : \|x\|_p = 1\}$ for $p = 1, 2, \infty$.
2. Show that the Euclidean norm $\|\cdot\|_2 : \mathbb{R}^n \to [0,+\infty),\ x \mapsto \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x^\top x}$ satisfies i)-iii).
   Hint: For iii) use the Cauchy-Schwarz inequality from the lecture and show $\|x+y\|^2 \leq (\|y\|_2 + \|x\|_2)^2$ and also use $|a+b| \leq |a| + |b|$ for $a, b \in \mathbb{R}$ (i.e., the triangle inequality is true on $\mathbb{R}^1$).

Solution:

Some further remarks first:

TRIANGLE INEQUALITY AND CONVEXITY

Δ-$\neq$ $\Rightarrow$ $\|\cdot\|$ convex $\Rightarrow$ level sets (= preimage of certain value under $\|\cdot\|$) are convex

<u>Example</u>

take $e_1 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}, e_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \|e_j\|_p = 1 \quad \forall p > 0$

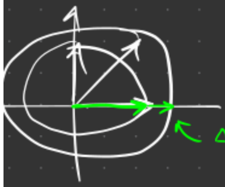
$s := (e_1 + e_2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad = (1^p)^{1/p}$

$\|s\|_1 = 1 + 1 = 2 \leq 2$ ✓

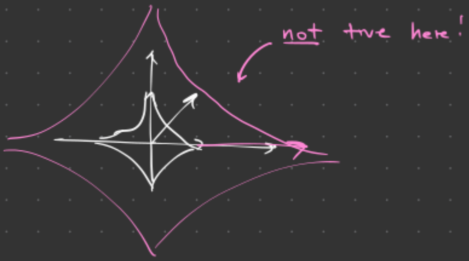
$\|s\|_\infty = 1 \leq 2$ ✓

$\|s\|_2 = (1+1)^{\frac{1}{2}} = \sqrt{2} \leq 2$ ✓

$\|s\|_{\frac{1}{2}} = (\sqrt{1} + \sqrt{1})^2 = 4 \not\leq 2$

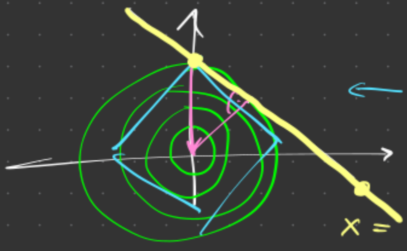


Δ-$\neq$: radii increases with at most a factor of 2



$\|\cdot\|_1$ - NORM ($p=1$) AND SPARSITY

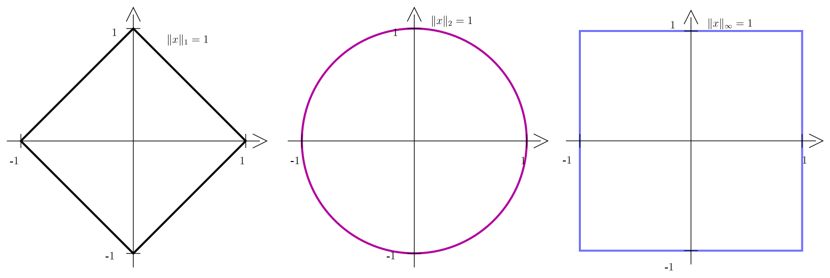
Let us consider the minimization problem: $\min_{x \in \mathbb{R}^2} \|x\|_p$ s.t. $x_2 = m x_1 + b$



$\leftarrow$ <u>Observation</u>: The diamond ($\|\cdot\|_1$ - level set) touches a line always on a "sharp" point (its corners)

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ m x_1 + b \end{pmatrix}$$

1.



2. Recall: $\|x\|_2 := \sqrt{\sum_{i=1}^n x_i^2} = \sqrt{x^T x}$

i)

$$\|x\|_2 = 0 \Rightarrow \quad \|x\|_2^2 = 0 \quad \Rightarrow \quad \sum_{i=1}^n x_i^2 = 0 \quad \Rightarrow \quad x_i^2 = 0 \;\; \forall i \quad \Rightarrow \quad x_i = 0 \;\; \forall i$$

$$\Rightarrow \quad x = 0$$

ii) Let $\lambda \in \mathbb{R},\ x \in \mathbb{R}^n$, then

$$\|\lambda x\|_2 = \sqrt{\sum_{i=1}^{n} \underbrace{(\lambda x_i)^2}_{\lambda^2 x_i^2}} = \sqrt{\lambda^2 \sum_{i=1}^{n} x_i^2} = \sqrt{\lambda^2}\sqrt{\sum_{i=1}^{n} x_i^2} = |\lambda| \cdot \|x\|_2.$$

iii) Let $x, y \in \mathbb{R}^n$ (to show: $\|x+y\| \leq \|x\| + \|y\|$, or equivalently $\|x+y\|^2 \leq (\|x\| + \|y\|)^2$ (note that $x \mapsto x^2$ is monotonically increasing for $x \geq 0$)). We find

$$\begin{aligned}\|x+y\|^2 = |\underbrace{\|x+y\|^2}_{\geq 0}| &= \left|(x+y)^T(x+y)\right| = \left|(x+y)^T x + (x+y)^T y\right| = \left|x^T x + y^T x + x^T y + y^T y\right| \\ &= \left|x^T x + 2x^T y + y^T y\right| = \left|\|x\|_2^2 + 2x^T y + \|y\|_2^2\right| \\ &\leq \|x\|_2^2 + 2 \underbrace{|x^T y|}_{\leq \|x\|\|y\|} + \|y\|_2^2 \\ &\leq \left(\|x\|_2 + \|y\|_2\right)^2 .\end{aligned}$$

Ex 41 SciPy Stack, Python

NumPy and Matplotlib II

In this exercise you need to make use of the Python packages numpy and `matplotlib`.

1. Construct an identity matrix `I` of dimension 100x100 as `numpy.ndarray`.
2. Construct a banded matrix `A` of the form

$$A = \frac{100^2}{4\pi^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & -2 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 \end{bmatrix}$$

   of dimension 100x100 as `numpy.ndarray`.
3. Plot both matrices with the function `imshow()` of the package `matplotlib.pyplot`.
4. Construct a vector z with the `linspace()` function of the numpy package. It should contain a grid of 102 values between 0 and 2π.
5. Use Python's *slicing* capabilities to copy all except from the first and the last value of z into another vector x.
6. Calculate $y = \sin(x)$ and the matrix vector product $d = Ay$ using `numpy.sin()` and `numpy.dot()`.
7. Plot y and d into the same plot using `plot()` from the `matplotlib.pyplot` package.

Hint: You might need to use `matplotlib.pyplot.show()` in order to guarantee that the notebook shows some output.

Solution:

```
import numpy as np

dim = 100

## Identity Matrix
# Create matrix with numpy.eye()
I = np.eye(dim)
print(I)
help(np.eye)

## Construct matrix A
```

```python
### via for loop
A = np.zeros((dim,dim), dtype = float)
# for loop for running through rows and columns
for i in range(dim):
    for j in range(dim):
        # diagonal entries
        if i==j:
            A[i,j] = -2
        # first lower and upper off diagonals
        if np.abs(i-j) == 1:
            A[i,j] = 1

# The factor s
# The number pi is also provided by numpy
s = 100**2/(4*(np.pi**2))
# scale A
A = s*A
print("A=", A)

### Construct A via np.eye
# numpy.eye() can also create diagonal entries
A = -2*I + np.eye(dim, k=-1) + np.eye(dim, k=1)

# scale A
A = s*A
print("I=", I)
print("\n")
print("A=", A)

## Plot using `imshow`
# Matplotlib provides plotting functionalities
import matplotlib.pyplot as plt
plt.imshow(I)
plt.show()
plt.imshow(A)
plt.show()

## Use `linspace`
# Numpy linspace allows to easily create a regular 1D-grid.
z = np.linspace(0, 2*np.pi, 102)
plt.plot(z,np.zeros(102), 'rx')

## Slicing
# Python allows to easily take slices from arrays.
# The syntax is vector[start:stop:step]
# -1 can be used to get the index of the last entry
x = z[1:-1]

## Apply
y = np.sin(x)

# The @ operator can be used for matrix multiplication of numpy (!) arrays.
d = A@y

## Plot
# We can plot two plots into the same plot
plt.plot(x, y, label = "sin")
plt.plot(x, d, label = "- sin")

# This line is necessary for the legend. It does not show up otherwise.
plt.legend()

# This line plots all figures which were created above.
plt.show()

### Remark
#$\sin''(x) = -\sin(x)$
```

```
#The matrix $A$ is a "discretization" of the second derivative
```

Ex 42 Linear Algebra, Python

The Matrix-Vector Product
Implement a function that takes as input a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^n$ and then returns the matrix-vector product Ax.

1. Implement the following four ways:
   a) **Dense:** Input expected as `numpy.ndarray`:
   Assume that the matrix and the vector are delivered to your function as `numpy.ndarray`.
      i. Implement the matrix-vector product "by hand" using for loops, i.e., *without* using numpy functions/methods.
      ii. Implement the matrix-vector product using `A.dot(x)`, `A@x`, `numpy.matmul(A,x)` or `numpy.dot(A,x)`.

   b) **Sparse:** Matrix expected in CSR format:
   Assume that the matrix is delivered to your function as `scipy.sparse.csr_matrix` object. The vector x can either be expected as `numpy.ndarray` or simply as a Python `list`.
      i. Access the three CSR lists via `A.data`, `A.indptr`, `A.indices` and implement the matrix-vector product "by hand" using for loops.
      ii. Implement the matrix-vector product using `A.dot(x)` or `A@x` .

2. **Test** your four different routines from above on the following matrix $A \in \mathbb{R}^{n\times n}$ with constant diagonals given by

$$A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

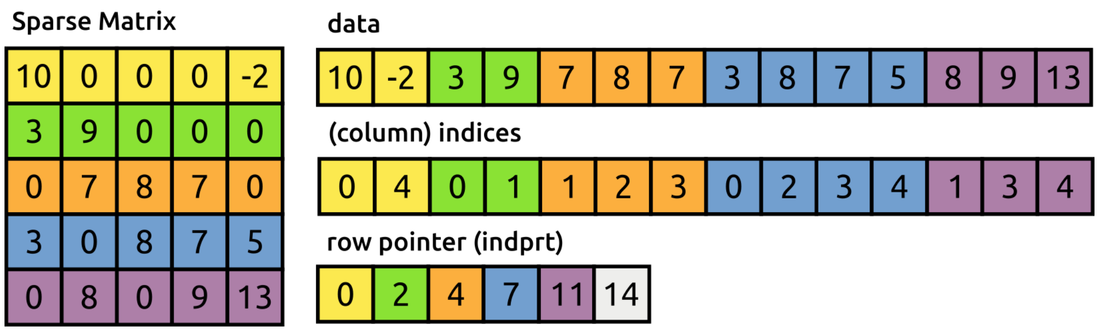
   and the input vector

$$x = (1, \cdots, 1)^\top \in \mathbb{R}^n \quad \text{(you can use: } \texttt{x = numpy.ones(n)}\text{).}$$

   a) Determine how $b := A \cdot x \in \mathbb{R}^n$ looks like in this example in order to facilitate a test.
   b) Test whether your four routines compute the matrix–vector product correctly by checking $A \cdot x = b$.
   c) Use different values for the dimension n (especially large $n \geq 10^5$ – note that you may exceed your hardware capacities for the dense computations).

   Remark: The matrix has "2" on the main diagonal and "-1" on the first off-diagonals.

3. For all cases:
   a) **Memory:** What is the number of Gbytes needed to store an $m \times n$ array of `floats`? Print the number of Gbytes which are needed to store the matrix in all cases.
   Hint: A number implemented as `float` in Python implements double precision and therefore needs 64 Bits of storage. For a `numpy.ndarray` you can type `A.nbytes` and for the `scipy.sparse.csr_matrix` you can type `A.data.nbytes + A.indptr.nbytes + A.indices.nbytes`.
   b) **Computation times:** Measure the time which is needed in each case to compute the matrix-vector product for a random input vector `x = numpy.random.rand(n)`.
   Hint: In the IPython shell you can simply use the *magic function* `%timeit` to measure the time for a certain operation. For example, you can type `%timeit pythonfunction(x)`. Alternatively you can use the package `timeit`.



Sparse Matrix

10	0	0	0	-2
3	9	0	0	0
0	7	8	7	0
3	0	8	7	5
0	8	0	9	13

data

10	-2	3	9	7	8	7	3	8	7	5	8	9	13

(column) indices

0	4	0	1	1	2	3	0	2	3	4	1	3	4

row pointer (indprt)

0	2	4	7	11	14

Example of a Matrix in CSR Format

Solution:

```python
#!/usr/bin/python
import numpy as np
import scipy.sparse as scs
import timeit


def matvec_dense(A, x, byhand=0):
    """
    computes the matrix vector product based on numpy.ndarray

    Parameters
    ----------
    A : (m,n) numpy.ndarray
       matrix
    x : (n, ) numpy.ndarray
       vector
    byhand : int
       switch between for loop and @ operator

    Returns
    -------
        A*x: matrix-vector product
    """
    if byhand:
        # read the dimensions of the input objects
        m, n = np.shape(A)
        nx = len(x)

        # raise an error if the dimensions do not match
        if n != nx:
            raise Exception('dimension of A and x must match. The dimension \
                            for A and x were: {}'.format(str(np.shape(A))
                                                         + " " + str(len(x))))

        # if dimensions match, start computing the matrix-vector product:
        else:
            # initialize the output vector
            b = np.zeros(m)
            # a loop over row indices to compute each entry of b
            for i in range(m):
                # a loop over column indices to compute the inner product
                for j in range(n):
                    b[i] += A[i, j] * x[j]
    else:
        b = A.dot(x)  # np.dot(A,x), A@x
    return b


# we could implement our own csr-class in python:
# class csr_matrix:
#    def __init__(self, data, indices, indptr):
#        self.data = data
#        self.indices = indices
#        self.indptr = indptr

def matvec_sparse(A, x, byhand=0):
    """computes the matrix vector product based on csr matrix

    Parameters
    ----------
    A: (m,n) matrix  stored in CSR, i.e., in terms of three lists; here:
       class with attributes data, indices, indptr
    x: (n, ) numpy.ndarray or list of length n (= number of cols) numbers
       vector
    byhand : int
```

```
        switch between for loop and @ operator

    Returns
    -------
        A*x: matrix-vector product
    """
    if byhand:
        # dimension check?
        # can we get the column dimension from sparse csr class? > depends
        b = [0] * (len(A.indptr) - 1)
        for i, pair in enumerate(zip(A.indptr[0:-1], A.indptr[1:])):
            for a_ij, j in zip(A.data[pair[0]:pair[1]],
                               A.indices[pair[0]:pair[1]]):
                b[i] += a_ij * x[j]
    else:
        # make sure A and x have the correct format for the dot method
        A = scs.csr_matrix(A)
        x = np.array(x)
        # compute matrix-vector product
        b = A.dot(x)
    return np.array(b)


print("\nIn order to get the docstring of our function we can type  \
      \n\n     help(functionName)\n\nFor example: ")
print(help(matvec_dense))

if __name__ == "__main__":
    # Note: the following part is only executed if the current script is
    #       run directly, but not if it is imported into another script
    # ---------------------------------------------------------------#
    #    EXPERIMENT
    # ---------------------------------------------------------------#
    # the experiment
    n = int(1e1)  # matrix column dimension
    m = n  # matrix row dimension
    runs = 50  # how many runs for time measurement
    x = np.random.rand(n)  # random vector x

    # test arrays for which we know the result
    xtest = np.ones(n)  # test input x
    btest = np.zeros(m)  # known test output b
    btest[[0, -1]] = 1

    # just some strings for printing commands
    expstr = ["Time dot:     ", "Time hand:    "]
    teststr = ["Test dot:     ", "Test by hand: "]

    # ---------------------------------------------------------------#
    #    NUMPY DENSE
    # ---------------------------------------------------------------#
    print("\n---- Numpy Dense ----")
    A = 2 * np.eye(n) - np.eye(n, k=1) - np.eye(n, k=-1)
    print("Memory:", np.round(A.nbytes * 10 ** -9, decimals=4), "Gbytes\n")
    for byhand in [0, 1]:
        print(teststr[byhand], np.allclose(btest,
                                           matvec_dense(A, xtest, byhand=byhand)))
        def dense():
            return matvec_dense(A, x, byhand=byhand)


        print(expstr[byhand], timeit.timeit("dense()",
                                            setup="from __main__ import dense", number=runs),
    "\n")

    # ---------------------------------------------------------------#
    #   SCIPY SPARSE
    # ---------------------------------------------------------------#
    print("\n---- Scipy Sparse ----")
    A = 2 * scs.eye(n) - scs.eye(n, k=1) - scs.eye(n, k=-1)
    print("Memory:", np.round((A.data.nbytes + A.indptr.nbytes +
                               A.indices.nbytes) * 10 ** -9, decimals=4),
          "Gbytes\n")
    for byhand in [0, 1]:
        print(teststr[byhand],
              np.allclose(btest, matvec_sparse(A, xtest, byhand=byhand)))


        def sparse():
            return matvec_sparse(A, x, byhand=byhand)


        print(expstr[byhand], timeit.timeit("sparse()",
                                            setup="from __main__ import sparse", number=runs),
     "\n")
```

Ex 43 Linear Algebra

Property of a skew-symmetric Matrix
A matrix $C \in \mathbb{R}^{n\times n}$ is called *skew-symmetric* if

$$C^\top = -C.$$

Please show that for a real-valued skew-symmetric matrix $C \in \mathbb{R}^{n\times n}$ it holds

$$x^\top Cx = 0$$

for all $x \in \mathbb{R}^n$ (i.e., a vector x is always mapped to a perpendicular vector Cx).

Solution:

Let $C \in \mathbb{R}^{n\times n}$, be a skew-symmetric matrix, i.e. $C^T = -C$.
Now show, that $x^T Cx = 0$ holds.

Proof: $x \in \mathbb{R}^n$,

$$x^T Cx = (x^T Cx)^T = x^T C^T x = -x^T Cx$$
$$\Rightarrow \quad x^T Cx = 0 \quad (r \in \mathbb{R}, r = -r \Rightarrow r = 0)$$

Ex 44 Linear Algebra

Rank/Image and Nullity/Kernel
Consider the matrix

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

the column vector $\mathbf{1} := \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ (i.e., a (3×1) matrix) and the row vector $\tilde{\mathbf{1}} := (1\ 1\ 1)$ (i.e., a (1×3) matrix).

1. Show that $A = \mathbf{1}\tilde{\mathbf{1}}$.
2. Find two distinct nonzero vectors x and y, so that $Ax = 0$ and $Ay = 0$.
3. How does the image Im(A) look like? First draw a picture. Then find a basis of Im(A) to determine the rank of the matrix.
4. How does the kernel ker(A) look like? First draw a picture. Then find a basis of ker(A) to determine the nullity of the matrix.
   Remark: You have to prove that your vectors are a basis.

Solution:

1. By applying the matrix-matrix product definition we multiply the matrix $\mathbf{1}$ with each column in $\tilde{\mathbf{1}}$ (here, a column is just the number 1). We obtain
$$\mathbf{1}\tilde{\mathbf{1}} = \begin{pmatrix}1\\1\\1\end{pmatrix} \cdot (1\ 1\ 1) = \left(1 \cdot \begin{pmatrix}1\\1\\1\end{pmatrix} 1 \cdot \begin{pmatrix}1\\1\\1\end{pmatrix} 1 \cdot \begin{pmatrix}1\\1\\1\end{pmatrix}\right) = A.$$

2. Since $a_1 = a_2 = a_3 = \mathbf{1}$, we have
$$0 = Ax = a_1x_1 + a_2x_2 + a_3x_3 = a_1(x_1 + x_2 + x_3) \Leftrightarrow x_1 + x_2 + x_3 = 0.$$
Choose, e.g., $x = \begin{pmatrix}-1\\1\\0\end{pmatrix}$ and $y = \begin{pmatrix}-1\\0\\1\end{pmatrix}$.

3. By definition of the image we have
$$\begin{aligned}\text{Im}(A) &= \text{span}(a_1, a_2, a_3)\\ &= \{\lambda_1\mathbf{1} + \lambda_2\mathbf{1} + \lambda_3\mathbf{1} \colon \lambda_i \in \mathbb{R}\}\\ &= \{\lambda\mathbf{1} \colon \lambda \in \mathbb{R}\}\\ &= \text{span}(\mathbf{1}).\end{aligned}$$
Since $\mathbf{1} \neq 0$, we have that $\{\mathbf{1}\}$ is a basis of length 1 for $\text{Im}(A)$. In particular we find
$$\text{rank}(A) := \dim \text{Im}(A) = 1.$$
(Note that two equal vectors $x = y$ are linearly dependent and that a single nonzero vector $x \neq 0$ is linearly independent.)

4. From 2. we already know
$$\begin{aligned}\ker(A) := \{x \in \mathbb{R}^3 \colon Ax = 0\} &= \{x \in \mathbb{R}^3 \colon x_1 + x_2 + x_3 = 0\}\\ &= \{x \in \mathbb{R}^3 \colon x_1 = -(x_2 + x_3)\}\\ &= \left\{\begin{pmatrix}-x_2 - x_3\\x_2\\x_3\end{pmatrix} : x_2, x_3 \in \mathbb{R}\right\}\\ &= \left\{x_2\begin{pmatrix}-1\\1\\0\end{pmatrix} + x_3\begin{pmatrix}-1\\0\\1\end{pmatrix} : x_2, x_3 \in \mathbb{R}\right\}\\ &= \text{span}\left\{\begin{pmatrix}-1\\1\\0\end{pmatrix}, \begin{pmatrix}-1\\0\\1\end{pmatrix}\right\}.\end{aligned}$$
Since $b_1 := \begin{pmatrix}-1\\1\\0\end{pmatrix}$ and $b_2 := \begin{pmatrix}-1\\0\\1\end{pmatrix}$ are linearly independent (in fact, one can show $[b_1, b_2]x = 0$ implies $x = 0$), they form a basis of $\ker(A)$ and thus we have $\dim(\ker(A)) = 2$.

Ex 45 Linear Algebra

The Inner Product and SPD Matrices

- A mapping $(\cdot,\cdot)\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F}$ is called **inner product** on $\mathbb{F}^n$ if it satisfies:
  - i) Hermitian: $\forall x, y \in \mathbb{F}^n:\ (x, y) = \overline{(y, x)}$,
  - ii) Linear in its second argument: $\forall x, y_1, y_2 \in \mathbb{F}^n, \lambda \in \mathbb{F}:\ (x, y_1 + \lambda y_2) = (x, y_1) + \lambda(x, y_2)$,
  - iii) Positive definite: $\forall x \in \mathbb{F}^n \backslash \{0\}:\ (x, x) > 0$.
- A hermitian matrix $A \in \mathbb{F}^{n\times n}$ ($A^H = A$) is called **positive definite** (p.d.), if
$$x^\top Ax > 0 \quad \forall x \in \mathbb{F}^n \setminus \{0\}.$$

- We find the following relation:

$$(\cdot,\cdot)\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F} \text{ is inner product } \iff \exists A \in \mathbb{F}^{n\times n} \text{ hermitian and p.d.: } (x,y) = x^T A y.$$

In the lecture so far we considered the *standard inner product*

$$(\cdot,\cdot)_2\colon \mathbb{F}^n \times \mathbb{F}^n \to \mathbb{F},\ (x,y) \mapsto x^H I_n y = \sum_{i=1}^{n} \overline{x_i} y_i$$

which corresponds to the hermitian and positive definite matrix I_n (identity matrix).

Tasks:

1. Consider $\mathbb{F} = \mathbb{R}$. Find an example $A \in \mathbb{R}^{2\times 2}$ which is symmetric and positive definite.
2. Draw the set $\{Ax\colon \|x\|_2 = 1\}$.

Solution:

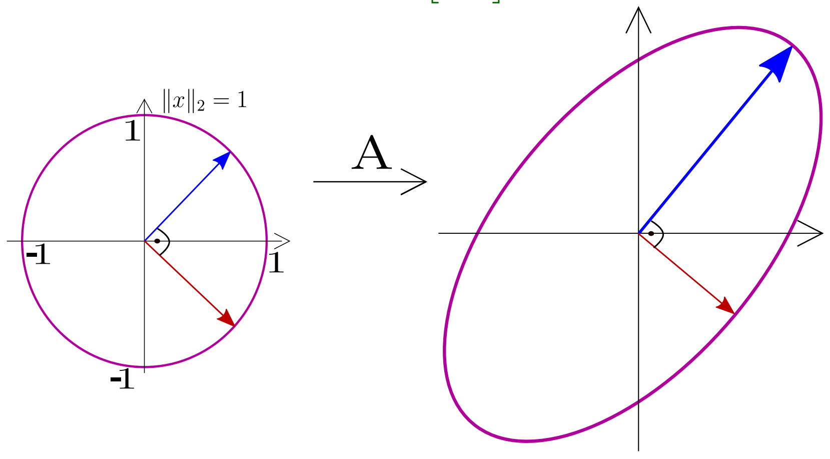
1. Let us consider $A := \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ with $a_{12} = a_{21}$ and first analye the property for being A being positive definite. We see

$$x^T A x = a_{11}x_1^2 + a_{12}x_1x_2 + a_{12}x_2x_1 + a_{22}x_2^2 = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 \overset{!}{>} 0 \quad \forall x \neq 0.$$

<u>Example:</u>

$$\begin{aligned} A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},\ x^T A x &= \overbrace{2x_1^2}^{x_1^2+x_1^2} + 2x_1x_2 + 2x_2^2 \\ &= x_1^2 + x_2^2 + \underbrace{(x_1+x_2)^2}_{\geq 0} \\ &\geq x_1^2 + x_2^2 \\ &> 0 \qquad \forall \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{aligned}$$

2. Consider $\{Ax : \|x\|_2 = 1\}$ with $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. Also recall from the lecture that $\left(3, \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right), \left(1, \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right)$ are eigenpairs.



SPD Matrices

Definition. Let $A \in \mathbb{R}^{n\times n}$ be a matrix.

- A is called **symmetric**, if $A^\top = A$.
- A is called **positive definite (semi-definite)**, if $x^\top A x > 0$ ($x^\top A x \geq 0$) for all $x \neq 0$.

Remark: If A is symmetric and positive definite (SPD), then $(x, y) := x^T Ay$ defines a general so-called *inner product.* In the lecture we mainly consider the *standard inner product*

$$(\cdot,\cdot)_2 \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R},\ (x,y) \mapsto x^T I_n y = \sum_{i=1}^{n} x_i y_i$$

which corresponds to the symmetric and positive definite matrix I_n (identity matrix).

Tasks:

1. Find an example $A \in \mathbb{R}^{2\times 2}$ which is symmetric and positive definite.
2. Draw the set $\{Ax\colon \|x\|_2 = 1\}$.

Solution:

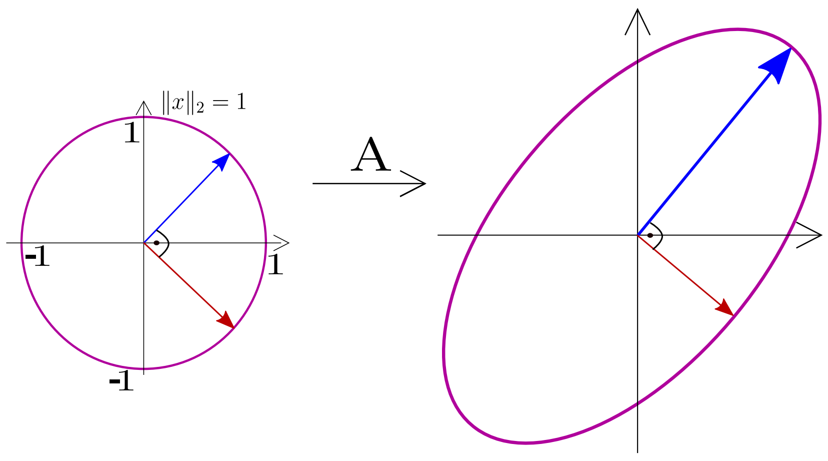
1. Let us consider $A := \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ with $a_{12} = a_{21}$ and first analye the property for being A being positive definite. We see

$$x^T Ax = a_{11}x_1^2 + a_{12}x_1x_2 + a_{12}x_2x_1 + a_{22}x_2^2 = a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2 \overset{!}{>} 0 \quad \forall x = (x_1, x_2)^\top \neq (0, 0)^\top.$$

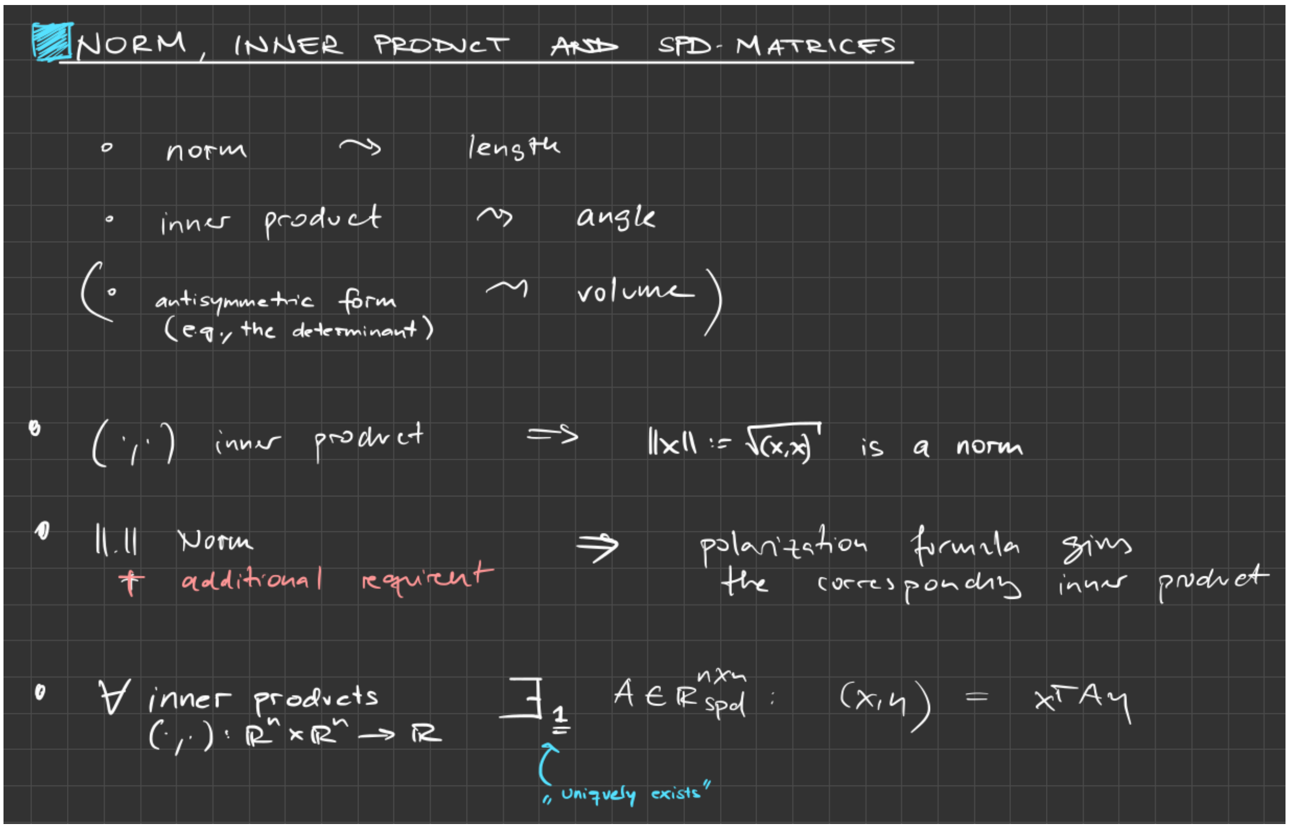
Example:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},\ x^T Ax = \overbrace{2x_1^2}^{x_1^2+x_1^2} + 2x_1x_2 + 2x_2^2$$
$$= x_1^2 + x_2^2 + \underbrace{(x_1 + x_2)^2}_{\geq 0}$$
$$\geq x_1^2 + x_2^2$$
$$> 0 \qquad \forall \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

2. Consider $\{Ax : \|x\|_2 = 1\}$ with $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. Also recall from the lecture that $\left(3, \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right), \left(1, \begin{bmatrix} 1 \\ -1 \end{bmatrix}\right)$ are eigenpairs (see later).



Further general note:



NORM, INNER PRODUCT AND SPD-MATRICES

- norm $\rightsquigarrow$ length
- inner product $\rightsquigarrow$ angle

(- antisymmetric form (e.g., the determinant) $\rightsquigarrow$ volume)

- $(\cdot,\cdot)$ inner product $\Rightarrow$ $\|x\| := \sqrt{(x,x)}$ is a norm
- $\|.\|$ Norm + additional requirent $\Rightarrow$ polarization formula gives the corresponding inner product
- $\forall$ inner products $(\cdot,\cdot): \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ $\exists_1$ $A \in \mathbb{R}^{n\times n}_{spd}$: $(x,y) = x^\top A y$

„ Uniquely exists"

Ex 47 Linear Algebra

Thin and Fat Full Rank Matrices

Answer the following questions without using the SVD. Instead, exploit the orthogonality relation between the four fundamental subspaces and the dimension formula.

1. What is the orthogonal complement of $\{0\}$ and $\mathbb{R}^n$ in $\mathbb{R}^n$, respectively?
2. Give an example for a matrix $C \in \mathbb{R}^{m\times n}$ with nullity$(C) = 0$ (injective).
   a) What do we know about the order relation between m and n? (which one is greater or equal than the other?)
   b) What do we know about the columns of C?
   c) Let $b \in$ im(C). Can we find two distinct $x_1 \neq x_2 \in \mathbb{R}^n$ such that $Cx_1 = b = Cx_2$? Explain your answer.
   d) What do we know about the matrix $C^\top C \in \mathbb{R}^{n\times n}$?
3. Give an example for a matrix $A \in \mathbb{R}^{m\times n}$ with $n > m$ and rank$(A) < m$.
4. Give an example for a matrix $R \in \mathbb{R}^{m\times n}$ with rank$(R) = m$ (surjective).
   a) What do we know about the order relation between m and n?
   b) What do we know about the rows of R?
   c) What do we know about the matrix $RR^\top \in \mathbb{R}^{m\times m}$?
   d) Let $b \in \mathbb{R}^m$. Can we find an $x \in \mathbb{R}^n$ such that $Rx = b$? Explain your answer and give an example for your matrix.
5. Give an an example for a matrix $A \in \mathbb{R}^{n\times n}$ with rank$(A) = n$ (invertible).
   a) What do we know about the dimensions of $\ker(A)$, $\ker(A^\top)$ and im$(A^\top)$?
   b) What do we know about the columns and rows of A?
   c) Let $b \in \mathbb{R}^n$. Can we find an *unique* $x \in \mathbb{R}^n$ such that $Ax = b$? Explain your answer and give an example for your matrix.
6. **Bonus*:** Give an an example for a matrix $A \in \mathbb{R}^{m\times n}$ with rank$(A) =$ rank$(A^\top)$ and nullity$(A) \neq$ nullity$(A^\top)$.

Solution:

Let us recall the dimension formula here

$$n = \text{rank}(A) + \text{nullity}(A),$$
$$m = \text{rank}(A^\top) + \text{nullity}(A^\top).$$

1. We have

$$\{0\}^\perp = \{x \in \mathbb{R}^n : x^\top v = 0 \ \ \forall v \in \{0\}\} = \{x \in \mathbb{R}^n : x^\top 0 = 0\} = \mathbb{R}^n$$

and

$$(\mathbb{R}^n)^\perp = \{x \in \mathbb{R}^n : x^\top y = 0 \ \ \forall y \in \mathbb{R}^n\} = \{x \in \mathbb{R}^n : \ker(x^\top) = \{0\}\} = \{0\}.$$

[Not part of the exercise:]
Together with $(U^\perp)^\perp = U$ this gives in general

$$\begin{aligned} \mathbb{R}^m = \text{im}(A) = \ker(A^\top)^\perp &\iff \ker(A^\top) = \{0\}, \quad \text{or} \\ \text{rank}(A) = m &\iff \text{nullity}(A^\top) = 0, \quad \text{or} \\ A \ \text{ surjective} &\iff A^\top \ \text{ injective.} \end{aligned}$$

Analogously for the transpose

$$\begin{aligned} \mathbb{R}^n = \text{im}(A^\top) = \ker(A)^\perp &\iff \ker(A) = \{0\}, \quad \text{or} \\ \text{rank}(A^\top) = n &\iff \text{nullity}(A) = 0, \quad \text{or} \\ A^\top \ \text{ surjective} &\iff A \ \text{ injective.} \end{aligned}$$

2. Example:

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

a) By dimension formula we know

$$m \geq \dim(\text{im}(C)) = \text{rank}(C) = n - \text{nullity}(C) = n.$$

Only square or thin matrices can be injective!

b) Since $\text{nullity}(C) = 0$ we know $\ker(C) = \{0\}$ and thus the n columns of C are independent (in fact, only the zero combination gives the zero vector).

c) No (because f_C injective). Recall the proof: Assume yes, then $Cx_1 = b = Cx_2 \iff C(x_1 - x_2) = 0$, where $x_1 - x_2 \neq 0$ due to $x_1 \neq x_2$. This contradicts the fact that $\ker(C) = \{0\}$.

Independent columns assure that a solution to $Cx = b$ is unique (if it exists).

d) Since $\ker(C^\top C) = \ker(C) = \{0\}$, we have that the $(n \times n)$-matrix $C^\top C$ is invertible.

3. Example: Take for two nonzero vectors $u \in \mathbb{R}^m, v \in \mathbb{R}^n$ with $n > m > 1$ the outer product

$$A := uv^\top,$$

so that each column is a scaling of u and thus $\text{rank}(A) = 1 < m$.

Simply many columns are not enough for surjectivity!
We need independence to get a "larger" subspace.

4. Example: $R := C^\top$.

a) From above recall: $\text{rank}(R) = m \iff \text{nullity}(R^\top) = 0$. Now by dimension formula we get

$$n \geq \dim(\text{im}(R^\top)) = \text{rank}(R^\top) = m - \text{nullity}(R^\top) = m.$$

Only square or fat matrices can be surjective!

b) Again, by $\text{nullity}(R^\top) = 0$, they are independent.

c) Since $\ker(RR^\top) = \ker(R^\top) = \{0\}$, it's invertible.

d) Yes, since $\text{im}(R) = \mathbb{R}^m$, any $b \in \mathbb{R}^m$ is of the form $Rx = b$ for some $x \in \mathbb{R}^n$.

Independent rows assure that a solution to $Rx = b$ exists.

5. a) By dimension formula

$$\text{nullity}(A) = n - \text{rank}(A) = 0,$$

Then using from above $\text{rank}(A^\top) = n \iff \text{nullity}(A) = 0$, we find $\text{rank}(A^\top) = n$, and therefore also

$$\text{nullity}(A^\top) = n - \text{rank}(A^\top) = 0.$$

b) Since $\ker(A) = \{0\} = \ker(A^\top)$ the n rows and the n columns are independent.

c) Yes, because from above we know: Independent rows give existence and independent columns uniqueness.

6. Take $A = C$ with C from above. Then

$$\text{rank}(A) = 2 = \text{rank}(A^\top).$$

However

$$\text{nullity}(C) = 0 \neq 1 = \text{nullity}(C^\top).$$

We will learn below:
Always $\text{rank}(A) = \text{rank}(A^\top)$,
but a similar result is not true in general for $\text{nullity}(A)$.

Ex 48 Linear Algebra

Computation Rules for Matrices II

Let $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times m}$. Then

i) $(A^\top)^\top = A$,

ii) $(AB)^\top = B^\top A^\top$,

iii) $(A + B)^\top = A^\top + B^\top$,

iv) for $A \in GL(n, \mathbb{R})$ we have $(A^\top)^{-1} = (A^{-1})^\top$.

Solution:

$$C := A \cdot B, C := [c_{ij}]_{ij}, c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}, A^T = [a'_{ij}]_{ij}, B^T = [b'_{ij}]_{ij},$$

$$[C]_{ij} = C_{ij} = \sum_{k=1}^{n} a_{jk}b_{ki} = \sum_{k=1}^{n} a'_{kj}b'_{ik} = \sum_{k=1}^{n} b'_{ik}a'_{kj} = [B^T A^T]_{ji} \Rightarrow C^T = B^T A^T$$

$$A \in GL(n, \mathbb{F}), A^{-1}A = I_n \quad \Rightarrow \quad \underbrace{(A^{-1}A)^T}_{A^T(A^{-1})^T} = I_n^T = I_n$$

$$\Rightarrow (A^{-1})^T = (A^T)^{-1}, \text{because Inverse is unique}$$

Ex 49 Linear Algebra

Height of a tree

Let $a, b, c \in \mathbb{R}^2$ be vectors as illustrated in the figure below. We know from the lecture that the angle α between two vectors a, c is defined by

$$\cos(\alpha) = \frac{a^\top c}{\|a\|_2\|c\|_2}.$$

Use this equality to compute the height $\|b\|_2$ of the tree in the figure below. The angle $\alpha := 36.87°$ and the distance to the tree $a := (4, 0)^\top$ are given.

Solution:

$$\begin{aligned}
&a = (4,0), b = (0,h), c = (4,h)\\
&\cos(\alpha) = \frac{a^\top c}{\|a\|\|c\|}\\
&\Rightarrow \|c\| = \frac{4}{\cos(\alpha)}\\
&\Rightarrow \|c\| = 5\\
&\Rightarrow 25 = \|c\|^2 = \|a\|^2 + \|b\|^2 = 16 + h^2\\
&\Rightarrow h = 3
\end{aligned}$$

3 Solving Linear Systems

Ex 50

Curve Fitting
Assume you were given the following data:

z	-1	0	1
y	-1	3	1

1. **Polynomial Interpolation:**
   Find coefficients $c_i \in \mathbb{R}$ for the following parabola (polynomial of degree 2)
   $$f(z) = c_0 + c_1 z + c_2 z^2,$$
   such that $f(z_i) = y_i$ for all three measurements $i = 1, 2, 3$ from the table above. Therefore, set up and solve a linear system of the form $Ax = b$, where $x = (c_0, c_1, c_2)$. Draw the measurements and the parabola into one plot.

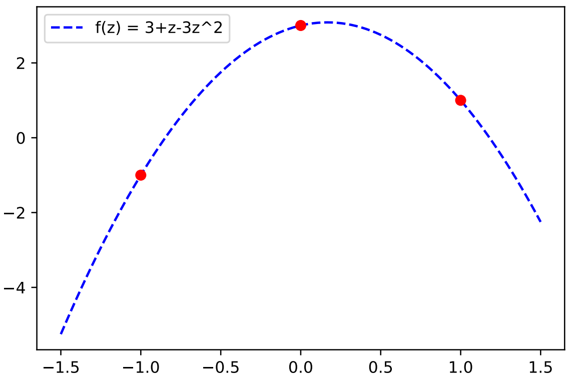
2. **Univariate Linear Regression:**
   Now assume a linear model of the form
   $$f(z) = c_0 + c_1 z.$$
   Consider $x = (c_0, c_1)$. Determine a matrix A and a vector b, such that
   $$\sum_{i=1}^{3} (f(z_i) - y_i)^2 = \|Ax - b\|_2^2.$$
   Solve the least squares problem
   $$\min_{x \in \mathbb{R}^2} \|Ax - b\|_2^2,$$
   to determine the coefficients $x = (c_0, c_1)$ by solving the *normal equation* (explanation follows later in the lecture)
   $$A^T A x = A^T b.$$
   Draw the measurements and the straight line into one plot.

Solution:

1. INTERPOLATION:
   - Aim: We want to find c_0, c_1, c_2, such that
     $f(z_1) = c_0 \cdot 1 + c_1 \cdot z_1 + c_2 \cdot z_1^2 = y_1,$
     $f(z_1) = c_0 \cdot 1 + c_1 \cdot z_2 + c_2 \cdot z_2^2 = y_2,$
     $f(z_1) = c_0 \cdot 1 + c_1 \cdot z_3 + c_2 \cdot z_3^2 = y_3.$
   - With $A = (a_{ij})_{ij} := (z_i^{j-1})_{ij},\ x := (c_0, c_1, c_2)^T,\ b := (y_1, y_2, y_3)^T$ this is equivalent to:
     $$Ax = b.$$
   - Inserting the data points (z_i, y_i) gives:
     $$A = \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad b = \begin{pmatrix} -1 \\ 3 \\ 1 \end{pmatrix}.$$

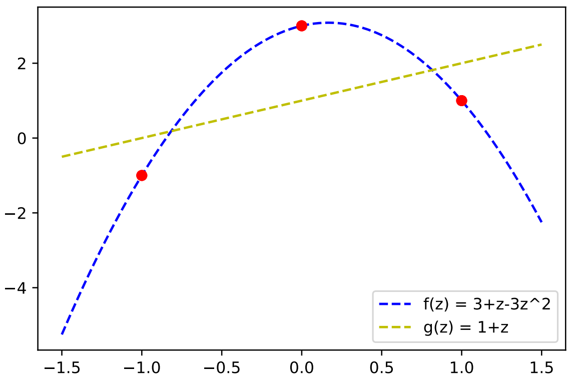
     (II) $\Rightarrow x_1 = 3$
     (I) $\Rightarrow \underbrace{x_1}_{=3} - x_2 + x_3 = -1 \quad \Leftrightarrow \quad x_3 = x_2 - 4$
     (III) $\Rightarrow \underbrace{x_1}_{=3} + x_2 + \underbrace{x_3}_{=x_2-4} = 1 \quad \Leftrightarrow \quad x_2 = 1$

- All in all:
$$x = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ -3 \end{pmatrix}.$$
- Thus, our fitting parabola is given by:
$$f(z) = 3 + z - 3z^2.$$



2. LINEAR REGRESSION:
   - Aim: Find c_0, c_1 such that
   $$f(z_i) = c_0 + c_1 z_i \approx y_i \quad \forall i = 1, 2, 3.$$
   - As above we define
   $$A := (a_{ij})_{ij} = (z_i^{j-1})_{ij} = \begin{pmatrix} 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix},$$
   $$x := \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}, \quad b := \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}.$$
   - Then by construction of A, x and b we find:
   $$\|Ax - b\|_2^2 = \left\| \underbrace{\begin{pmatrix} 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \end{pmatrix}\begin{pmatrix} c_0 \\ c_1 \end{pmatrix}}_{= \begin{pmatrix} c_0 + z_1 c_1 \\ c_0 + z_2 c_1 \\ c_0 + z_3 c_1 \end{pmatrix} = \begin{pmatrix} f(z_1) \\ f(z_2) \\ f(z_3) \end{pmatrix}} - \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \right\|_2^2 = \left\| \begin{pmatrix} f(z_1) - y_1 \\ f(z_2) - y_2 \\ f(z_3) - y_3 \end{pmatrix} \right\|_2^2 \overset{\text{Def. of } \|\cdot\|_2}{=} \sum_{i=1}^{3} (f(z_i) - y_i)^2.$$
   - We now solve the normal equation
   $$A^T A x = A^T b$$
   to determine the solution of the least squares problem:
   $$\min_{x=(c_0,c_1)\in\mathbb{R}^2} \|Ax - b\|_2^2 = \min_{c_0,c_1} \sum_{i=1}^{3} [(c_0 + c_1 z_i) - y_i]^2.$$
   - We find
   $$A^T A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix},$$
   $$A^T b = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}.$$

- Thus:
$$A^T A x = A^T b \quad \Leftrightarrow \quad \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \Leftrightarrow \quad \begin{matrix} c_0 = 1 \\ c_1 = 1 \end{matrix}.$$
- Therefore the best linear fit in the least squares sense is given by
$$g(z) = 1 + z.$$



Ex 51 Linear Algebra

Gram-Schmidt-Algorithm

The Gram-Schmidt algorithm is an algorithm to compute a QR-decomposition of some matrix A. The basic idea is to successively built up an orthogonal system from a given set of linearly independent vectors. Those are, in this case, given by the columns of an invertible matrix $A = [a_1, \dots, a_n] \in \mathbb{R}^{n \times n}$. We choose the first column as starting point for the algorithm and set $\widetilde{q_1} := a_1$. Of course, in order to generate an orthogonal matrix Q we have to rescale the vector and set $q_1 := \frac{\widetilde{q_1}}{\|\widetilde{q_1}\|}$. The successive vectors $\widetilde{q}_k$ are generated by subtracting all the shares $a_k^\top q_\ell$ of the previous vectors q_ℓ from the column a_k, i.e.
$$\widetilde{q}_k := a_k - \sum_{\ell=1}^{k-1} a_k^\top q_\ell \, q_\ell.$$

The following algorithm computes a QR-decomposition of some matrix $A \in \mathbb{R}^{n \times n}$.

```
1  $r_{11} \leftarrow \|a_1\|$;
2  $q_1 \leftarrow \frac{a_1}{r_{11}}$;
3  for $k = 2, \dots, n$ do
4      for $\ell = 1, \dots, k-1$ do
5          $r_{\ell k} \leftarrow a_k^\top q_\ell$
6      end
7      $\widetilde{q}_k \leftarrow a_k - \sum_{\ell=1}^{k-1} r_{\ell k} \, q_\ell$;
8      $r_{kk} \leftarrow \|\widetilde{q}_k\|$;
9      $q_k \leftarrow \frac{\widetilde{q}_k}{r_{kk}}$;
10 end
```

Algorithm 1: Gram-Schmidt algorithm

1. Please check that the matrix $Q := [q_1, \dots, q_n] \in \mathbb{R}^{n \times n}$ is orthogonal, i.e. that $Q^T Q = I_n$.

   Hint: Show $\|q_i\| = 1$ first, and then perform an induction proof using the induction assumption that q_k is orthogonal to all previous $q_1, \dots, q_{k-1}$.

2. Let $R := (r_{\ell k})_{\ell \le k}$ be the upper triangular matrix which results from the Gram-Schmidt algorithm. Please show that the algorithm provides a QR decomposition, i.e., that $QR = A$.

   Hint: It suffices to show $Qr_k = a_k$, where r_k (a_k) is the k-th column of R (A).

Solution:

1. $Q = \begin{pmatrix} | & & | \\ q_1 & \dots & q_n \\ | & & | \end{pmatrix}$ orthogonal $\Leftrightarrow \quad Q^TQ = I \quad \Leftrightarrow \quad q_i^T q_j = \delta_{ij}$

   a) $\|q_i\| = \|\frac{\tilde{q}_i}{\|\tilde{q}_i\|}\| = \frac{\|\tilde{q}_i\|}{\|\tilde{q}_i\|} = 1 \;\; \forall i$
   $\Rightarrow$ diagonal of Q^TQ contains only 1's.

   b) **Induction Basis** $(k = 1)$ $\{q_1\}$ trivially orthogonal system
   Induction Step $(k \mapsto k + 1)$
   Assume $\{q_1, \dots, q_k\}$ are pairwise orthogonal and let

$$\tilde{q}_{k+1} := a_{k+1} - \sum_{l=1}^{k} r_{lk+1} q_l \quad \text{and} \quad q_{k+1} := \frac{1}{\tilde{q}_{k+1}} \|\tilde{q}_{k+1}\|.$$

   Now let $j \in \{1, \dots, k\}$, then

$$q_j{}^T q_{k+1} = \left[q_j{}^T a_{k+1} - \sum_{l=1}^{k} r_{lk+1} \underbrace{q_j{}^T q_l}_{=\delta_{jl} \text{ by induction assumption}} \right] \frac{1}{\|\tilde{q}_{k+1}\|}$$

$$= [q_j^T a_{k+1} - \underbrace{r_{jk+1}}_{=q_j^T a_{k+1}}] \frac{1}{\|\tilde{q}_{k+1}\|} = 0$$

2. Show hint: For all k we have:

$$Qr_k = \begin{pmatrix} | & & | \\ q_1 & \dots & q_n \\ | & & | \end{pmatrix} \begin{pmatrix} r_{1k} \\ \vdots \\ r_{kk} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \sum_{l=1}^{k} r_{lk} q_l = \underbrace{r_{kk} q_k}_{=\tilde{q}_k = a_k - \sum_{l=1}^{k-1} r_{lk} q_l} + \sum_{l=1}^{k-1} r_{lk} q_l$$

$$= a_k - \sum_{l=1}^{k-1} r_{lk} q_l + \sum_{l=1}^{k-1} r_{lk} q_l = a_k$$

Ex 52

****Bonus:** Operation Count in Classical Gram Schmidt Algorithm**
Consider the classical Gram-Schmidt algorithm for a matrix $A \in \mathbb{R}^{m \times n}$ with $\ker(A) = \{0\}$.
Count each addition, subtraction, multiplication, division and square root that is computed in the classical Gram Schmidt algorithm to obtain a *reduced* QR factorization of A.

Solution:

```
1  $r_{11} = \|a_1\|$   m products, m-1 addition, 1 square root
2  $q_1 = \frac{a_1}{r_{11}}$   m divisons
3  for $k = 2, \dots, n$   n-1 times do
4      for $\ell = 1, \dots, k-1$   k-1 times do
5          $r_{\ell k} = a_k^T q_\ell$   m products, m-1 additions
6      end
7      $\widetilde{q}_k = a_k - \sum_{\ell=1}^{k-1} r_{\ell k}\, q_\ell$   (k-1) $\times$ m products, (k-2) $\times$ m additions)
8      $r_{kk} = \|\widetilde{q}_k\|$   m products, m-1 addition, 1 square root
9      $q_k = \frac{\widetilde{q}_k}{r_{kk}}$   m divisons
10 end
```

Ex 53

Consider the system

$$\begin{pmatrix} \frac{1}{2} & -2 & 0 \\ 2 & 8 & -2 \\ 1 & 0 & 2 \end{pmatrix} x = \begin{pmatrix} -1 \\ 10 \\ 4 \end{pmatrix}.$$

1. Define a matrix A and a vector b, so that this system reads as $Ax = b$. Then compute an LU-decomposition of A by applying Gaussian elimination with **row pivoting**. Denote the respective matrices L, U and P, such that $PA = LU$. (*Hint:* Verify the desired properties of the factor matrices and test whether $PA = LU$ holds.)

2. Use the result from the LU-decomposition to determine an x which solves $Ax = b$. (*Hint:* Test whether $Ax = b$ holds.)

Solution:

1. $$A = \begin{pmatrix} \frac{1}{2} & -2 & 0 \\ 2 & 8 & -2 \\ 1 & 0 & 2 \end{pmatrix}, \; b = \begin{pmatrix} -1 \\ 10 \\ 4 \end{pmatrix} \qquad (1+1P)$$

2. $$\left(\begin{array}{ccc|c} \frac{1}{2} & -2 & 0 & -1 \\ 2 & 8 & -2 & 10 \\ 1 & 0 & 2 & 4 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array}, \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\overset{\text{(I)}<-->\text{(II)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} 2 & 8 & -2 & 10 \\ \frac{1}{2} & -2 & 0 & -1 \\ 1 & 0 & 2 & 4 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array}, \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$$

$$\overset{\text{(II')}=\text{(II)}-\frac{1}{4}\text{(I)},\ \text{(III')}=\text{(II)}-\frac{1}{2}\text{(I)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} 2 & 8 & -2 & 10 \\ \frac{1}{4} & -4 & \frac{1}{2} & -3.5 \\ \frac{1}{2} & -4 & 3 & -1 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array}, \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$$

$$\overset{\text{(III)}'=\text{(III)}-\text{(II)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} 2 & 8 & -2 & 10 \\ \frac{1}{4} & -4 & \frac{1}{2} & -3.5 \\ \frac{1}{2} & 1 & 2.5 & 2.5 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array}, \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}$$

   We obtain

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} (1P), \quad L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix} (1P), \quad U = \begin{pmatrix} 2 & 8 & -2 \\ 0 & -4 & \frac{1}{2} \\ 0 & 0 & 2.5 \end{pmatrix} (1P)$$

3. Test: (1+1P)

4. $$\begin{aligned} x_3 = 1 \quad &\Rightarrow \quad -4x_2 = -3.5 - 0.5 = -4 \cdot 1 \quad \Rightarrow \quad x_2 = 1 \\ &\Rightarrow \quad 2x_1 = 10 - 8 + 2 = 4 \quad \Rightarrow \quad x_1 = 2 \\ &\Rightarrow \quad x^* = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \quad (1+1+1P) \end{aligned}$$

Ex 54

Consider the system

$$\begin{pmatrix} 0 & -2 & 1 \\ 2 & 2 & 0 \\ -2 & -\frac{3}{2} & 0 \end{pmatrix} x = \begin{pmatrix} -5 \\ 6 \\ -5 \end{pmatrix}.$$

1. Define a matrix A and a vector b, so that this system reads as $Ax = b$. Then compute an LU-decomposition of A by applying Gaussian elimination with **row pivoting**. Denote the respective matrices L, U and P, such that $PA = LU$. (*Hint: Verify the desired properties of the factor matrices and test whether $PA = LU$ holds.*)

2. Use the result from the LU-decomposition to determine an x which solves $Ax = b$. (*Hint: Test whether $Ax = b$ holds.*)

Solution:

$$\left(\begin{array}{ccc|c} 0 & -2 & 1 & -5 \\ 2 & 2 & 0 & 6 \\ -2 & -1,5 & 0 & -5 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$\overset{\text{(III)}\leftrightarrow\text{(I)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} -2 & -1,5 & 0 & -5 \\ 2 & 2 & 0 & -6 \\ 0 & -2 & 1 & -5 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

$$\overset{\text{(II)}'=\text{(II)}+\text{(I)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} -2 & -1,5 & 0 & -5 \\ -1 & 0,5 & 0 & 1 \\ 0 & -2 & 1 & -5 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)}' \\ \text{(III)} \end{array} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

$$\overset{\text{(III)}'=\text{(III)}+4\text{(II)}}{\rightsquigarrow} \left(\begin{array}{ccc|c} -2 & -1,5 & 0 & -5 \\ -1 & 0,5 & 0 & 1 \\ 0 & -4 & 1 & -1 \end{array}\right) \begin{array}{c} \text{(I)} \\ \text{(II)}' \\ \text{(III)}' \end{array} \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \; (1P) \; , L = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -4 & 1 \end{pmatrix} \; (3P) \; , U = \begin{pmatrix} -2 & -1,5 & 0 \\ 0 & 0,5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \; (3P)$$

$$\begin{aligned} x_3 = -1 \quad &\Rightarrow \quad 0,5x_2 = 1 \quad \Rightarrow \quad x_2 = 2 \\ &\Rightarrow \quad -2x_1 - 1,5 \cdot 2 = -5 \quad \Rightarrow \quad x_1 = 1 \\ &\Rightarrow \quad x^* = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \quad (1+1+1P) \end{aligned}$$

Ex 55

Solving Linear Systems using LU Decomposition
Consider the following linear systems.

1.
$$\begin{aligned} 2x_1 + x_2 + 3x_3 &= -3 \\ x_1 - x_2 - x_3 &= 4 \\ 3x_1 - 2x_2 + 2x_3 &= 5 \end{aligned}$$

2.
$$\begin{aligned} x_1 + 2x_2 + 2x_3 &= 1 \\ 2x_1 + x_3 &= 3 \\ 3x_1 + 2x_2 + 3x_3 &= 4 \end{aligned}$$

3.
$$\begin{aligned} x_1 + x_2 + 2x_3 &= 2 \\ 1x_1 - x_2 &= 0 \\ 2x_1 + 2x_3 &= 1 \end{aligned}$$

Cast them into the form $Ax = b$ and compute the LU-decomposition of A by rigorously applying Algorithm 3 (i.e., use the pivot element determined in Line 8):

- Find the matrices L, U and P such that $PA = LU$.
- Then determine the solution set $S := \{x \in \mathbb{R}^n : Ax = b\}$.

Hint: System 2. does not have a *unique* solution (but infinitely many). Determine the set of vectors $x \in \mathbb{R}^3$ for which the linear system 2. is valid.

```
INPUT: $A \in \mathbb{R}^{n \times n}$ (and $b \in \mathbb{R}^n$)
OUTPUT: LU decomposition $PA = LU$ (and if $Ax = b$ is uniquely solvable the solution $x \in \mathbb{R}^n$)

# FACTORIZATION
initialize piv = [1,2,...,n]
for $j = 1, ..., n-1$ do
    # Find the j-th pivot
    $k_j := \arg\max_{k \geq j} |a_{kj}|$
    if $a_{k_j j} \neq 0$ then
        # Swap rows
        A[$k_j$,:] $\leftrightarrow$ A[j,:]
        # by hand we also transform b on the fly
        b[$k_j$] $\leftrightarrow$ b[j]
        piv[$k_j$] $\leftrightarrow$ piv[j]
        # Elimination
        for $k = j+1, \ldots, n$ do
            $\ell_{kj} := a_{kj}/a_{jj}$
            $a_{kj} = \ell_{kj}$
            for $i = j+1, \ldots, n$ do
                $a_{ki} = a_{ki} - \ell_{kj} a_{ji}$
            end
            # by hand we also transform b on the fly
            ($b_k = b_k - \ell_{kj} b_j$)
        end
    end
end

# SOLVE
...
```

Algorithm 2: In-place Gaussian Elimination with Row Pivoting: Factor and Solve

Solution:

You can later use your Python Code to check your LU decomposition $PA = LU$ with all intermediate steps. Therefore we just copy the factors and solution here.

1. Unique solution:

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

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ 1/3 & -1/7 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 3 & -2 & 2 \\ 0 & 7/3 & 5/3 \\ 0 & 0 & -10/7 \end{pmatrix}$$

Solving steps yields: $x = (1, -2, -1)^T$, thus $S = \{(1, -2, -1)^T\}$

2. Infinitely many solutions: The algorithm outputs the following arrays

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

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & -1 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 3 & 2 & 3 \\ 0 & 4/3 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

and we obtain

$$z := L^{-1}P^\top b = \begin{pmatrix} 4 \\ -1/3 \\ 0 \end{pmatrix}.$$

Therefore we see that the system $Ux = z$ has infinitely many solutions (last zero row in U and zero in same row in z). By solving $Ux = z$ we find

$$\begin{array}{ll} \text{(II)} & \Rightarrow \quad 4/3x_2 + 1x_3 = -1/3 \quad \Rightarrow \quad x_2 = -\frac{1}{4}(1 + 3x_3) \\ \text{(I)} & \Rightarrow \quad 3x_1 + 2x_2 + 3x_3 = 4 \quad \Rightarrow \quad x_1 = \frac{1}{3}(4 - 2x_2 - 3x_3) = \frac{3}{2} - \frac{1}{2}x_3 \end{array}$$

$$\begin{aligned} \Rightarrow \quad S &:= \{x \in \mathbb{R}^3 : Ax = b\} \\ &= \{x \in \mathbb{R}^3 : x_1 = \tfrac{1}{2}(3 - x_3), \quad x_2 = -\tfrac{1}{4}(1 + 3x_3), \quad x_3 \in \mathbb{R}\} \\ &\overset{s:=x_3\in\mathbb{R}^3}{=} \left\{ \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{4} \\ 0 \end{pmatrix} + s \begin{pmatrix} -\frac{1}{2} \\ -\frac{3}{4} \\ 1 \end{pmatrix} : s \in \mathbb{R} \right\}, \text{(i.e., we have infinitely many solutions!)} \end{aligned}$$

3. No solution: $S = \emptyset$

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

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

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

and we obtain

$$z := L^{-1}P^\top b = \begin{pmatrix} 1 \\ -1/2 \\ 1 \end{pmatrix}.$$

Thus, last row in U is a zero row but $1 \neq 0$ in z.

Ex 56

Proof for LU decomposition (with row pivoting)
Let $m \in \mathbb{N}$. As above, let $P_{ik_i} \in \mathbb{R}^{m\times m}$ be the permutation matrix which results from interchanging the i-th and k_i-th column ($k_i \geq i$) of the identity matrix in $\mathbb{R}^{m\times m}$. Further for $\ell_j := (0, \dots, 0, \ell_{j+1,j}, \dots, \ell_{m,j})^\top \in \mathbb{R}^m$ and the j-th unit vector $e_j \in \mathbb{R}^m$, let $L_j := I + \ell_j e_j^\top \in \mathbb{R}^{m\times m}$. Then show that for all $1 \leq j < i \leq k_i \leq m$ we have

$$P_{ik_i} L_j = \widehat{L}_j P_{ik_i}$$

where $\widehat{L}_j := I + (P_{ik_i}\ell_j)e_j^\top$.

Solution:

We find

$$\begin{aligned} P_{ik_i} L_j &= P_{ik_i}\left(I + \ell_j e_j^\top\right) \\ &= P_{ik_i} + P_{ik_i}\ell_j e_j^\top \\ &= P_{ik_i} + P_{ik_i}\ell_j e_j^\top P_{ik_i}^\top P_{ik_i} \\ &= (I + P_{ik_i}\ell_j e_j^\top P_{ik_i}^\top)P_{ik_i} \\ &= (I + P_{ik_i}\ell_j (P_{ik_i} e_j)^\top)P_{ik_i}. \\ &= (I + P_{ik_i}\ell_j e_j^\top)P_{ik_i}. \end{aligned}$$

Since $j < i \leq k_i$ we find that $P_{ik_i}e_j = e_j$, since only zeroes are swapped.

Ex 57 Linear Systems, Factor and Solve

Permutation Matrices and Row Swap
A matrix $P \in \mathbb{R}^{m\times m}$ is called *permutation matrix* if it has exactly one entry of 1 in each row and each column and 0s elsewhere.

1. **Sparse representation (CSR Format):** A permutation matrix $P = (p_{ij})_{ij} \in \mathbb{R}^{m\times m}$ can be represented by an m-dimensional vector piv$\in \{1, 2, \dots, m\}^m$ in the following way:
$$\texttt{piv}_i = j \quad :\Leftrightarrow \quad p_{ij} = 1.$$
Given a permutation matrix P with its sparse representation `piv`. Determine the sparse representation, say pivT$\in \{1, 2, \dots, m\}^m$, of the transpose $P^T = (\widetilde{p}_{ij})_{ij}$, so that
$$\texttt{pivT}_i = j \quad :\Leftrightarrow \quad \widetilde{p}_{ij} = 1.$$
Hint: Have a look at the routine `numpy.argsort()`.
2. **Inverse:** Show that the inverse of a permutation matrix $P \in \mathbb{R}^{m\times m}$ is given by its transpose, i.e., $P^T = P^{-1}$.
3. **Row Swap**: Let $P_{jk} \in \mathbb{R}^{m\times m}$ be the permutation matrix which results from interchanging the j-th and k-th column ($k \geq j$) of the identity matrix in $\mathbb{R}^{m\times m}$. Thus if its applied to a matrix $A \in \mathbb{R}^{m\times n}$ it interchanges the j-th and k-th row of A. Show that
$$P_{jk}^{\top} = P_{jk}.$$
In particular we find $P_{jk} = P_{jk}^{-1}$, i.e., P_{jk} is self-inverse.

Solution:

1. $\texttt{piv}_i = j \Leftrightarrow 1 = p_{ij} = \widetilde{p}_{ji} \Leftrightarrow: \texttt{pivT}_j = i$
2. By definition, the columns of a permutation matrix are given by the m unit vectors (in potentially permuted order), which are orthonormal.
3. Let $P_{jk} = (q_{i\ell})_{i\ell}$ and $P_{jk}^T = (\widetilde{q}_{i\ell})_{i\ell}$, then by definition we find
$$q_{i\ell} = \begin{cases} 1: & (i = \ell, k \neq i \neq j) \text{ or } (i = j, \ell = k) \text{ or } (i = k, \ell = j) \\ 0: & \text{else} \end{cases}$$
and therefore
$$\widetilde{q}_{i\ell} = q_{\ell i} = \begin{cases} 1: & (\ell = i, k \neq \ell \neq j) \text{ or } (\ell = j, i = k) \text{ or } (\ell = k, i = j) \\ 0: & \text{else} \end{cases}.$$
Thus, we obviously find $q_{i\ell} = \widetilde{q}_{i\ell}$.

Ex 58 Linear Systems, Factor and Solve, Python

The Cholesky Decomposition for SPD Matrices

1. Find SciPy routines to perform the Cholesky factorization and solution steps separately. Non-obligatory: Implement them on your own (algorithms can be found on Wikipedia).
2. Compare the performance of the Cholesky routines to the LU routines from SciPy by testing them on large symmetric and positive definite matrices $A \in \mathbb{R}^{n\times n}$ (e.g., $A = B^TB + \delta I$ for some random $B \in \mathbb{R}^{n\times n}$ and $\delta > 0$) and some (random) right-hand side $b \in \mathbb{R}^n$.

Solution:

```
import numpy as np
import scipy.linalg as linalg
from time import time

def Aspd(n,delta):
    B = np.random.rand(n,n)
    return B.T@B + delta * np.eye(n)

def FacSol(A,b, method = "lu"):
    """
    ...
    """
    # choose correct SciPy routines
    if method == "lu":
        factor = lambda A : linalg.lu_factor(A)
        solve = lambda tup, b : linalg.lu_solve(tup, b, overwrite_b=True)
    else:
        factor = lambda A : linalg.cho_factor(A)
        solve = lambda tup, b : linalg.cho_solve(tup, b, overwrite_b=True)

    # factor
    tfac = time()
    tup = factor(A)
    tfac = time()-tfac

    # solve
    tsolve = time()
    x = solve(tup, b)
    tsolve = time()-tsolve

    return {"x": x, method: tup, "t": (tfac, tsolve)}

if __name__ == "__main__":
    n = 1000
    delta = 0.5
    runs = 10
    for i in range(runs):
        print("run", i)
        A = Aspd(n, delta)
        b = np.random.rand(n)
        print("dim, method, [t_factor t_solve], Ax==b")
        for method in ["lu", "cho"]:
            data = FacSol(A.copy(),b.copy(), method)
            print(n, method, "\t ",np.round(data["t"],4),"\t   ",np.allclose(A.dot(data["x"]),
     b))
        print("---------------------------------")
```

Ex 59

Curve Fitting using reduced QR Factorization

Assume you were given some numpy.ndarray called "data" containing measurements $(z_1, y_1), \dots, (z_m, y_m) \in \mathbb{R} \times \mathbb{R}$. Further assume that this data has shape (2,m) so that the first row data[0,:] contains the z_i values and the second row data[1,:] contains the y_i values.

1. Implement a function poly_curvefit(data, p) which computes a polynomial fit to the data.

   The **input** data shall have the form as described above and p shall determine the polynomial used to fit the data. More precisely, p shall be a list [p1,..., pn] containing $n \leq m$ *distinct* natural numbers between 0 and $m-1$ which determine the polynomial model f by

$$f(z) = c_1 z^{p_1} + c_2 z^{p_2} + \ldots + c_k z^{p_n} = \sum_{j=1}^{n} c_j f_j(z) \quad \text{with} \quad f_j(z) := z^{p_j}.$$

For example, `p = [0,1]` amounts for a linear model

$$f(z) = c_1 z^{p_1} + c_2 z^{p_2} = c_1 z^0 + c_2 z^1 = c_1 + c_2 z^1.$$

Then the **function** `poly_curvefit(data, p)` shall determine appropriate coefficients $x = (c_1, \ldots, c_n)$ by following this recipe:

a) Assemble the vector $b = (y_1, \ldots, y_m) \in \mathbb{R}^m$.

b) Assemble the matrix $A = (a_{ij})_{ij} \in \mathbb{R}^{m \times n}$, where $a_{ij} := f_j(z_i)$.

c) The vector b may not be in the image of A. Therefore compute the reduced QR factorization of A and solve the auxiliary problem $\widehat{R}x = \widehat{Q}^T b$. You can use `qr_factor` and `solve_tri` from previous exercises (or appropriate SciPy routines).
Remark: We will later learn that this yields precisely the least squares solution.

2. Test your routine by fitting the data

z	-2	-1	0	1	2
y	-2	1	-1	2	6

with different polynomials of your choice. Plot the measurements and the fitting polynomial into one figure.

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt

# the given data containing 5 measurements
data = np.array([[-2.,-1., 0., 1., 2.],
                 [-2., 1., -1., 2., 6.]])
#data = np.array([[-1., 0., 1.],
#                  [-1., 3., 1.]])

# polynomial curve fitting
def poly_curve_fit(data, p, own=True):
    """
    INPUT:
        numpy.ndarray data of shape (2,m) with
            data[0,:] = (z_1, ..., z_m) explanatory/independent variables
            data[1,:] = (y_1, ..., y_m) response/dependent variables
        list p = [p1, p2,..., pn] determining the polynomial model
            f_c(z) = c1 * z^p1 + ... + cn * z^pn
        own : switch to use either our approach with (reduced) QR-decomposition
              or SciPy's routine to solve the least squares problem

    OUTPUT:
        numpy.ndarray c of shape (n,) such that
           c = argmin_c sum_i (f_c(z_i) - y_i)^2

    """
    # (a) assemble the vector b
    b = data[1,:]

    # (b) assemble the matrix A
    z_i = data[0,:][np.newaxis].T
    A = z_i**p

    # (c) determine c by solving using QR Decomposition
    if own==True:
        # "factor_qr"
        import scipy.linalg as linalg
        Q, R = linalg.qr(A)
        m, n = len(b), len(p)
        Q, R = Q[0:m,0:n], R[0:n,0:n]
        # "solve_qr"
        c = linalg.solve_triangular(R, Q.T @ b )
    else:
        import scipy.linalg as linalg
        c , res , rnk , s = linalg.lstsq(A,b)

    # (d) Plot measurements and fitting polynomial into one figure
    print("\n---------------------------\nExample: p =", p, "\nown =", own )
    Z = np.linspace(-2, 2, 50) # other z values
    Y = (Z[np.newaxis].T**p).dot(c) # evaluate model on Z
    plt.figure()
    plt.title("Polynomial degree = " + str(max(p)))
    plt.plot(z_i, b, 'ro')
    plt.plot(Z, Y, 'b')
    plt.show()

    return c

if __name__ == "__main__":
    # different choices of polynomials
    P = [[1], [0,1], [0,1,2], [0, 1, 2, 3], [0, 1, 2 ,3, 4] ]

    # apply the routine for all those choices
    for p in P:
        for own in [True, "Scipy <lstsq>"]:
            poly_curve_fit(data, p, own = own)
        print("\n")
```

Ex 60 Linear Algebra, Python

Classical Gram Schmidt Algorithm

Let $A \in \mathbb{R}^{m\times n}$ with $\ker(A) = \{0\}$. Then the classical Gram Schmidt algorithm to compute a *reduced* QR-decomposition of A reads as follows:

```
1  $r_{11} = \|a_1\|$
2  $q_1 = \frac{a_1}{r_{11}}$
3  for  $k = 2, \dots, n$ do
4      for  $\ell = 1, \dots, k-1$ do
5          $r_{\ell k} = a_k^\top q_\ell$
6      end
7      $\widetilde{q}_k = a_k - \sum_{\ell=1}^{k-1} r_{\ell k}\, q_\ell$
8      $r_{kk} = \|\widetilde{q}_k\|$
9      $q_k = \frac{\widetilde{q}_k}{r_{kk}}$
10 end
```

Task:

1. Implement this algorithm as a function `Q, R = qr_factor(A)`, which takes a matrix A as input and returns the matrices $\widehat{Q}$ and $\widehat{R}$.

2. Legendre polynomials: Run your algorithm on the following example.
   a) Find a NumPy function to generate an equidistant grid on the interval $(-1, 1)$.
   b) Find a NumPy function to generate a Vandermonde matrix `A` based on this grid.
   c) Compute `Q,R = qr_factor(A)` and plot the columns of `A` and `Q` over the interval $(-1, 1)$ into one figure.
   d) Compute `Q.TQ` and `Q@R - A` and check whether the first yields the identity and the latter the zero-matrix.
   Hint: You can use `numpy.allclose()` *to check whether two* `numpy.ndarray`*'s are equal up to a certain tolerance.*

3. Find a SciPy function to compute a full QR factorization. To obtain the reduced QR you need to set the parameters accordingly.

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt


# in scipy: scipy.linalg.qr(A, mode="economic")
def qr_factor(A):
    """
    Computes a (reduced) QR-decomposition of a (mxn)-matrix with m>=n
    via Gram-Schmidt Algorithm assumed rank(A)=n.

    Parameters
    ----------
    A : (mxn) matrix with m>=n, rank(A)=n

    Returns
    -------
    Q : (mxn) with orthonormal columns
    R : (nxn) upper triangular matrix
    """
    m, n = A.shape
    R = np.zeros((n, n))
    Q = np.zeros((m, n))

    R[0, 0] = np.linalg.norm(A[:, 0])
    Q[:, 0] = A[:, 0] / R[0, 0]

    for k in range(1, n):
        for l in range(0, k):
            R[l, k] = A[:, k] @ Q[:, l]
        q = A[:, k] - Q @ R[:, k]
        R[k, k] = np.linalg.norm(q)
        Q[:, k] = q / R[k, k]

    return Q, R


def example_vander_legendre(gridpoints, degree):
    grid = np.linspace(-1, 1, 200)
    A = np.vander(grid, degree, increasing=True)
    plt.figure("Standard Monomials")
    plt.plot(grid, A)
    plt.legend(range(A.shape[1]))

    Q, R = qr_factor(A)
    plt.figure("Variant of Legendre Polynomials")
    Q, R = qr_factor(A)
    plt.plot(grid, Q, "--")
    plt.legend(range(Q.shape[1]))
    plt.show()

    return A, Q, R


def test_qr(A, Q, R, mode="full"):
    print("\nTest 1: Q.TQ = I is",
          np.allclose(Q.transpose()@Q, np.eye(A.shape[1])))
    print("\nTest 2: QR = A is",
          np.allclose(Q@R, A))


if __name__ == "__main__":
    m, n = 200, 5
    A, Q, R = example_vander_legendre(m, n)
    test_qr(A, Q, R)
```

Implementation Matters
In order to solve the problem $Ax = b$, there are plenty of algorithms available. In this exercise we invoke several SciPy routines to solve linear systems numerically. Thereby, we will learn that different algorithms, or even different implementations of the same algorithm, can have a huge effect on the efficiency.

Consider the following test example: The matrix $A \in \mathbb{R}^{n\times n}$ with constant diagonals given by

$$A = \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

and the right-hand side vector

$$b = (1, 0, \cdots, 0, 1)^\top \in \mathbb{R}^n.$$

In this case we know that the unique[1] solution $x^* = A^{-1}b$ is given by

$$x^* = (1, 1, \cdots, 1)^\top \in \mathbb{R}^n.$$

Implement the following options to solve the problem $Ax = b$ (i.e., given A and b from above, find a numerical solution $\tilde{x}$ such that $A\tilde{x} \approx b$):

1. Dense: Work with A as dense `numpy.ndarray`.
   a) The forbidden way: Find a SciPy function to compute a numerical inverse and apply it to b to obtain $\tilde{x}$.
   b) The default way: Find a general solver for linear systems.
   c) Structure exploiting solver I: Find a way to inform this general solver about the fact that A is *positive definite*.[2]
   d) Structure exploiting solver II: Find another solver which exploits the fact, that A has constant diagonals.
2. Sparse: Exploit the sparsity of the matrix and work with A as `scipy.sparse.csr_matrix`.
   a) The forbidden way: Find a SciPy function to compute the inverse of a sparse matrix and apply it to b to obtain $\tilde{x}$.
   b) The default way: Find a general solver for sparse linear systems.
   c) Structure exploiting solver: Find a function that implements the *conjugate gradient (cg)* method to solve the system iteratively. Play with the optional parameter `<maxiter>` to restrict the number of iterations. What do you observe?

Run your code for different dimensions n (especially large $n \geq 10^5$) and:

1. measure the time needed for each of your solving routine,
2. and find a SciPy function to compute the error $\frac{1}{n}\|\tilde{x} - x^*\|_2$ for each solving routine.

Remark: For now, it is not important to understand how the algorithms work. We'll learn more about them throughout this course. Here, you learn to find appropriate SciPy functions which solve your problem and to serve the interfaces of these functions.

Solution:

```
import numpy as np
import scipy.linalg as linalg
from scipy.sparse import linalg as spla
import scipy.sparse as sparse
import timeit


def error(approx, exact):
```

[1]One can show that the matrix A is invertible.
[2]We'll learn about this property later.

```python
    return linalg.norm(approx-exact)/n


def solver_wrapper(a, b, solver="general"):
    if solver == "inv":
        a_inv = linalg.inv(a)
        return a_inv @ b
    elif solver == "general":
        return linalg.solve(a, b)
    elif solver == "spd":
        return linalg.solve(a, b, assume_a="pos")
    elif solver == "toeplitz":
        return linalg.solve_toeplitz(a[0, :], b)
    elif solver == "sparse_inv":
        a_inv = spla.inv(a)
        return a_inv @ b
    elif solver == "sparse_general":
        return spla.spsolve(a, b)
    elif solver == "cg":
        return spla.cg(a, b, maxiter=50)[0]
    else:
        raise Exception("Unknown solver")


def experiment(solver, a, b, num_runs):
    print("\nSolver:  {}".format(solver), end="\n"+"-"*25+"\n")
    wall_time = timeit.timeit('x = solver_wrapper(a,b,solver=solver)',
                             globals=globals(), number=num_runs)
    print("Time:    {:2.2f} seconds".format(wall_time))
    x = solver_wrapper(a, b, solver=solver)
    print("Error:   {:.2e}".format(error(x, exact)))


if __name__ == "__main__":

    n = 2500

    b = np.zeros(n)
    b[0], b[-1] = 1, 1
    exact = np.ones(n)

    num_runs = 5

    # DENSE
    a = -np.eye(n, k=-1) + 2 * np.eye(n) + -np.eye(n, k=1)
    solvers = ("inv", "general", "spd", "toeplitz", "cg")
    for solver in solvers:
        experiment(solver, a, b, num_runs)

    # SPARSE
    a = sparse.csr_matrix(a)
    sparse_solvers = ("sparse_inv", "sparse_general", "cg")
    for solver in sparse_solvers:
        experiment(solver, a, b, num_runs)
```

Ex 62 Linear Systems, Factor and Solve, Python

Solve with LU decomposition [Direct method]

1. Implement a routine `lu,piv = lu_factor(A)` which computes the LU decomposition $PA = LU$ (by applying Gaussian elimination with row pivoting; see Algorithm 3) for a given matrix $A \in \mathbb{R}^{n\times n}$ and another routine `lu_solve((lu, piv), b)` which takes the output of `lu_factor(A)` and returns the solution x of $Ax = b$ for some $b \in \mathbb{R}^n$ (in case the system admits an *unique* solution).
   - Store L and U in one array `lu` and the permutation P as sparse representation in an array `piv`.
   - If the system $Ax = b$ admits an unique solution then compute it by using your routine `solve_tri` from previous

exercises or an appropriate SciPy routine. If the system is not uniquely solvable, check whether the system has infinitely many or no solution and give the user a respective note.

- *Hint:* With the numpy routines `numpy.triu` and `numpy.tril` you can extract the factors L and U from the array `lu`. Also observe that we expect A to be of square format (for simplicity).

2. Test your routine at least on the systems which you were asked to solve by hand previously. Verify that $PA = LU$ and potentially $Ax = b$. For this purpose you can use `numpy.allclose()`.

3. Find SciPy routines to perform the factorization and solution steps and compare to your routine.

```
INPUT: $A \in \mathbb{R}^{n \times n}$
OUTPUT: LU decomposition $PA = LU$

# FACTORIZATION
initialize piv = [1,2,...,n]
for $j = 1, ..., n-1$ do
    # Find the j-th pivot pivot
    $k_j := \arg\max_{k \geq j} |a_{kj}|$
    if $a_{k_j j} \neq 0$ then
        # Swap rows
        A[$k_j$,:] $\leftrightarrow$ A[j,:]
        piv[$k_j$] $\leftrightarrow$ piv[j]
        # Elimination
        for $k = j+1, \ldots, n$ do
            $\ell_{kj} := a_{kj}/a_{jj}$
            $a_{kj} = \ell_{kj}$
            for $i = j+1, \ldots, n$ do
                $a_{ki} = a_{ki} - \ell_{kj} a_{ji}$
            end
        end
    end
end
```

Algorithm 3: In-place Gaussian Elimination with Row Pivoting for implementing Factorization (same as above just without the b vector.)

Solution:

```python
import numpy as np
import scipy.linalg as linalg

def lu_factor(A, printsteps=False):
    """
    Compute (partially row) pivoted LU decomposition of a matrix.
    The decomposition is:
       P A = L U
    where P is a permutation matrix, L lower triangular with unit
    diagonal elements, and U upper triangular.

    Parameters
    ----------
    A : (n, n) array_like
        Matrix to decompose
    printsteps : switch to print intermediate steps

    Returns
    -------
    lu : (n, n) ndarray
        Matrix containing U in its upper triangle, and L in its
        lower triangle.
        The unit diagonal elements of L are not stored.
    piv : (n,) ndarray
```

```
        Pivot indices representing the permutation matrix P:
        row i of matrix was interchanged with row piv[i].
    """
    m,n = np.shape(A)
    if m != n:
        raise ValueError("expected square matrix")

    # in-place elimination
    lu = A
    # make sure that the data type is 'float'
    lu = lu.astype('float64')
    piv = np.arange(m)
    if printsteps:
        print("input","\n lu =\n",lu, "\n piv=\n",piv,\
              "\n--------------------------\n")
    ### ELIMINATION with partial row pivoting (-> get P A = L U )
    for j in range(min(m,n)-1):

        # find pivot
        k_piv = j + np.argmax(np.abs(lu[j:,j]))

        # only if pivot is nonzero we proceed, otherwise we go to next column
        if lu[k_piv, j] != 0:
            ## ROW SWAP
            lu[[k_piv, j]] = lu[[j, k_piv]]
            # store row swap in piv
            piv[[k_piv, j]] = piv[[j, k_piv]]

            ## ELIMINATION
            for k in range(j+1,n): # rows
                lkj = lu[k,j] / lu[j,j]
                lu[k,j] = lkj
                for i in range(j+1,n): # columns
                    lu[k,i] = lu[k,i] - lkj*lu[j,i]
        if printsteps:
            print(j,"\n lu =\n", np.round(lu,3), "\n\n piv=\n",piv,\
                  "\n--------------------------\n")
    return lu, piv

def lu_solve(lupiv, b, info=False):
    """
    Solve a system A x = b, where A is given as
       P A = L U
    with P being a permutation matrix, L lower triangular with unit
    diagonal elements, and U upper triangular.

    Parameters
    ----------
    lupiv : tuple containing lu and piv computed by lu_factor
    b : (n,) ndarray
        right-hand side
    info : switch whether to print info about existence of solutions

    Returns
    -------
    x : (n,) ndarray or None
        unique solution if exists, otherwise nothing
    """
    lu, piv = lupiv
    lu = np.round(lu, 12)
    m,n = np.shape(lu)
    L = np.eye(m,m) + np.tril(lu[:,:min(m,n)], k=-1)
    U = np.triu(lu[:,:n])

    first0row = -1
    if 0 in list(U.diagonal()):
        # check if any diagonal element is zero (then U is singular)
        # if so, overwrite first0row with the row index of the first zero row
        first0row = list(U.diagonal()).index(0.0)
        if info:
            print("first zero row is (start counting from 0):", first0row)
    if first0row < 0:
        # no zero rows (U is invertible) detetected,
        # since first0row is still negative
        if info:
            print("the system A x = b has an unique solution")
        z = linalg.solve_triangular(L, b[piv], lower = True)
        x = linalg.solve_triangular(U, z, lower = False)
        return x
    else:
        # at least one zero row ==> A is singular
        z = linalg.solve_triangular(L, b[piv], lower = True)
        print("z", z)
        if np.all(lu[first0row:,-1]==0) and np.all(z[first0row:]==0):
            # if all rows including transformed rhs are zero then the system has infinitely
    many sols
            if info:
                print("LinAlg Warning: A is singular, but the system A x = b, has infinitely
    many solutions")
        else:
            # otherwise the system does not admit a solution
            if info:
                print("LinAlg Warning: A is singular and no solution exists")
        return None

if __name__ == "__main__":
    printsteps = 1
    AA = [np.array([[2,  1,  3],
                    [1, -1, -1],
                    [3, -2,  2]]),
         np.array([[1,  2,  2],
                   [2,  0,  1],
                   [3,  2,  3]]),
         np.array([[1,  1,  2],
                   [1, -1,  0],
                   [2,  0,  2]]),
         np.array([[0.5,  -2,  0],
                   [2,8,-2],
                   [1,0,2]])]

    bb = [np.array([-3, 4, 5]),
          np.array([ 1, 3, 4]),
          np.array([ 2, 0, 1]),
          np.array([ -1,10,4])]

    for i, (A,b) in enumerate(zip(AA, bb)):
        print("\n\n--------------------------------\n\
              Example",i+1,\
              "\n--------------------------------")

        print("##### FACTORIZATION #####")
        lu, piv = lu_factor(A, printsteps=printsteps)

        # sparse representation of P^T
        pivT = np.argsort(piv)

        # extract L and U from lu
        m,n = np.shape(A)
        L = np.eye(m,m)+np.tril(lu[:,:min(m,n)], k=-1)
        U = np.triu(lu[:,:n])

        # print tests and results
        print(" A = \n",np.around(A, 3))
        print("\n L = \n",np.around(L, 3))
        print("\n U = \n",np.around(U, 3))
        print("\n piv =",piv,",  pivT =", np.argsort(piv),"\n")
        print(" P A = L U     is ", np.allclose(A[piv],L@U, atol = 1e-6))
        print(" A   = P^T L U is ", np.allclose(A,(L@U)[pivT], atol = 1e-6),"\n")

        print("##### SOLUTION #####")
        x = lu_solve((lu,piv), b, info = True)
        if not np.all(x == None):
            print(" x =\n", np.around(x, 3))
            print("\n A x = b     is ", np.allclose(A@x,b, atol = 1e-6))

        if i<2: # (not that 3. example is not sovable)
            print("\n\n ===== SciPy Factor =====")
            lu, piv = linalg.lu_factor(A)
            L = np.eye(m,m)+np.tril(lu[:,:min(m,n)], k=-1)
            U = np.triu(lu[:,:n])
            print("\n SciPy L = \n",np.around(L, 3))
            print("\n SciPy U = \n",np.around(U, 3))
#            print("\n SciPy piv =", piv," (note that this is LAPACK's piv)\n\n")

    # Another example with a rather large (random) matrix
    from time import time
    n = 10
    A = np.random.rand(n, n)
    b = np.random.rand(n)
    print("\n\n--------------------------------\n\
      Example: Large Random",\
      "\n--------------------------------")

    print("##### FACTORIZATION #####")
    t = time()
    lu, piv = lu_factor(A, printsteps=0)
    print("time our factorization:", time()-t)

    # sparse representation of P^T
    pivT = np.argsort(piv)

    # extract L and U from lu
    m,n = np.shape(A)
    L = np.eye(m,m)+np.tril(lu[:,:min(m,n)], k=-1)
    U = np.triu(lu[:,:n])

    # print tests and results
    print(" P A = L U     is ", np.allclose(A[piv],L@U, atol = 1e-6))
    print(" A   = P^T L U is ", np.allclose(A,(L@U)[pivT], atol = 1e-6),"\n")

    t = time()
    lu, piv = linalg.lu_factor(A)
    print("time SciPy factorization:", time()-t)
```

Ex 63 Linear Systems, Factor and Solve, Python

Solving Linear Systems with Triangular Matrices

1. Implement a function `solve_tri(A, b, lower=False)` which takes as input a triangular matrix A, a vector b and an optional boolean parameter `lower`, which is set to `False` by default. This function shall first check if the dimensions of the input parameters fit and if the matrix is invertible. If these two conditions are true, it shall compute and output the solution x by applying the above derived formulas.

2. Test your routine on some examples with lower and upper triangular matrices. Find the corresponding SciPy Routine and compare.

Solution:

```
import numpy as np
```

```python
def solve_tri(A, b, lower=False):
    """
    Solves a system Ax = b where A is triangular.

    Parameters:
    -----------
    A : triangular matrix as numpy.ndarray of shape (n,n)
    b : right-hand side vector as numpy.ndarray of shape (n,)
    lower : boolean determining if lower or upper triangular

    Returns:
    --------
    x : solution of Ax = b as numpy.ndarray of shape (n,)
    """
    m, n = A.shape
    nb = len(b)
    d = A.diagonal()

    # test dimensions of A and b
    if m != nb:
        raise Exception('dimension of A and b must match. The dimension for A\
                        and b are: {} and {}'.format(np.shape(A), len(b)))

    # test if A is invertible: quadratic?
    elif n != m:
        raise Exception('A is not invertible, its shape is nonquadratic:\
                        {}'.format(np.shape(A)))

    # test if A is invertible: nonzero diagonals?
    elif np.any(d == 0):
        raise Exception('A is not invertible, it has zero diagonal entries')

    # A is invertible:
    else:
        x = np.zeros(n)
        # solve for lower triangular matrix
        if lower:
            for i in range(n):
                for j in range(i):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
                x[i] += 1. / A[i, i] * b[i]

        # solve for upper triangular matrix
        else:
            for i in range(n)[::-1]:
                for j in range(i+1, n):
                    x[i] += 1. / A[i, i] * (-A[i, j] * x[j])
                x[i] += 1. / A[i, i] * b[i]
    return x


if __name__ == "__main__":
    ## Test dimension mismatch
#    A = np.array([[2, 0],
#                  [0, 1]])
#    b = np.array([6])
#    x = solve_tri(A, b, lower = True)

    ## Test nonqudratic A
#    A = np.array([[2, 0, 1],
#                  [0, 1, 1],])
#    b = np.array([6, 2])
#    x = solve_tri(A,b, lower = True)

    ## Test noninvertible A
```

```
#    A = np.array([[2, 0,],
#                  [0, 0],])
#    b = np.array([6,2])
#    x = solve_tri(A,b, lower = True)

    ## Test: ill-conditioned for small delta and large n
    n = 100
    delta = 1.0
    # draw from uniform distribution, shift with -0.5 and strengthen diagonal
    A = np.tril(np.random.rand(n,n)-0.5) + delta * np.eye(n)
    print("det(A) = {}\ncond(A) = {}\nmin(diag(A)) = {}".format(np.linalg.det(A),
                                                              np.linalg.cond(A),
                                                              np.min(abs(A.diagonal())
                                                              )))

    b = np.random.rand(n)
    x = solve_tri(A,b, lower = True)
    print("our test Ax=b:", np.allclose(A.dot(x),b))
    # in SciPy
    from scipy.linalg import solve_triangular
    x = solve_triangular(A, b, lower = True)
    print("scipy test Ax=b:", np.allclose(A.dot(x),b))
    # imshow on the matrix
#    import matplotlib.pyplot as plt
#    plt.imshow(A)
#    plt.show()

    ### Test ill-conditioned diagonal matrix ##
#    n = 10
#    D = np.diag(np.arange(1,n+1))
#    b = np.ones(n)
#    x = solve_tri(D, b, lower = True)
#    print("diag test Ax=b:", np.allclose(D.dot(x),b))
#    # imshow on the matrix
#    import matplotlib.pyplot as plt
#    plt.imshow(D)
#    plt.show()
#    # condition and determinant of the matrix
#    print("condition number of = ", np.linalg.cond(D))
#    print("determinant of A = ", np.linalg.det(D))
```

Ex 64 Linear Systems, Factor and Solve, Python

Solving Linear Systems using QR Decomposition: Factorize and Solve

[Diese Aufgabe nicht mehr stellen, da qr-factor ja schon gemacht und qr-solve exakt solve-triangular ist daher ebenfalls erledigt. daher das besser in die prog aufgabe "curve-fitting mit qr" gesetzt!]

Let $A \in \mathbb{R}^{m\times n}$ be a matrix with $n \leq m$ and linearly independent columns (this implies R is invertible) and let $b \in \mathbb{R}^m$. Then, using a QR decomposition $A = QR$, we can compute the solution x of $Ax = b$ (basically in two steps) by solving

$$Rx = Q^T b.$$

Tasks:

1. Implement a function `factor_qr(A)` which computes a reduced QR decomposition of a matrix $A \in \mathbb{R}^{m\times n}$. Thus, it shall output an orthogonal matrix $Q \in \mathbb{R}^{m\times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n\times n}$, so that $A = QR$.
   You can copy the Gram-Schmidt algorithm implemented as a function `QR(A)` *from previous sheets or find an appropriate SciPy Routine.*

2. Implement a function `solve_qr((Q, R), b)` which takes as input the matrices $Q \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$ computed by `factor_qr(A)` in form of a tuple, as well as a vector $b \in \mathbb{R}^m$. It shall then apply the solving procedure above and output the solution x of $Ax = b$.
   You can recycle the function `solve_tri(A, b, lower=False)` *from previous sheets or use an appropriate SciPy routine for triangular systems.*

3. Test your routine on multiple examples.

Solution:

```python
import numpy as np


def factor_qr(A, own=False):
    """
    computes reduced QR decomposition A=QR of a (mxn) matrix A with m >= n

    INPUT:
        A : numpy.ndarray of shape (m,n), m >= n
        own : switch to use either our or SciPy's routine
    OUTPUT:
        Q : orthogonal matrix Q as numpy.ndarray of shape (m,n)
        R : upper triangular matrix R as numpy.ndarray of shape (n,n)
    """
    m, n = A.shape
    if own:
        # import your own routine to compute reduced QR-decomposition
        # see previous build using Gram-Schmidt Algorithm
        pass
    else:
        import scipy.linalg as linalg
        Q, R = linalg.qr(A)
        # attention: SciPy computes full QR decomposition, i.e.,
        # Q is extended to an orthogonal (mxm) matrix
        # R is extended by zeroes to a (mxn) matrix
        # therefore we need to slice the output to obtain a redcued QR-decomp.

    return Q[0:m,0:n], R[0:n,0:n]

def solve_qr(QR, b, own=False):
    """
    solves a system Ax = b where A = QR
    Assumptions:
        A is expected to have shape (m, n) with m >= n
        the columns of A are indepenent, so that R is invertible
    Remark:
        if A is not square, then linear least square solution is computed

    INPUT:
        QR=(Q,R) : tuple containing
             Q orthogonal matrix Q as numpy.ndarray of shape (m,n)
             R upper triangular matrix R as numpy.ndarray of shape (n,n)
        b : right-hand side vector b as numpy.ndarray of shape (n,)
    OUTPUT:
        x : (least square) solution of Ax = b as numpy.ndarray of shape (n,)
    """
    Q, R = QR
    if own:
        # import your own routine to solve triangular systems here
        pass
    else:
        from scipy.linalg import solve_triangular as solve_tri
        x = solve_tri(R, Q.T @ b )
    return x

if __name__ == "__main__":
    # Test on random data:
    # a few vectors in high-dimensional space are independent with high prob.
    m, n = 1000, 50
    A = np.random.rand(n,n)
    b = np.random.rand(n)

    Q,R = factor_qr(A)
    x = solve_qr((Q,R),b)
    print("Ax = b is", np.allclose(A.dot(x), b, atol = 1e-8))
```

Ex 65

extended qr factor and solve code

Solution:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May 29 15:19:39 2020

@author: vollmann
"""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu May 28 17:16:20 2020

@author: vollmann
"""

import numpy as np
from scipy.linalg import solve_triangular as solve_tri


def qr(A):
    m, n = A.shape
    R = np.zeros((n,n))
    Q = np.zeros((m,n))

    R[0,0] = np.linalg.norm(A[:,0])
    Q[:,0] = A[:,0]/R[0,0]

    for k in range(1,n):
        #print("Q=",Q)
        #print("R=", R)

        for l in range(0, k):
            R[l,k] = A[:,k]@ Q[:,l]

        q = A[:,k] - Q @ R[:,k]
        R[k,k] = np.linalg.norm(q)
        Q[:,k] = q/R[k,k]

    return Q, R


def factor_qr(A):
    # factorization
#    Q,R = np.linalg.qr(A)
    Q,R = qr(A)
    return Q, R

def solve_qr_gen(Q,R,b):

    # solving
    # solve first equation
    b = R.T @ Q.T @ b

    # solve second equation
    A_gram = R.T @ R
    x = ... solve ()

    return x
```

```
A = np.array([[-1,  0,  0, 4, 5],
              [2,  6,  0, 3, 1],
              [1,  3,  -2, 0, 0],
              [1,  1,  1, 0, 1]])
#Q,R = np.linalg.qr(A)
#print(R)
b = np.array([1./3.,2,4,2])
Q,R = factor_qr(A)
x = solve_qr(Q,R,b)
#print(x)
#print(np.allclose(A.dot(x), b, atol = 1e-8))
```

Ex 66

Oblique Projector

For $\alpha \in \mathbb{R}$ consider the matrix

$$P_\alpha = \begin{pmatrix} 1 & \alpha \\ 0 & 0 \end{pmatrix}.$$

1. Show that P_α is a projector for all $\alpha \in \mathbb{R}$.
2. Determine $(I - P_\alpha)$ and show that it is a projector for all $\alpha \in \mathbb{R}$.
3. Find a basis for $\text{Im}(P_\alpha)$ and $\ker(I - P_\alpha)$ as well as $\ker(P_\alpha)$ and $\text{Im}(I - P_\alpha)$.
4. For an arbitrary $y \in \mathbb{R}^2$ determine a vector $v = v(y, \alpha) \in \text{Im}(P_\alpha)$ and a vector $r = r(y, \alpha) \in \ker(P_\alpha)$, so that $y = v + r$. Are v and r unique?
5. Determine the α for which P_α is an orthogonal projector.

Solution:

1. Let $\alpha \in \mathbb{R}$. We find
$$P_\alpha^2 = \begin{pmatrix} 1 & \alpha \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & \alpha \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & \alpha \\ 0 & 0 \end{pmatrix} = P_\alpha.$$
2. Let $\alpha \in \mathbb{R}$. First note
$$I - P_\alpha = \begin{pmatrix} 0 & -\alpha \\ 0 & 1 \end{pmatrix}.$$
Since P_α is a projector, we find $I - P_\alpha$ is a projector, too. We can also verify
$$(I - P_\alpha)^2 = \begin{pmatrix} 0 & -\alpha \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0 & -\alpha \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -\alpha \\ 0 & 1 \end{pmatrix} = I - P_\alpha.$$
3. We know $\text{Im}(P_\alpha) = \ker(I - P_\alpha)$. Since the columns are dependent, we have
$$\text{Im}(P_\alpha) = \text{span}(\begin{pmatrix} 1 \\ 0 \end{pmatrix}).$$
Similarly, we know $\ker(P_\alpha) = \text{Im}(I - P_\alpha)$ and find
$$\text{Im}(I - P_\alpha) = \text{span}(\begin{pmatrix} -\alpha \\ 1 \end{pmatrix}).$$
Remark: We observe that $\ker(P_\alpha) \cap \ker(I - P_\alpha) = \{0\}$.
4. Let $\alpha \in \mathbb{R}$ and $y \in \mathbb{R}^2$. Then from the lecture we know that we can choose
$$v = P_\alpha y = \begin{pmatrix} y_1 + \alpha y_2 \\ 0 \end{pmatrix} \in \text{Im}(P_\alpha)$$
and
$$r = (I - P_\alpha)y = \begin{pmatrix} -\alpha y_2 \\ y_2 \end{pmatrix} \in \ker(P_\alpha).$$
Also we know that r and v are the only vectors in $\text{Im}(P_\alpha)$ and $\ker(P_\alpha)$ respectively, so that $y = v + r$.

5. Choose $\alpha = 0$, then
$$P_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = P_0^\top.$$

Ex 67

Classical Gram–Schmidt Algorithm
Let $A \in \mathbb{R}^{n\times n}$ be defined as
$$A := \begin{pmatrix} 3 & 4 \\ 2 & 7 \end{pmatrix}.$$

1. Compute a QR-decomposition of A using the Gram-Schmidt algorithm.
2. Compute $QR = A$ to check your result.

Solution:

Let a_i denote the i-th column of A.

1. Compute QR-decomposition via Gram-Schmidt:

$$\begin{aligned}
\tilde{q}_1 &:= a_1 = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \\
r_{11} &:= \|\tilde{q}_1\| = \sqrt{13} \\
q_1 &:= \frac{1}{\sqrt{13}} \begin{pmatrix} 3 \\ 2 \end{pmatrix} \\
r_{12} &:= a_2^T q_1 = (4,7) \begin{pmatrix} 3 \\ 2 \end{pmatrix} \frac{1}{\sqrt{13}} = \frac{1}{\sqrt{13}} 26 = 2\sqrt{13} \\
\tilde{q}_2 &:= a_2 - r_{12} q_1 = \begin{pmatrix} 4 \\ 7 \end{pmatrix} - 2\sqrt{13} \frac{1}{\sqrt{13}} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} -2 \\ 3 \end{pmatrix} \\
r_{22} &:= \|\tilde{q}_2\| = \sqrt{4+9} = \sqrt{13} \\
q_2 &:= \frac{1}{\sqrt{13}} \begin{pmatrix} -2 \\ 3 \end{pmatrix} \\
\Rightarrow \quad Q &= \frac{1}{\sqrt{13}} \begin{pmatrix} 3 & -2 \\ 2 & 3 \end{pmatrix} \quad , \quad R = \begin{pmatrix} \sqrt{13} & 2\sqrt{13} \\ 0 & \sqrt{13} \end{pmatrix} = \sqrt{13} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}
\end{aligned}$$

2. Test:
$$QR = \frac{1}{\sqrt{13}} \begin{pmatrix} 3 & -2 \\ 2 & 3 \end{pmatrix} \sqrt{13} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 2 & 7 \end{pmatrix} = A \ \checkmark$$

Ex 68

Consider the matrix
$$A := \begin{pmatrix} 1 & 2 \\ 1 & 0 \end{pmatrix}.$$

1. Compute a QR-decomposition of A using the Gram–Schmidt algorithm.
   (*Hint: Verify the desired properties of the factor matrices and test $QR = A$.*))
2. Is A invertible? Use your QR decomposition to explain your answer.

Solution:

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

1. Compute QR-decomposition via Gram-Schmidt:

$$\begin{aligned}
\tilde{q}_1 &:= A_1 = \begin{pmatrix}1\\1\end{pmatrix}\\
r_{11} &:= \|\tilde{q}_1\| = \sqrt{2}\\
q_1 &:= \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}\\
r_{12} &:= A_2^T q_1 = (2,0)\begin{pmatrix}1\\1\end{pmatrix}\frac{1}{\sqrt{2}} = \sqrt{2}\\
\tilde{q}_2 &:= A_2 - r_{12}q_1 = \begin{pmatrix}2\\0\end{pmatrix} - \sqrt{2}\frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}1\\-1\end{pmatrix}\\
r_{22} &:= \|\tilde{q}_2\| = \sqrt{2}\\
q_2 &:= \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}\\
\Rightarrow \quad Q &= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix} \quad (2P), \quad R = \begin{pmatrix}\sqrt{2} & \sqrt{2}\\0 & \sqrt{2}\end{pmatrix} \quad (2P)
\end{aligned}$$

2. Test:
$$QR = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}\sqrt{2} & \sqrt{2}\\0 & \sqrt{2}\end{pmatrix} = \begin{pmatrix}1 & 2\\1 & 0\end{pmatrix} = A \;\checkmark \quad (2P)$$

3. Yes, because R has nonzero diagonal entries.

Ex 69

QR Factorization
Let $A = \widehat{Q}\widehat{R}$ be a reduced QR factorization of $A \in \mathbb{R}^{m\times n}$. Show that
$$\ker(A) = \{0\} \Leftrightarrow \widehat{R} \in \mathrm{GL}_n(\mathbb{R}).$$

Solution:

Since $\widehat{Q}$ has orthonormal columns, we have $\ker(\widehat{Q}) = \{0\}$. Thus
$$\ker(A) = \ker(\widehat{Q}\widehat{R}) = \ker(\widehat{R}).$$
This implies
$$\ker(A) = \{0\} \Leftrightarrow \ker(\widehat{R}) = \{0\} \Leftrightarrow \widehat{R} \in \mathrm{GL}_n(\mathbb{R}).$$

Ex 70

Solving Linear Systems using the QR Decomposition
Let $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given. Assume you already have the QR decomposition of A, i.e., an orthogonal matrix Q and an upper triangular matrix R, so that $A = QR$. Further assume that R has *nonzero* diagonal elements (i.e., $r_{ii} \neq 0$).

1. How can you use the QR decomposition $A = QR$ for solving a linear system of the form
$$Ax = b \;\; ?$$

2. Use your idea from 1. to solve the system $Ax = b$ where
$$A := \begin{pmatrix}1 & 1 & 0\\1 & 0 & 1\\0 & 1 & 1\end{pmatrix} = QR, \quad \text{with} \quad Q = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}}\\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}}\end{pmatrix}, \; R = \begin{pmatrix}\frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}}\\ 0 & 0 & \frac{2}{\sqrt{3}}\end{pmatrix} \quad \text{and} \quad b := \begin{pmatrix}1\\1\\1\end{pmatrix}.$$

Remark: The square root terms will cancel out nicely while solving for x.

Solution:

1.

$$
\begin{aligned}
Ax = b \quad &\overset{A=QR}{\Leftrightarrow} \quad QRx = b \\
&\overset{\cdot Q^T}{\Leftrightarrow} \quad Rx = Q^T b = \hat{b}
\end{aligned}
$$

Recipe:

$$\boxed{\begin{array}{l} \text{a) Compute} \quad \hat{b} = Q^T b \\ \text{b) Solve} \quad Rx = \hat{b} \end{array}}$$

2. a)

$$\hat{b} = Q^T b = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{2}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{pmatrix}$$

b)

$$\begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{2}{\sqrt{2}} \\ \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}$$

$$
\begin{aligned}
&\text{(III)} \quad \Rightarrow \quad x_3 = \frac{1}{2} \\
&\text{(II)} \quad \frac{3}{\sqrt{6}} x_2 + \frac{1}{\sqrt{6}} x_1 = \frac{2}{\sqrt{6}} \\
&\Rightarrow \quad x_2 = \frac{\sqrt{6}}{3} \left(\frac{2}{\sqrt{6}} - \frac{1}{2\sqrt{6}} \right) = \frac{\sqrt{6}}{\sqrt{6}} \frac{1,5}{3} = \frac{1}{2} \\
&\text{(I)} \quad x_1 \frac{2}{\sqrt{2}} + x_2 \frac{1}{\sqrt{2}} + x_3 \frac{1}{\sqrt{2}} = \frac{2}{\sqrt{2}} \\
&\Rightarrow \quad x_1 = \frac{\sqrt{2}}{2} \left(\frac{2}{\sqrt{2}} - \frac{1}{\sqrt{2}} \right) = \frac{1}{2}
\end{aligned}
$$

$$\Rightarrow \quad x = \frac{1}{2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Ex 71 Linear Systems, Factor and Solve

Solving Linear Systems with Triangular Matrices: Forward/Backward Substitution

1. Solve the following linear system by hand:

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

   Hint: Start with the last equation/row.

2. Let $U = (u_{ij})_{ij} \in \mathbb{R}^{n \times n}$ be an upper triangular matrix, i.e., $u_{ij} = 0$ for $i > j$ and let $b \in \mathbb{R}^n$ be a given vector.
   a) Derive a general (iterative) formula to obtain $x \in \mathbb{R}^n$, which solves $Ux = b$.
   b) What requirements have to be put on the diagonal entries u_{ii} of U, so that the system has a unique solution?

Hint: Write out the system as

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}$$

and start with the last row to determine x_n. Then use x_n to determine x_{n-1} from the second but last row. Continue this procedure until you reach the first row to determine x_1 with the help of the previously determined values $x_2, \ldots, x_n$.

3. Find a similar formula for lower triangular matrices $L = (\ell_{ij})_{ij} \in \mathbb{R}^{n\times n}$, for which $\ell_{ij} = 0$ for $i < j$.

Remark: A diagonal matrix is a special case of a triangular matrix.

Solution:

1.

$$\begin{array}{c} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{array} \begin{pmatrix} 1 & -1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{array}{lllll} \text{(III)} & \Rightarrow & \frac{1}{2}x_3 = 1 & \Rightarrow & x_3 = 2 \\ \text{(II)} & \Rightarrow & 2x_2 + x_3 = 0 & \Rightarrow & x_2 = -1 \\ \text{(I)} & \Rightarrow & x_1 - x_2 + x_3 = 3 & \Rightarrow & x_1 = 0 \end{array}$$

$$\Rightarrow \quad x = \begin{pmatrix} 0 \\ -1 \\ 2 \end{pmatrix}$$

2. First note that a triangular matrix is invertible if and only if the diagonal entries are nonzero (see the formula below or note that $\det(U) = \prod_i u_{ii}$). Now, from considering the i-th row (equation)

$$u_{ii}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{in}x_n = b_i,$$

we obtain a representation for x_i given by

$$x_n = \frac{b_n}{x_n}$$

$$x_i = \underbrace{\frac{1}{u_{ii}}}_{[\text{assume } u_{ii}\neq 0]} \left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j \right), \quad \text{for } i < n,$$

where all the x_j for $j \in \{i+1, \ldots, n\}$ in the sum are computed in previous steps.

3. In the same way, by simply changing the indices in the sum, we find a formula for lower triangular matrices given by

$$x_1 = \frac{b_1}{x_1}$$

$$x_i = \underbrace{\frac{1}{\ell_{ii}}}_{[\text{assume } \ell_{ii}\neq 0]} \left(b_i - \sum_{j=1}^{i-1} \ell_{ij}x_j \right), \quad \text{for } i > 1,$$

where the x_j for $j \in \{1, \ldots, i-1\}$ in the sum are determined in previous steps.

4 Eigenvalues

Ex 72 Linear Algebra, Eigenvalues

```
def r(coeffs):
    coeffs = np.array(coeffs[:-1])
    n = len(coeffs)
    A = np.eye(n, k = -1)
    A[:,-1] = -coeffs
    lam, v = np.linalg.eig(A)
    return lam
```

1. Which algorithm is implemented in the function above? What role does the parameter `coeffs` play?
2. Which value `lam` will the function return at `coeffs = [-1,0,1]`? (No proof needed)
3. What would the function return at `coeffs = [-1,0,-1]`? Can you give a suggestion for improvement, in order to make the function more robust to bad input data?

Solution:

1. "coeffs" are the coefficients of a polynomial and the function r finds the roots of the polynomial.
2. The corresponding polynomial is $x^2 - 1$. Hence lam$= 1, -1$.
3. In case of $[-1, 0, 1]$ we get the same answer. The case of non-normed polynomials should be handled.

Ex 73

Compute the eigenvalues of the following matrices.

1.
$$A = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}$$
2.
$$B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -9 \\ 0 & 1 & 6 \end{pmatrix}$$
3.
$$C = \begin{pmatrix} \pi & 3 & -1 & 6 \\ 0 & 4 & 1 & 7 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Solution:

1. A upper triangular $(1P)$ $(\Rightarrow\ \sigma(A) =$ diag$(A))$
$$\Rightarrow \quad \text{det}(A - \lambda I) = (\pi - \lambda)(4 - \lambda)(1 - \lambda)\left(\frac{1}{2} - \lambda\right)$$
$$\Rightarrow \quad \sigma(A) = \{\pi, 4, 1, \frac{1}{2}\} \quad (2P)$$

2.
$$0 \overset{!}{=} \text{det}(B - \lambda I) = \text{ det}\begin{pmatrix} -\lambda & 0 & 0 \\ 1 & -\lambda & -9 \\ 0 & 1 & 6-\lambda \end{pmatrix} \begin{matrix} -\lambda & 0 \\ 1 & -\lambda \\ 0 & 1 \end{matrix} \quad (1P) \quad \overset{\text{[Sarrus]}}{=} \lambda^2(6 - \lambda) - 9\lambda$$
$$\Leftrightarrow \quad 0 = \lambda(\lambda(6 - \lambda) - 9) = \lambda(-\lambda^2 + 6\lambda - 9) = -\lambda(\lambda - 3)^2 \quad (1P)$$
$$\Leftrightarrow \quad \lambda = 0 \text{ or } \lambda = 3 \ (\sigma(B) = \{0, 3\}) \quad (1P)$$

3. $p(x) = x^3 + \underbrace{(-6)}_{=c_2} x^2 + \underbrace{9}_{=c_1} x + \underbrace{0}_{=c_0}$

companion matrix $= C_p = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -9 \\ 0 & 1 & 6 \end{pmatrix} = B$, see 2. $(3P)$

4.

$$D - \lambda I = \begin{pmatrix} \sqrt{2} - \lambda & \pi \\ 0 & 1 - \lambda \end{pmatrix} \ (1P) \quad \Rightarrow \quad 0 = \det(D - \lambda I) = (\sqrt{2} - \lambda)(1 - \lambda) \ (1P)$$
$$\Rightarrow \quad \sigma(D) = \{\sqrt{2}, 1\} \ (1P)$$

5.

$$E - \lambda I = \begin{pmatrix} -\lambda & 0 & 0 \\ 1 & (1-\lambda) & 3 \\ 2 & 0 & (1-\lambda) \end{pmatrix} \begin{matrix} -\lambda & 0 \\ 1 & (1-\lambda) \\ 2 & 0 \end{matrix} \ (1P) \quad \Rightarrow \quad 0 = \det(E - \lambda I) = \lambda(1-\lambda)^2 \ (1P)$$
$$\Rightarrow \quad \sigma(E) = \{0, 1\} \ (1P)$$

Ex 74

Compute the eigenvalues of the following matrices.

1.

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

2.

$$B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 3 \\ 2 & 0 & 1 \end{pmatrix}$$

3.

$$C = \begin{pmatrix} \pi & 0 & 0 & 0 & 0 \\ 1 & -7 & 0 & 0 & 0 \\ 2 & 0 & i & 0 & 0 \\ 3 & 0 & 8 & 0 & 0 \\ 4 & 7 & 9 & 0 & \frac{1}{2} \end{pmatrix}$$

Solution:

1.

$$A - \lambda I = \begin{pmatrix} \sqrt{2} - \lambda & \pi \\ 0 & 1 - \lambda \end{pmatrix} \ (1P) \quad \Rightarrow \quad 0 = \det(D - \lambda I) = (\sqrt{2} - \lambda)(1 - \lambda) \ (1P)$$
$$\Rightarrow \quad \sigma(A) = \{\sqrt{2}, 1\} \ (2P)$$

—
Alternatively: Argue via triangular matrix (2P)

2.

$$B - \lambda I = \begin{pmatrix} -\lambda & 0 & 0 \\ 1 & (1-\lambda) & 3 \\ 2 & 0 & (1-\lambda) \end{pmatrix} \begin{matrix} -\lambda & 0 \\ 1 & (1-\lambda) \\ 2 & 0 \end{matrix} \ (1P)$$

$$\Rightarrow \quad 0 = \det(E - \lambda I) = -\lambda(1-\lambda)^2 \ (1P)$$
$$\Rightarrow \quad \sigma(B) = \{0, 1\} \ (2P)$$

Ex 75

Compute the eigenvalues of the following matrices.

1.
$$A = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}$$

2.
$$B = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 4 \\ 0 & 1 & 1 \end{pmatrix}$$

3.
$$C = \begin{pmatrix} \pi & 3 & -1 & 2 \\ 0 & 4 & 1 & 7 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Ex 76

Compute the eigenvalues of the following matrices.

1.
$$A = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}$$

2.
$$B = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & -9 \\ 0 & 1 & 6 \end{pmatrix}$$

Ex 77 Linear Algebra, Eigenvalues

Compute Gershgorin Disks
Compute the Gershgorin disks of the following matrices and denote the matrices which necessarily have positive eigenvalues:

$$A = \begin{pmatrix} 14 & 10 & 8 \\ 10 & 14 & 8 \\ 8 & 8 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 10 & 1 & 5 \\ 1 & 17 & 5 \\ 5 & 5 & 25 \end{pmatrix}, \quad C = \begin{pmatrix} 5 & 1 & 3 \\ 1 & -2 & -3 \\ 3 & -3 & -6 \end{pmatrix}.$$

Solution:

- $a_{11} = 14, \ R_1 = 18$
  $a_{22} = 14, \ R_2 = 18$
  $a_{33} = 6, \ R_3 = 16$
  Based on Gershgorin discs we cannot say wether A is positive definite.

- $b_{11} = 10, \ R_1 = 6$
  $b_{22} = 17, \ R_2 = 6$
  $b_{33} = 25, \ R_3 = 10$
  A is positive definite.

- $c_{11} = 5, \ R_1 = 4$
  $c_{22} = -2, \ R_2 = 4$
  $c_{33} = -6, \ R_3 = 9$
  Based on Gershgorin discs we cannot say wether A is positive definite.

Eigendecomposition
Consider the matrix

$$A := \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

1. Why does this matrix possess an eigendecomposition $A = Q\Lambda Q^T$? Compute the matrices Λ and Q, by following this recipe:
   a) Determine its eigenvalues λ_1 and λ_2 to find Λ by solving $\chi_A(\lambda) = \det(A - \lambda I) = 0$.
   b) Determine the corresponding eigenvectors v_1 and v_2 by solving $(A - \lambda_i I)v = 0$.
   c) Normalize the eigenvectors to find Q by setting $\tilde{v}_i := \frac{v_i}{\|v_i\|_2}$ and $Q := [\tilde{v}_1, \tilde{v}_2]$. Test if Q^TQ equals I_2.
   d) Test if $Q\Lambda Q^T$ equals A.
2. Use the eigendecomposition of A to derive its inverse A^{-1}.

Solution:

First note that A is symmetric $(A = A^T)$. Thus, the theorem on eigendecomposition implies the existence of an orthogonal matrix Q and a diagonal matrix Λ, with $A = Q\Lambda Q^T$, which we will now determine.

1. a) Eigenvalues:

$$0 = \det(A - \lambda I) = \det\left(\begin{pmatrix} 2-\lambda & 3 \\ 3 & 2-\lambda \end{pmatrix}\right) = (2-\lambda)^2 - 9$$
$$\Leftrightarrow \quad (2-\lambda)^2 = 9 \quad \Leftrightarrow \quad 2-\lambda = \pm 3 \quad \Leftrightarrow \quad \lambda = 2 \pm 3 \quad (\lambda_1 := 5, \lambda_2 := -1)$$

Thus we set

$$\Lambda := \text{diag}(\lambda_1, \lambda_2) = \begin{pmatrix} 5 & 0 \\ 0 & -1 \end{pmatrix}.$$

b) Eigenvectors:

1) Determine an eigenvector corresponding to $\lambda_1 = 5$:

$$(A - \lambda_1 I)v^1 = 0 \Leftrightarrow \begin{pmatrix} -3 & 3 \\ 3 & -3 \end{pmatrix}\begin{pmatrix} v_1^1 \\ v_2^1 \end{pmatrix} = 0$$
$$\Leftrightarrow -3v_1^1 + 3v_2^1 = 0$$
$$\Leftrightarrow v_1^1 = v_2^1.$$

Choose, e.g., $v^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

2) Determine an eigenvector corresponding to $\lambda_2 = -1$:

$$(A - \lambda_2 I)v^2 = 0 \Leftrightarrow \begin{pmatrix} 3 & 3 \\ 3 & 3 \end{pmatrix}\begin{pmatrix} v_1^2 \\ v_2^2 \end{pmatrix} = 0$$
$$\Leftrightarrow 3v_1^2 + 3v_2^2 = 0$$
$$\Leftrightarrow v_1^2 = -v_2^2.$$

Choose, e.g., $v^2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$.

c) Normalize eigenvectors to define Q:

$$\tilde{v}_1 := \frac{v_1}{\|v_1\|} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tilde{v}_2 := \frac{v_2}{\|v_2\|} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Set

$$Q := [\tilde{v}_1, \tilde{v}_2] = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

We find that Q is orthogonal, more precisely,

$$Q^TQ = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = I.$$

d) Test:

$$Q\Lambda Q^T = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\underbrace{\begin{pmatrix}5 & 0\\ 0 & -1\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}}_{=\frac{1}{\sqrt{2}}\begin{pmatrix}5 & 5\\ -1 & 1\end{pmatrix}}$$

$$= \frac{1}{2}\underbrace{\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}5 & 5\\ -1 & 1\end{pmatrix}}_{=\begin{pmatrix}4 & 6\\ 6 & 4\end{pmatrix}}$$

$$= A \quad (\checkmark)$$

2. The inverse of A is given by

$$A^{-1} = Q\Lambda^{-1}Q^\top = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\underbrace{\begin{pmatrix}\frac{1}{5} & 0\\ 0 & -1\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}}_{=\frac{1}{\sqrt{2}}\begin{pmatrix}\frac{1}{5} & \frac{1}{5}\\ -1 & 1\end{pmatrix}}$$

$$= \frac{1}{2}\underbrace{\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}\frac{1}{5} & \frac{1}{5}\\ -1 & 1\end{pmatrix}}_{=\begin{pmatrix}-\frac{4}{5} & \frac{6}{5}\\ \frac{6}{5} & -\frac{4}{5}\end{pmatrix}}$$

$$= \begin{pmatrix}-0.4 & 0.6\\ 0.6 & -0.4\end{pmatrix}.$$

Test

$$AA^{-1} = \begin{pmatrix}2 & 3\\ 3 & 2\end{pmatrix}\begin{pmatrix}-0.4 & 0.6\\ 0.6 & -0.4\end{pmatrix} = I \quad (\checkmark).$$

Ex 79 Linear Algebra, Eigenvalues

Eigenvalue Characteristics

We denote the identity matrix of $\mathbb{F}^{n\times n}$ by I. Let $A \in \mathbb{F}^{n\times n}$ be a matrix. Some value $\lambda \in \mathbb{C}$ is called eigenvalue of A, if there is a vector $v \neq 0$ in $\mathbb{F}^n$ such that

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

where $0 \in \mathbb{F}^n$ denotes the zero vector. Use the above definition to prove the following assertions.

1. If A is invertible, then for all eigenvalues λ of A we have $\lambda \neq 0$ and $\frac{1}{\lambda}$ is an eigenvalue of A^{-1}.
2. If λ is an eigenvalue of A, then $(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)$ for any $\alpha \in \mathbb{C}$.
3. If λ is an eigenvalue of A, then λ is also an eigenvalue of $Q^\top AQ$ for any orthogonal matrix $Q \in \mathbb{F}^{n\times n}$.

Solution:

Let $A \in \mathbb{F}^{n\times n}$.

1. Let $A \in GL_n(\mathbb{F})$ and $\lambda \in \sigma(A)$ (with eigenvector $v \neq 0$).
   To show, that $\lambda \neq 0$ holds, we assume $\lambda = 0$. $\Rightarrow$ $Av = \lambda v = 0$ $\Rightarrow$ $v = A^{-1}\cdot 0 = 0$. Here we see the contradiction.
   Now we proof $\frac{1}{\lambda} \in \sigma(A^{-1})$:
   $Av \overset{A\in GL_n(\mathbb{F})}{\Leftrightarrow} v = \lambda A^{-1}v \overset{\lambda\neq 0}{\Leftrightarrow} \frac{1}{\lambda}v = A^{-1}v \Leftrightarrow \frac{1}{\lambda} \in \sigma(A^{-1})$ (with the same eigenvector v).
2. Let $\alpha \in \mathbb{C}$.
   $((A - \alpha I) - (\lambda - \alpha)I)v = (A - \lambda I)v \overset{\lambda\in\sigma(A)}{=} 0 \ \checkmark$
   $\Rightarrow$ $(\lambda - \alpha)$ eigenvalue of $(A - \alpha I)$ with the same eigenvector v

3.

$$\begin{aligned} Av = \lambda v \quad &\Leftrightarrow \quad Q^T Av = Q^T \lambda v \\ &\Leftrightarrow \quad Q^T A \underbrace{QQ^T}_{=I} v = \lambda Q^T \underbrace{QQ^T}_{=I} v \\ &\Leftrightarrow \quad Q^T AQ(Q^T v) = \lambda (Q^T v) \\ &\Leftrightarrow \quad \lambda \text{ is eigenvalue of } Q^T AQ \text{ with eigenvector } Q^T v. \end{aligned}$$

Ex 80

Let $A \in \mathbb{F}^{n\times n}$ be invertible and λ some eigenvalue of A.

1. Please show that $\lambda \neq 0$.
2. Please show that $\frac{1}{\lambda}$ is an Eigenvalue of A^{-1}.

Solution:

Let (λ, v) be eigenvalue-eigenvector-pair of A.
$\Leftrightarrow \quad Av = \lambda v \quad \Leftrightarrow \quad BAv = \lambda Bv \quad \Leftrightarrow \quad BA\underbrace{B^{-1}B}_{=I} v = \lambda Bv$
$\Leftrightarrow \quad (\lambda, Bv)$ is eigenvalue-eigenvector-pair of B.

Ex 81

Eigenvalues and Positivity of a Matrix
Definition. A matrix $A \in \mathbb{R}^{n\times n}$ is called positive definite (semi-definite) it $x^T Ax > 0 \quad (\geq 0)$ for all $x \in \mathbb{R}^n \setminus \{0\}$.

Let $A \in \mathbb{R}^{n\times n}$ be any matrix. Please show:

1. If A is positive definite (semi-definite), then $\lambda > 0$ (≥ 0) for all eigenvalues $\lambda \in \sigma(A)$.
   (*Hint:* Rayleigh quotient.)
2. The reverse is not true: Find an example of a matrix, which has positive eigenvalues, but is *not* positive definite.
   (*Hint:* Consider, e.g., a triangular 2×2 matrix.)
3. The reverse is true for symmetric matrices: Let A be *symmetric* and $\lambda > 0$ (≥ 0) for all eigenvalues $\lambda \in \sigma(A)$, then A is positive definite (semi-definite).
   (*Hint:* Eigendecompostion.)
4. Let A be *symmetric*, then A is invertible if and only if $\lambda \neq 0$ for all eigenvalues $\lambda \in \sigma(A)$.
   (*Hint:* Eigendecompostion.)

Solution:

1. Let A be positive definite (semi–definite) and let (λ, v) be an eigenpair of A. Then by using the Rayleigh quotient we find
$$\lambda = \frac{v^\top Av}{v^\top v} = \frac{v^\top Av}{\|v\|_2^2} \quad \frac{>0 \quad (\geq 0)}{>0 \text{ since } v\neq 0} \quad > 0 \quad (\geq 0).$$
2. Take for example
$$A = \begin{pmatrix} 1 & -4 \\ 0 & 2 \end{pmatrix}$$
which has positive eigenvalues $\sigma(A) = \{1, 2\}$, but for $x = (1, 1)^\top$ we have
$$x^\top Ax = -3 + 2 = -1 < 0.$$
3. For symmetric $A \in \mathbb{R}^{n\times n}$ we find an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and diagonal matrix $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ with $\lambda_i \in \sigma(A)$, so that
$$A = Q\Lambda Q^T \quad \text{(eigendecomposition)}.$$

Since Q is orthogonal, any vector $x \in \mathbb{R}^n$ can be written as $x = Ix = QQ^\top x = Q\mu$, with coordinates $\mu := Q^\top x$ where $\mu \neq 0$ for $x \neq 0$. Therefore

$$x^\top Ax = \mu^\top Q^\top Q\Lambda Q^\top Q\mu = \mu^\top \Lambda\mu = \sum_{i=1}^{n} \mu_i^2 \lambda_i \geq 0.$$

We have "$>$" if $x \neq 0$.

4. For symmetric $A \in \mathbb{R}^{n\times n}$ we find an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and diagonal matrix $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ with $\lambda_i \in \sigma(A)$, so that
$$A = Q\Lambda Q^T \quad \text{(eigendecomposition)}.$$
Therefore:
$$A \text{ invertible} \overset{(*)}{\Leftrightarrow} \Lambda \text{ invertible} \overset{(**)}{\Leftrightarrow} \lambda \neq 0 \;\; \forall \lambda \in \sigma(A).$$
$(*)$ product of matrices is invertible if all factors are invertible and orthogonal matrices are invertible with inverse $Q^T = Q^{-1}$

$(**)$ diagonal matrices are invertible, if and only if diagonal elements are nonzero

Example: A symmetric and positive definite (spd) $\Rightarrow$ A invertible.

Ex 82

Eigenvalues and Positivity of a Matrix

Definition. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called positive definite (semi-definite) if $x^T Ax > 0$ (≥ 0) for all $x \in \mathbb{R}^n \setminus \{0\}$.

Let $A \in \mathbb{R}^{n\times n}$ be any matrix. Please show:

1. Find an example of a nonsymmetric matrix, which has strictly positive eigenvalues, but $x^T Ax < 0$ for some $x \in \mathbb{R}^n$. (*Hint:* Consider, e.g., a triangular 2×2 matrix.)
2. Let A be *symmetric* and $\lambda > 0$ (≥ 0) for all eigenvalues $\lambda \in \sigma(A)$, then A is positive definite (semi-definite).
3. Show that symmetric and positive definite matrices are invertible.

Solution:

1. Take for example
$$A = \begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}$$
which has positive eigenvalues $\sigma(A) = \{1, 1\}$. For $x = (1, 1)^\top$ we have
$$x^\top Ax = 1 + 1 + \alpha < 0 \quad \forall \alpha < -2.$$
2. For symmetric $A \in \mathbb{R}^{n\times n}$ we find an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and diagonal matrix $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ with $\lambda_i \in \sigma(A)$, so that
$$A = Q\Lambda Q^T \quad \text{(eigendecomposition)}.$$
Let $x \in \mathbb{R}^n$ and define $\mu := Q^\top x$ where $\mu \neq 0$ for $x \neq 0$. Then
$$x^\top Ax = \mu^\top \Lambda\mu = \sum_{i=1}^{n} \mu_i^2 \lambda_i \geq 0.$$
We have "$>$" if $x \neq 0$.
3. For $A = Q\Lambda Q^T$ (eigendecomposition) we have
$$A \text{ invertible} \overset{(*)}{\Leftrightarrow} \Lambda \text{ invertible} \overset{(**)}{\Leftrightarrow} \lambda \neq 0 \;\; \forall \lambda \in \sigma(A).$$
$(*)$ product of matrices is invertible if all factors are invertible and orthogonal matrices are invertible with inverse $Q^T = Q^{-1}$

($**$) diagonal matrices are invertible, if and only if diagonal elements are nonzero; symmetric and positive definite matrices have positive eigenvalues

---

Ex 83

Properties of Eigenvalues
Prove the following statements (see also the corresponding Lemma from the lecture):

1. *The eigenvalues of the powers of a matrix:*
   Let $A \in \mathbb{F}^{n\times n}$, $\lambda \in \sigma(A)$ then $\lambda^k \in \sigma(A^k)$ for any $k \in \mathbb{N}$.
2. *Eigenvalues of invertible Matrices:*
   Let $A \in \mathbb{F}^{n\times n}$ be invertible and $\lambda \in \sigma(A)$, then $\lambda \neq 0$ and $\frac{1}{\lambda}$ is an eigenvalue of A^{-1}.
3. *Eigenvalues of a scaled matrix:*
   Let $A \in \mathbb{F}^{n\times n}$ and $\lambda \in \sigma(A)$, then $\alpha\lambda \in \sigma(\alpha A)$ for any $\alpha \in \mathbb{F}$.
4. *Real symmetric matrices have real eigenvalues:* (Not obvious without using properties of complex numbers! Solution not relevant for exam and thus not discussed here.)
   $A \in \mathbb{R}^{n\times n}$, $A = A^T \quad \Rightarrow \quad \sigma(A) \subset \mathbb{R}$.
5. *The eigenvalues of real orthogonal matrices:* (Not obvious without using properties of complex numbers! Solution not relevant for exam and thus not discussed here.)
   $Q \in \mathbb{R}^{n\times n}$ be orthogonal, $\lambda = a + ib \in \sigma(Q) \quad \Rightarrow \quad |\lambda| := \sqrt{a^2+b^2} = 1$
6. *The eigenvalues of an upper (or lower) triangular matrix are sitting on its diagonal:*
   Let $U \in \mathbb{F}^{n\times n}$ with $u_{ij} = 0$ for $i > j$. Then $\sigma(U) = \{u_{11}, \dots, u_{nn}\}$.
7. *Similar matrices have the same eigenvalues:*
   Let $A \in \mathbb{F}^{n\times n}$ and $T \in GL_n(\mathbb{F})$, i.e., T is invertible. Then $\sigma(A) = \sigma(T^{-1}AT)$.
8. *Eigenvalues of a shifted matrix:*
   Let $A \in \mathbb{F}^{n\times n}$ and $\lambda \in \sigma(A)$, then $(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)$ for any $\alpha \in \mathbb{F}$.
9. *Symmetric matrices have orthogonal eigenvectors:* (Solution not relevant for exam and thus not discussed here.)
   Let $\lambda_1 \neq \lambda_2$ be two *distinct* eigenvalues of a real *symmetric* matrix $A \in \mathbb{R}^{n\times n}$ (i.e., $A = A^T$), and let $v_1, v_2 \in \mathbb{R}^n$ be corresponding eigenvectors. Proof that v_1 and v_2 are orthogonal, i.e., $v_1^\top v_2 = 0$.

Solution:

1. ***The eigenvalues of the powers of a matrix:***
   For an eigenpair (λ, v) of A we find
   $$A^k v = A^{k-1}Av = A^{k-1}\lambda v.$$
   Iterating k times gives the desired result.
2. ***Eigenvalues of invertible Matrices:***
   Let $A \in GL_n(\mathbb{F})$ and $\lambda \in \sigma(A)$ with eigenvector $v \neq 0$.
   First, we show that $\lambda \neq 0$ holds. For this purpose let us assume $\lambda = 0$. Then $Av = \lambda v = 0$ implies $v = A^{-1} \cdot 0 = 0$, which contradicts that v is an eigenvector (and therefore nonzero).
   Second, we proof $\frac{1}{\lambda} \in \sigma(A^{-1})$. We find
   $$Av = \lambda v \quad \overset{A\in GL_n(\mathbb{F})}{\Leftrightarrow} \quad v = \lambda A^{-1}v \quad \overset{\lambda\neq 0}{\Leftrightarrow} \quad \frac{1}{\lambda}v = A^{-1}v \quad \Leftrightarrow \quad \frac{1}{\lambda} \in \sigma(A^{-1}) \quad \text{(with the same eigenvector } v\text{)}.$$
3. ***Eigenvalues of a scaled matrix:***
   Let $\alpha \in \mathbb{F}$, then for an eigenpair (λ, v) of A we find
   $$Av = \lambda v \Leftrightarrow (\alpha A)v = (\alpha\lambda)v$$
   implying that $(\alpha\lambda, v)$ is an eigenpair of αA.
4. ***Real symmetric matrices have real eigenvalues:***
   We first collect some observations:

- In general, for $x, y \in \mathbb{F}^n$ and a matrix $A \in \mathbb{F}^{n\times n}$ we easily find by the definition of the matrix product
$$x^\top Ay = \sum_{i,j} a_{ij}x_iy_j.$$
If the matrix is symmetric, i.e., $a_{ij} = a_{ji}$, we further find
$$x^\top Ay = \sum_i a_{ii}x_iy_i + \sum_{i\neq j} a_{ij}(x_iy_j + x_jy_i). \tag{1}$$
- For $z = x + iy \in \mathbb{C}$ let us define $\overline{z} := x - iy$ (the so-called *complex conjugate* of z). Then we easily find
  i) $z\overline{z} = a^2 + b^2 \in \mathbb{R}$ (real number!),
  ii) for $w = c + id$ we find $\overline{z}w + z\overline{w} = 2(ac + bd) \in \mathbb{R}$ (real number!) and also $\overline{z \cdot w} = \overline{z} \cdot \overline{w}$.

Now to the task: Let (λ, v) be an eigenpair of $A = A^\top \in \mathbb{R}^{n\times n}$, i.e.,
$$Av = \lambda v.$$
Multiplying $\overline{v} := (\overline{v}_1, \dots, \overline{v}_n)$ from the left yields
$$\overline{v}^\top Av = \lambda\overline{v}^\top v.$$
Now, if we can show that $\overline{v}^\top Av$ and $\overline{v}^\top v$ are real, then we know that λ is real. First, we observe that
$$\overline{v}^\top v = \sum_{i=1}^n \overline{v}_iv_i,$$
which is real, since all summands $\overline{v}_iv_i$ are real by i) above. Secondly, since A is **symmetric** we can apply (**??**) to obtain
$$\overline{v}^\top Av = \sum_i a_{ii}\overline{v}_iv_i + \sum_{i\neq j} a_{ij}(\overline{v}_iv_j + \overline{v}_jv_i),$$
which is real, since all summands are real by i) and ii) above and the assumption that the matrix only has **real** coefficients.

Remarks:

- **We cannot relax symmetry assumption:** Matrices with just real coefficients can have complex eigenvalues. Take for example the (orthogonal) matrix
$$A = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$
for which $\det(A - \lambda I) = \lambda^2 + 1$, so that $\sigma(A) = \{i, -i\}$ with eigenvectors $\{\begin{pmatrix} 1 \\ -i \end{pmatrix}, \begin{pmatrix} 1 \\ i \end{pmatrix}\}$. However, the additional property of symmetry is a sufficient condition for a real matrix A to have solely real eigenvalues!
- **We cannot relax assumption of real coefficients:** A symmetric matrix with complex coefficients can have complex eigenvalues. Consider for example
$$A = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix},$$
for which $\det(A - \lambda I) = \lambda^2 + 1$, so that $\sigma(A) = \{i, -i\}$.
- **The general result for complex matrices:** Like *symmetry* for real matrices we introduce for complex matrices: A matrix $A \in \mathbb{C}^{n\times n}$ is called *Hermitian* or *self-adjoint* if $A^\top = \overline{A}$. With other words, Hermitian matrices are invariant under transposition *and* (additionally) conjugation. With the same proof as above one can show that such matrices also have real eigenvalues. For example, consider the Hermitian matrix
$$A = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix},$$
for which $\det(A - \lambda I) = \lambda^2 - 1$, so that $\sigma(A) = \{1, -1\}$.

5. ***The eigenvalues of real orthogonal matrices:***
We know $Q^\top Q = I$. Now let (λ, v) be an eigenpair of Q, i.e., $Qv = \lambda v$. Using the notation from the previous subtask, i.e., letting $\overline{v} = (\overline{v}_1, \dots, \overline{v}_n)$ denote the complex conjugate of v, we find on the one hand that
$$(Q\overline{v})^\top (Qv) = \overline{v}^\top Q^\top Qv = \overline{v}^\top v \in \mathbb{R}.$$

On the other hand, since Q is assumed to be real we find

$$(Q\overline{v})^\top(Qv) = (\overline{Qv})^\top(Qv) = (\overline{\lambda v})^\top(\lambda v) = \overline{\lambda}\lambda(\overline{v}^\top v) = |\lambda|^2(\overline{v}^\top v) \in \mathbb{R}.$$

Thus combining these two equations gives

$$|\lambda| = 1.$$

6. ***The eigenvalues of an upper (or lower) triangular matrix are sitting on its diagonal:***
Recall:
1) The determinant of an (upper) triangular matrix is given by the product of its diagonal entries

$$\det(U) = u_{11} \cdot u_{22} \cdot \ldots \cdot u_{nn}.$$

2) The eigenvalues λ of a matrix $A \in \mathbb{R}^{n\times n}$ are the roots of the characteristic polynomial

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

Since $U - \lambda I = \begin{pmatrix} u_{11}-\lambda & & * \\ & \ddots & \\ 0 & & u_{nn}-\lambda \end{pmatrix}$ is also upper triangular we find

$$\det(U - \lambda I) = (u_{11} - \lambda) \cdot (u_{22} - \lambda) \cdot \ldots \cdot (u_{nn} - \lambda) \overset{!}{=} 0 \Leftrightarrow \lambda \in \{u_{11}, \ldots, u_{nn}\}.$$

Analogue proof for lower triangular matrices.

7. ***Similar matrices have the same eigenvalues:***
Let (λ, v) be an eigenpair of A, then since T is invertible we find

$$\underbrace{Av = \lambda v}_{\text{by definition of}(\lambda,v)} \overset{T^{-1}\cdot|}{\Leftrightarrow} \underbrace{T^{-1}Av}_{=T^{-1}AIv} = T^{-1}(\lambda v) \overset{I=TT^{-1}}{\Leftrightarrow} T^{-1}AT(T^{-1}v) = \lambda(T^{-1}v).$$

Thus, $(\lambda, T^{-1}v)$ is an eigenpair for the matrix $T^{-1}AT$ (note that $T^{-1}v \neq 0$, since $v \neq 0$).

Remark: We call two matrices A and B *similar* if there exists an invertible matrix T such that $B = T^{-1}AT$.

8. ***Eigenvalues of a shifted matrix:***
Let $\alpha \in \mathbb{F}$ and (λ, v) be an eigenpair of A. Then $((A - \alpha I) - (\lambda - \alpha)I)v = (A - \lambda I)v \overset{\lambda\in\sigma(A)}{=} 0$
Thus $(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)$ with the same eigenvector v.

9. ***Symmetric matrices have orthogonal eigenvectors:***
Remark upfront: Since we are considering a real symmetric matrix, we know that the eigenvalues are real. However, one may still find complex eigenvectors. For example consider the identity matrix I with spectrum $\sigma(I) = \{1\}$. Then obviously any nonzero vector v (complex or not) is an eigenvector to the eigenvalue 1. We now show that real eigenvalues enable us to choose real eigenvectors (as implicitly assumed in the task). To clarify this, let (λ, v) be an eigenpair, where $\lambda \in \mathbb{R}$ but v may potentially have complex coefficients. Let us split v according to the real and imaginary parts of its coefficient, more precisely

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} x_1 + iy_1 \\ \vdots \\ x_n + iy_n \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} + i \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} =: x + iy.$$

Then we find

$$Av = \lambda v \Leftrightarrow A(x + iy) = \lambda(x + iy) \Leftrightarrow Ax + iAy = \lambda x + i\lambda y.$$

Note here that λ is real and thus we have the splitting into real λx and imaginary part λy. By comparing real parts in the last equation we obtain

$$Ax = \lambda x,$$

so that the real part of v is also an eigenvector to the eigenvalue λ.

Now to the task: Let A be symmetric, i.e., $A = A^T$ and let (λ_1, v_1) and (λ_2, v_2) be two (real) eigenpairs of A with $\lambda_1 \neq \lambda_2$, then

$$v_1^T \underbrace{Av_2}_{=\lambda_2 v_2} = \lambda_2 v_1^T v_2,$$

and also

$$v_1^T \underbrace{A}_{=A^T} v_2 = \underbrace{v_1^T A^T v_2}_{=(v_1^T A^T v_2)^T} = v_2^T \underbrace{Av_1}_{=\lambda_1 v_1} = \lambda_1 v_2^T v_1 = \lambda_1 v_1^T v_2.$$

Now we subtract these terms and find

$$0 = v_1^T A v_2 - v_1^T A v_2 = \lambda_2 v_1^T v_2 - \lambda_1 v_1^T v_2 = \underbrace{\lambda_2 - \lambda_1}_{\neq 0, \text{ since } \lambda_2 \neq \lambda_1} v_1^T v_2$$

$$\overset{\lambda_1 \neq \lambda_2}{\Rightarrow} v_1^T v_2 = 0.$$

Ex 84

Properties of Eigenvalues
Prove the following statements (see also the corresponding Lemma from the lecture):

1. *The eigenvalues of the powers of a matrix:*
   Let $A \in \mathbb{F}^{n \times n}$, $\lambda \in \sigma(A)$ then $\lambda^k \in \sigma(A^k)$ for any $k \in \mathbb{N}$.
2. *Eigenvalues of invertible Matrices:*
   Let $A \in \mathbb{F}^{n \times n}$ be invertible and $\lambda \in \sigma(A)$, then $\lambda \neq 0$ and $\frac{1}{\lambda}$ is an eigenvalue of A^{-1}.
3. *Eigenvalues of a scaled matrix:*
   Let $A \in \mathbb{F}^{n \times n}$ and $\lambda \in \sigma(A)$, then $\alpha\lambda \in \sigma(\alpha A)$ for any $\alpha \in \mathbb{F}$.
4. *The eigenvalues of an upper (or lower) triangular matrix are sitting on its diagonal:*
   Let $U \in \mathbb{F}^{n \times n}$ with $u_{ij} = 0$ for $i > j$. Then $\sigma(U) = \{u_{11}, \dots, u_{nn}\}$.
5. *Eigenvalues of a shifted matrix:*
   Let $A \in \mathbb{F}^{n \times n}$ and $\lambda \in \sigma(A)$, then $(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)$ for any $\alpha \in \mathbb{F}$.
6. *Hermitian matrices have orthogonal eigenvectors:*
   Let $\lambda_1 \neq \lambda_2$ be two *distinct* eigenvalues of a *Hermitian* matrix $A \in \mathbb{C}^{n \times n}$ (i.e., $A = A^H$), and let $v_1, v_2 \in \mathbb{C}^n$ be corresponding eigenvectors. Proof that v_1 and v_2 are orthogonal, i.e., $v_1^H v_2 = 0$.

Solution:

1. ***The eigenvalues of the powers of a matrix:***
   For an eigenpair (λ, v) of A we find
   $$A^k v = A^{k-1} A v = A^{k-1} \lambda v.$$
   Iterating k times gives the desired result.
2. ***Eigenvalues of invertible Matrices:***
   Let $A \in GL_n(\mathbb{F})$ and $\lambda \in \sigma(A)$ with eigenvector $v \neq 0$.
   First, we show that $\lambda \neq 0$ holds. For this purpose let us assume $\lambda = 0$. Then $Av = \lambda v = 0$ implies $v = A^{-1} \cdot 0 = 0$, which contradicts that v is an eigenvector (and therefore nonzero).
   Second, we proof $\frac{1}{\lambda} \in \sigma(A^{-1})$. We find
   $$Av = \lambda v \quad \overset{A \in GL_n(\mathbb{F})}{\Leftrightarrow} \quad v = \lambda A^{-1} v \quad \overset{\lambda \neq 0}{\Leftrightarrow} \quad \frac{1}{\lambda} v = A^{-1} v \quad \Leftrightarrow \quad \frac{1}{\lambda} \in \sigma(A^{-1}) \quad \text{(with the same eigenvector } v\text{).}$$
3. ***Eigenvalues of a scaled matrix:***
   Let $\alpha \in \mathbb{F}$, then for an eigenpair (λ, v) of A we find
   $$Av = \lambda v \Leftrightarrow (\alpha A) v = (\alpha \lambda) v$$
   implying that $(\alpha\lambda, v)$ is an eigenpair of αA.

4. ***The eigenvalues of an upper (or lower) triangular matrix are sitting on its diagonal:***
   Recall:
   1) The determinant of an (upper) triangular matrix is given by the product of its diagonal entries
$$\det(U) = u_{11} \cdot u_{22} \cdot \ldots \cdot u_{nn}.$$
   2) The eigenvalues λ of a matrix $A \in \mathbb{R}^{n\times n}$ are the roots of the characteristic polynomial
$$\det(A - \lambda I) = 0.$$
   Since $U - \lambda I = \begin{pmatrix} u_{11}-\lambda & & * \\ & \ddots & \\ 0 & & u_{nn}-\lambda \end{pmatrix}$ is also upper triangular we find
$$\det(U - \lambda I) = (u_{11}-\lambda)\cdot(u_{22}-\lambda)\cdot\ldots\cdot(u_{nn}-\lambda) \overset{!}{=} 0 \Leftrightarrow \lambda \in \{u_{11},\ldots,u_{nn}\}.$$
   Analogue proof for lower triangular matrices.

5. ***Eigenvalues of a shifted matrix:***
   Let $\alpha \in \mathbb{F}$ and (λ, v) be an eigenpair of A. Then $((A-\alpha I) - (\lambda - \alpha)I)v = (A - \lambda I)v \overset{\lambda\in\sigma(A)}{=} 0$
   Thus $(\lambda - \alpha)$ is an eigenvalue of $(A - \alpha I)$ with the same eigenvector v.

6. ***Hermitian matrices have orthogonal eigenvectors:***
   Let A be hermitian, i.e., $A = A^H$ and let (λ_1, v_1) and (λ_2, v_2) be two eigenpairs of A with $\lambda_1 \neq \lambda_2$ (both real), then
$$v_1^H \underbrace{Av_2}_{=\lambda_2 v_2} = \lambda_2 v_1^H v_2,$$
   and also
$$v_1^H \underbrace{A}_{=A^H} v_2 = v_1^H A^H v_2 = \underbrace{(Av_1)^H}_{=(\lambda_1 v_1)^H} v_2 = \overline{\lambda_1} v_1^H v_2 = \lambda_1 v_1^H v_2.$$
   Now we subtract these terms and find
$$0 = v_1^H A v_2 - v_1^H A v_2 = \lambda_2 v_1^H v_2 - \lambda_1 v_1^H v_2 = \underbrace{(\lambda_2 - \lambda_1)}_{\neq 0,\text{ since } \lambda_2\neq\lambda_1} v_1^H v_2$$
$$\overset{\lambda_1\neq\lambda_2}{\Rightarrow} v_1^H v_2 = 0.$$

Ex 85

Let
$$Q := \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

1. What can you say about the columns of Q? Justify your answer rigorously.
2. Compute the determinant of Q and the eigenvalues of Q.
3. Interpret the function $\mathbb{R}^3 \to \mathbb{R}^3, x \mapsto Qx$ geometrically.

Solution:

1.
   - orthonormal columns $\Rightarrow$ Q is orthogonal, i.e., $Q^TQ = QQ^T = I$
   - Q is invertible, $Q^{-1} = Q^T$
2.
   - Determinant (by Sarrus Rule): $\det(Q) \overset{[Sarrus]}{=} 0+0+0-0-0-(-1) = 1$
     (Recall: For any orthogonal matrix Q we have $|\det(Q)| = 1$)

- Similarly we find the eigenvalues:

$$0 \overset{!}{=} \det(q - \lambda I) \overset{(Sarrus)}{=} (-\lambda)\cdot(-\lambda)\cdot(1-\lambda) + (1-\lambda) = (1-\lambda)(\lambda^2+1)$$
$$\Leftrightarrow (1-\lambda) = 0 \text{ or } (\lambda^2+1) = 0 \Leftrightarrow \lambda = 1 \text{ or } \lambda = \pm i$$

  Thus: The spectrum of Q is given by $\sigma(Q) = \{1, i, -i\}$

3. Q is a rotation of 90^o around the x_3-axis:

$$Q = Q_{\alpha=\frac{\pi}{2}} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$Q\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -x_2 \\ x_1 \\ x_3 \end{pmatrix}$$

We can see, that Q switches x_1 and x_2 with a sign change for x_2 and x_3 remains the same.

Ex 86

Please show that the eigenvalues of an upper triangular matrix $A \in \mathbb{R}^{n\times n}$ are given on its diagonal.

Solution:

Show: $A \in \mathbb{R}^{n\times n}$ upper triangular $\Rightarrow \quad \sigma(A) = \{a_{11}, \dots, a_{nn}\}$
Proof:

$$\lambda \in \sigma(A) \quad \Leftrightarrow \quad \det(A - \lambda I) = 0 \quad (1P)$$
$$\Leftrightarrow \quad \prod_{i=1}^{n}(a_{ii} - \lambda) = 0 \quad (1P)$$
$$\left[\uparrow \text{ since } A - \lambda I =: U \text{ is upper triangular and } \det(U) = \prod_{i=1}^{n} u_{ii}\right] \quad (1P)$$

Now we show $\det(U) = \prod_{i=1}^{n} u_{ii}$:

$$\det(U) = u_{11} \cdot \underbrace{\det(U_1)}_{=u_{22}\cdot\det(U_2)}$$
$$(3P) \quad \vdots$$
$$= \prod_{i=1}^{n} u_{ii}$$

□

Ex 87

Eigenvalues Triangular Matrix
Use Theorem 12.2iii) to show that the eigenvalues of an upper triangular matrix $A \in \mathbb{R}^{n\times n}$ are given on its diagonal, i.e., that $\det(A - \lambda I) = 0$ for all $\lambda \in diag(A)$.

Solution:

Show: $\det(A - \lambda I) = 0 \quad \forall \lambda \in \text{diag}(A) = \{a_{11}, \dots, a_{nn}\}$
Proof:
Since $(A - \lambda I)$ is itself upper triangular it remains to show, that

$$\det(U) = \prod_{i=1}^{n} u_{ii}$$

for any upper triangular matrix.(Because then: $\det(A-\lambda I)=\prod_{i=1}^n (a_i i-\lambda)$.)
Consider:

$$U=\begin{pmatrix} u_{11} & & & \\ 0 & u_{22} & & \\ \vdots & 0 & \ddots & \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

Then by T.12.2: $\det(U)=\det([u_{11}])\det\underbrace{\begin{pmatrix} u_{22} & & \\ 0 & \ddots & \\ 0 & 0 & u_{nn} \end{pmatrix}}_{=:U_{11}}$

We can now apply the same idea to

$$U_{11}=\begin{pmatrix} u_{22} & & & \\ 0 & u_{33} & & \\ \vdots & 0 & \ddots & \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

$$\begin{aligned} \det(U) &= \underbrace{\det([u_{11}])}_{=u_{11}}\underbrace{\det([u_{22}])}_{=u_{22}}\det\begin{pmatrix} u_{33} & & \\ 0 & \ddots & \\ 0 & 0 & u_{nn} \end{pmatrix} \\ &\vdots \\ &= \prod_{i=1}^n u_{ii} \end{aligned}$$

Ex 88 Linear Algebra, Eigenvalues

Gershgorin Disks
Let $A\in\mathbb{C}^{n\times n}$ be a matrix with entries a_{ij} for $i,j=1,\dots,n$. Let $R_i:=\sum_{j\neq i}|a_{ij}|$ be the sum of the absolute values of the non-diagonal entries in the i-th row. Moreover, let

$$D(a_{ii},R_i):=\{z\in\mathbb{C}\,|\,\|z-a_{ii}\|\leq R_i\}$$

be the disk of radius R_i and centre a_{ii} in $\mathbb{C}$. Prove the following theorem.

Theorem: Every eigenvalue of A lies within at least one of the Gershgorin disks $D(a_{ii},R_i)$, i.e., $\forall\ \lambda\in\sigma(A)\ \exists\ i\in\{1,\dots,n\}:\ \ \lambda\in D(a_{ii},R_i)$.

Solution:

We show, that $\forall\ \lambda\in\sigma(A)\ \exists\ i\in\{1,\dots,n\}:\ \ \lambda\in D_i$

- Let $\lambda\in\sigma(A)$, then choose an eigenvector $v=\begin{pmatrix} v_1\\ \vdots \\ v_n\end{pmatrix}$, so that
  $\exists\ i\in\{1,\dots,n\}:\ v_i=1$ and $|v_j|\leq 1\ \ \forall i\neq j$
  $\Big($choose any eigenvector $\tilde{v}$ and then set $v:=\frac{\tilde{v}}{\tilde{v}_i}$, $\|\tilde{v}\|_\infty=|\tilde{v}_i|\Big)$.
- We know: $Av=\lambda v$, in particular (componentwise)

$$\begin{aligned} &\underbrace{(Av)_i}_{=\sum_{j=1}^n a_{ij}v_j=a_{ii}\cdot 1+\sum_{j\neq i}a_{ij}v_j} = (\lambda v)_i=\lambda\underbrace{v_i}_{=1}=\lambda \\ \Leftrightarrow\quad & |\lambda-a_{ii}|=\left|\sum_{j\neq i}a_{ij}v_j\right| \overset{\text{[triangle inequality]}}{\leq} \sum_{j\neq i}|a_{ij}|\underbrace{|v_j|}_{\leq 1}\leq R_i \\ \Leftrightarrow\quad & \lambda\in D_i \end{aligned}$$

□

Ex 89 Linear Algebra, Eigenvalues

Inverse Power Method
Let $A \in \mathbb{R}^{n\times n}$ be a matrix with eigenvalues λ_j for $j \in \{1, \dots, n\}$ with the property

$$|\lambda_n| \geq \cdots \geq |\lambda_{i+1}| > |\lambda_i| > |\lambda_{i-1}| \geq \cdots \geq |\lambda_1|.$$

Let $\hat\lambda \approx \lambda_i$ be a *good guess* for the eigenvalue λ_i, which means

$$0 < |\lambda_i - \hat\lambda| < |\lambda_j - \hat\lambda| \ \ \forall\, i \neq j. \tag{1}$$

Or in other words, $\hat\lambda$ is closer to λ_i (in absolute terms) than any other eigenvalue.
Let us define

$$B := (A - \hat\lambda I).$$

1. Show that $(\lambda_i - \hat\lambda)^{-1}$ is (in magnitude) the largest eigenvalue of the matrix B^{-1}.
2. Let $x^0 \in \mathbb{R}^n$. With respect to A, the iteration

$$x^{k+1} := \frac{B^{-1}x^k}{\|B^{-1}x^k\|} \tag{2}$$

for $k \geq 0$ is called *inverse power iteration*. Under which condition on x^0 does this iteration converge and what is the limit?

Hint: Apply the theorem about the power method from the lecture.

Solution:

The message here is: Under the assumption of having a "good" guess for λ_i, we find the "exact" eigenvector (and -value) with the help of the *inverse power iteration*. However note that it is computationally more expensive, because you have to solve a linear system in each iteration.

We have $A \in \mathbb{R}^{n\times n}$ with eigenvalues $|\lambda_n| \geq \cdots \geq |\lambda_{i+1}| > |\lambda_i| > |\lambda_{i-1}| \geq \cdots \geq |\lambda_1|$ and corresponding eigenvectors $v_j \neq 0$. Also we have $\hat\lambda \approx \lambda_i$ so that

$$0 < |\lambda_i - \hat\lambda| < |\lambda_j - \hat\lambda| \ \ \forall\, i \neq j\ (*).$$

Now define $B := (A - \hat\lambda I)$.

1. We show, that $(\lambda_i - \hat\lambda)^{-1}$ is the largest eigenvalue of B^{-1}.
   - (i) For all j we have that $(\lambda_j - \hat\lambda)$ is an eigenvalue of B to the same eigenvector v_j (previous sheets).
   - (ii) Also B is invertible, otherwise there would exist $v \neq 0$, so that $Bv = 0$, which would imply $\hat\lambda$ eigenvalue of A, which would contradict the assumption: $|\lambda_j - \hat\lambda| > 0\ \forall j \ \Rightarrow\ |\lambda_j - \hat\lambda| \neq 0\ \forall j$. Thus $\frac{1}{\lambda_j - \hat\lambda}$ is an eigenvalue of B^{-1} for all j to the same eigenvector v_j (see previous sheets).
     By $(*)$ we then find that
     $$\frac{1}{|\lambda_i - \hat\lambda|} > \frac{1}{|\lambda_j - \hat\lambda|}\ \forall j\ \ (\sharp).$$
2. Let v_1 be an eigenvector to the eigenvalue $(\lambda_i - \hat\lambda)^{-1}$, and x^0 an orthogonal initial guess, i.e., $(x^0, v_1) \neq 0$, then due to $(\sharp)$ we find by the theorem about the power method that

$$\frac{B^{-1}x^k}{\|B^{-1}x^k\|} \to v_1.$$

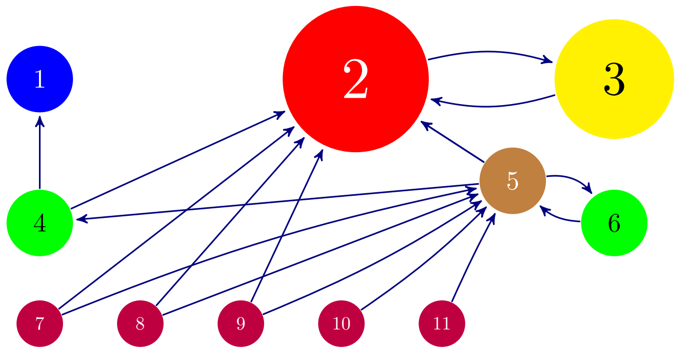
Note: v_1 is eigenvector of
   - ...A to the eigenvalue λ_i.
   - ...$B = A - \hat\lambda_i I$ to the eigenvalue $\lambda_i - \hat\lambda$.
   - ...$B^{-1} = (A - \hat\lambda_i I)^{-1}$ to the eigenvalue $(\lambda_i - \hat\lambda)^{-1}$.

An Application of Eigenvectors: The *PageRank*
Aim: To rank results of a web search engine (such as Google) according to the *"importance"* of the web pages.

1998: For this purpose, Larry Page and Sergei Brin developed the PageRank algorithm as the basis of the Google empire.

Assumption: *"important"* means more links from other (important) web pages.

Idea: Let us think of the web as a directed graph, i.e., web pages are nodes and links from one page to another, i.e, from one node to another, are modeled as directed edges. For example a web structure consisting of 11 web pages could look as follows:



We now randomly place a random surfer according to the initial probability distribution $x^0 = (x_1^0, \dots, x_n^0)$ on this graph. Here, n (in the example above $n = 11$) denotes the number of web pages and x_i^0 denotes the probability that the random surfer *starts* at web page i. Further let $e := (1, \dots, 1)^T$, then the fact that x^0 is a probability distribution (i.e., probabilities sum up to 1) translates into $e^T x^0 = x_1^0 + \ldots + x_n^0 = 1$. Now we make the assumption that the random surfer moves...

(1) ...with probability $\alpha \in (0, 1)$ according to the link structure (with equal preferences to outgoing links)

(2) ...with probability $(1 - \alpha)$ he can teleport to a random page (with equal probability) to prevent stranding in deadlocks

$\rightarrow$ Pages, where the random surfer is more likely to appear in the long run based on the web's structure are considered more important.

These two movements can be described by multiplying the current probability distribution with the two following matrices:

$$P_1 = \left(\begin{array}{cccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1 & 1 & & & 1/2 & & & & & & & \\ 2 & & & 1 & 1/2 & 1/3 & & 1/2 & 1/2 & 1/2 & & \\ 3 & & 1 & & & & & & & & & \\ 4 & & & & & 1/3 & & & & & & \\ 5 & & & & & & 1 & 1/2 & 1/2 & 1/2 & 1 & 1 \\ 6 & & & & & 1/3 & & & & & & \\ 7 & & & & & & & & & & & \\ 8 & & & & & & & & & & & \\ 9 & & & & & & & & & & & \\ 10 & & & & & & & & & & & \\ 11 & & & & & & & & & & & \end{array}\right), \quad P_2 := \frac{1}{n} e e^T = \left(\frac{1}{n}\right)_{ij}$$

More precisely,

(1) **Link structure:** P_1 is the probability matrix (column stochastic) defined by
$P_1^{ij} :=$ Probability that random surfer moves from page j to page i defined by the link structure

(2) **Jumps:** P_2 is the probability matrix (column stochastic) defined by
$P_2^{ij} := \frac{1}{n}=$ Probability that random surfer jumps from page j to page i

The movement of the random surfer is then completely defined by the probability matrix

$$P = \alpha P_1 + (1 - \alpha) P_2.$$

This matrix is also known as the ***Google Matrix***. For the next time instances we therefore obtain

$$x^1 = \alpha P_1 x^0 + (1-\alpha) P_2 x^0 = P x^0$$
$$x^2 = \alpha P_1 x^1 + (1-\alpha) P_2 x^1 = P x^1$$
$$x^{k+1} = \alpha P_1 x^k + (1-\alpha) P_2 x^k = P x^k = P^{k+1} x^0$$
$$x^* = \lim_{k\to\infty} x^k =: \textbf{\textit{PageRank}}$$

Observations:

- One can easily show that P_1, P_2 and thus P are column stochastic (i.e., $e^T P = e^T$)
- Consequently, since x^0 is a probability distribution (i.e., $e^T x^0 = 1$), also $e^T x^k = 1$ for all k and $e^T x^* = 1$

Question:
Does this sequence $\{x^k\}_{k\in\mathbb{N}}$ of vectors converge (to a steady state)? More precisely, is there a $x^* = \lim_{k\to\infty} x^k = \lim_{k\to\infty} P^k x^0$, so that

$$P x^* = 1 x^*. \tag{1}$$

→ With other words, is there an ***eigenvector*** x^* to the ***eigenvalue*** 1 of the matrix P?

→ ***Eigenvalue algorithms*** are developed to solve such problems. One of them is the ***Power iteration***, which, applied to the eigenvalue problem above, produces precisely the sequence

$$x^k = P^k x^0.$$

Intuitively, in the limit most of the "mass" would be located at web pages that have many incoming links and would therefore be ranked as being more important. In fact, the i-th component of x^* is called the *PageRank* of the web page i.

Remark: *Perron Theorem*
A positive damping factor $\alpha > 0$ is also technically necessary as it assures that the matrix P has only strictly positive coefficients. The Perron Theorem then sates that its largest eigenvalue is strictly larger than all other eigenvalues (in magnitude). Thus the convergence of the Power method is guaranteed. Since the matrix is column stochastic one can further show that the largest eigenvalue is 1.

Solution:

Ex 90 Linear Algebra, Eigenvalues, Python

The Power Iteration and the *PageRank*

1. Implement a function `power_iteration(A,m,p=1)` which takes as input a matrix $A \in \mathbb{R}^{n\times n}$, a maximum iteration number $m \in \mathbb{N}$ and an *optional* parameter p which determines the order of the p-norm and is set to $p = 1$ by default. This function shall then return the m-th iterates x_m and μ_m of the power iteration.

   Hints:

   - You can use a random distribution x_0 as initial guess by calling for example the numpy function

     ```
     x = numpy.random.dirichlet(np.ones(n),size=1).reshape(n)
     ```

     or simply choose

     ```
     x = 1./n * np.ones(n).
     ```

   - For the normalization step use the numpy function

     ```
     numpy.linalg.norm(x, ord=p),
     ```

     which allows the choices $p \in \{1, 2, \infty\}$ (among others).

2. Determine the **PageRank** of the web structure given above. Therefore apply your function `power_iteration(A,m,p=1)` to the PageRank matrix

$$A := P = \alpha P_1 + (1-\alpha) P_2,$$

where

$$P_1 = \left(\begin{array}{cccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1 & 1 & & & 1/2 & & & & & & & \\ 2 & & & 1 & 1/2 & 1/3 & & 1/2 & 1/2 & 1/2 & & \\ 3 & & 1 & & & & & & & & & \\ 4 & & & & & 1/3 & & & & & & \\ 5 & & & & & & 1 & 1/2 & 1/2 & 1/2 & 1 & 1 \\ 6 & & & & & 1/3 & & & & & & \\ 7 & & & & & & & & & & & \\ 8 & & & & & & & & & & & \\ 9 & & & & & & & & & & & \\ 10 & & & & & & & & & & & \\ 11 & & & & & & & & & & & \end{array}\right), \quad P_2 := \frac{1}{n}ee^T = \left(\frac{1}{n}\right)_{ij}.$$

Play around with the damping factor α. What do you observe?

Hint: For implementing P_1 and P_2, and thus P, you can use this code snippet.

```
import numpy as np
def P(alpha):
    P_1=np.array([[1,0,0,1.0/2,0,0,0,0,0,0,0],
                  [0,0,1.0,1.0/2,1.0/3,0,1.0/2,1.0/2,1.0/2,0,0],
                  [0,1.0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,1.0/3,0,0,0,0,0,0],
                  [0,0,0,0,0,1.0,1.0/2,1.0/2,1.0/2,1.0,1.0],
                  [0,0,0,0,1.0/3,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0]])

    P_2 = (1.0 / 11.0) * np.ones((11,11))

    return alpha * P_1 + (1-alpha) * P_2
```

Solution:

```
import numpy as np

def power_iteration(A, m, p=1):
    """
    Solves eigenvalue problem via Power Method
    Expects the largerst eigenvalue of A to be scritly larger

    Parameters
    ----------
    A : (n,n) ndarray
        matrix
    m : int
        number of iterations
    p : int or numpy.inf, optional
        specifying the order of the p-Norm used for normalization

    Returns
    -------
    x : (n,1) ndarray
        normalized (with p-Norm) eigenvector for largest eigenvalue
    mu : float
         largest eigenvalue
    """
    n = A.shape[1]
    x = 1./n * np.ones(n)
    for k in range(m):
        z = A.dot(x)
        x = z / np.linalg.norm(z, ord=p)
        mu = x.dot(z) / (x.dot(x))
    return x, mu


def P(alpha):
    P_1 = np.array([[1, 0, 0, 1.0/2, 0, 0, 0, 0, 0, 0, 0],
                    [0,0,1.0,1.0/2,1.0/3,0,1.0/2,1.0/2,1.0/2,0,0],
                    [0,1.0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,1.0/3,0,0,0,0,0,0],
                    [0,0,0,0,0,1.0,1.0/2,1.0/2,1.0/2,1.0,1.0],
                    [0,0,0,0,1.0/3,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0]])

    P_2 = (1.0 / 11.0) * np.ones((11, 11))

    return alpha * P_1 + (1-alpha) * P_2


if __name__ == "__main__":
    m = 20
    k = 10
    # run with different damping factors
    for alpha in np.linspace(0, 1, k, endpoint=True):
        print("\n-----------------\nalpha =",
              np.round(alpha, 4),
              "\n------------------")
        eigvec, mu_max = power_iteration(P(alpha), m, p=1)
        print("Maximal eigenvalue = ", np.around(mu_max, 10))
        print("\nEigenvector = \n", np.around(eigvec, 5))

        lbdmax_np = np.max(np.linalg.eigvals(P(alpha)))
        print("\nMaximal eigenvalue from numpy = ",
              np.around(np.real(lbdmax_np), 10))
```

Ex 91 Linear Algebra, Eigenvalues, Python

The QR Algorithm

1. Implement the QR eigenvalue algorithm as a function `eig(A,m)`. The function shall take as input a matrix $A \in \mathbb{R}^{n\times n}$ and a maximum iteration number $m \in \mathbb{N}$. It shall return the diagonal of the last iterate A_m. For the QR decomposition you can use the Gram-Schmidt algorithm from previous sheets or an appropriate Python function.

2. Test your algorithm on a random matrix $A \in \mathbb{R}^{n\times n}$. In order to generate such a random matrix use the following code snippet:

   ```
   def A_gen(n):
      from numpy as np
      from scipy.linalg import qr
      A = np.random.rand(n,n)
      Q, R = qr(A)
      Lambda = np.diag(np.arange(1,n+1))
      A = Q @ (Lambda @ Q.T)
   return A
   ```

3. Find a routine in Scipy to compute the eigenvalues and -vectors of a matrix. Test the routine on multiple examples, especially for higher dimensions. Compare to your algorithm.

Solution:

```
import numpy as np
```

```python
def qr_factor(A):
    """
    Computes a QR-decomposition of a (mxn)-matrix with m>=n via Gram-Schmidt.

    Parameters
    ----------
    A : (mxn) matrix with m>=n

    Returns
    -------
    Q : (mxn) with orthonormal columns
    R : (nxn) upper triangular matrix
    """
    m, n = A.shape
    R = np.zeros((n, n))
    Q = np.zeros((m, n))

    R[0, 0] = np.linalg.norm(A[:, 0])
    Q[:, 0] = A[:, 0] / R[0, 0]

    for k in range(1, n):
        for l in range(0, k):
            R[l, k] = A[:, k] @ Q[:, l]
        q = A[:, k] - Q @ R[:, k]
        R[k, k] = np.linalg.norm(q)
        Q[:, k] = q / R[k, k]

    return Q, R


def eig(A, m=50, qr="own"):
    """
    Computes the eigenvalues of a square matrix via QR eigenvalue algorithm

    Parameters
    ----------
    A : (nxn) matrix with *distinct* eigenvalues
    m : iteration number
    qr : optional parameter to switch between own qr and scipy qr

    Returns
    -------
    d : diagonal of the last QR-iterate
    """
    if qr == "own":
        qr = qr_factor
    else:
        pass

    for k in range(m):
        Q, R = qr(A)
        A = R @ Q
    return A.diagonal()


def A_gen(n):
    import numpy as np
    from scipy.linalg import qr
    A = np.random.rand(n, n)
    Q, R = qr(A)
    eigvals = - np.linspace(0, 1, n)  # np.arange(1, n+1)
    Lambda = np.diag(eigvals)
    A = Q @ Lambda @ Q.T
    return A
```

```
if __name__ == "__main__":

    # 2 test
    n = 10
    qr = "own"
    A = A_gen(n)
    for m in [10, 50, 75, 150]:
        print("number of iterations:\n m =", m, "\napproximate eigenvalues:\n",
              np.sort(np.round(eig(A, m, qr=qr), 6)), "\n")
    # 3 compare to numpy.linalg
    print("--> compare to numpy.linalg:\n", np.sort(np.linalg.eigvals(A)))
```

Ex 92 Linear Algebra, Determinant

Rule of Sarrus

Derive the *Rule of Sarrus* for the determinant of a (3×3)-matrix by using the Laplace formula from the lecture with $n = 3$. Then compute the determinant of

$$A = \begin{pmatrix} 1 & 4 & 3 \\ 0 & 2 & 2 \\ 0 & 1 & 1 \end{pmatrix}.$$

What does it tell us about the columns?

Solution:

Recall:

$$\begin{aligned} \det(A) &= \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(A_{ij}) \quad (i \in \{1, \dots, n\},\ \text{fixed}) \\ \det(a) &:= a \end{aligned}$$

Now consider $n = 3$ and let us fix $i = 1$. We indicate the submatrices A_{ij} by colors:

$$j = 1: \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$j = 2: \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$j = 3: \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

By Laplace formula we obtain

$$\begin{aligned} \det(A) &= \underbrace{(-1)^{1+1}}_{=1} a_{11} \det(A_{11}) + \underbrace{(-1)^{1+2}}_{=-1} a_{12} \det(A_{12}) + \underbrace{(-1)^{1+3}}_{=1} a_{13} \det(A_{13}) \\ &= a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{31}a_{23}) + a_{12}(a_{21}a_{32} - a_{31}a_{22}) \\ &= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{31}a_{23} + a_{13}a_{21}a_{32} - a_{13}a_{31}a_{22}. \end{aligned}$$

Note that we have exploited also the Laplace formula for 2×2 matrices. For the example matrix this yields $\det(A) = 2 - 2 = 0$, so that we can conclude that the columns are linearly dependent.

5 Singular Values and SVD

Ex 93 Linear Systems, Factor and Solve

Frobenius Matrices
Let $\ell_j := (0, \dots, 0, \ell_{j+1,j}, \dots, \ell_{m,j})^\top \in \mathbb{R}^m$, $e_j \in \mathbb{R}^m$ be the j-th unit vector and $I \in \mathbb{R}^{m\times m}$ be the identity matrix. Then show that the matrix

$$L_j := I + \ell_j e_j^\top \in \mathbb{R}^{m\times m}$$

satisfies:

1. The matrix L_j is an invertible lower triangular matrix.
2. The inverse of L_j is given by $L_j^{-1} = I - \ell_j e_j^\top \in \mathbb{R}^{m\times m}$.
3. For $i \le j$ it holds that $L_i L_j = I + \ell_j e_j^\top + \ell_i e_i^\top$ and $L_i^{-1} L_j^{-1} = I - \ell_j e_j^\top - \ell_i e_i^\top$.

Solution:

1. First note that $\ell_j e_j^\top$ is a lower triangular matrix with zeroes on its diagonal because $\ell_{i,j} = 0$ for $i \le j$. Therefore L_j is a lower triangular matrix with ones on its diagonal and thus invertible (note, e.g., that $\det(L_j) = 1 \neq 0$).
2. Since the inverse matrix is unique it is sufficient to show that $L_j(I - \ell_j e_j^\top) = I$. By inserting the definition we find that

$$\begin{aligned} L_j(I - \ell_j e_j^\top) &= (I + \ell_j e_j^\top)(I - \ell_j e_j^\top) \\ &= I + \ell_j e_j^\top - \ell_j e_j^\top - \ell_j e_j^\top \ell_j e_j^\top \\ &= I - \ell_j (e_j^\top \ell_j) e_j^\top \\ &= I, \end{aligned}$$

   where we have exploited $e_j^\top \ell_j = 0$ which follows from $\ell_{j,j} = 0$.
3. We insert definitions and compute the products. First,

$$\begin{aligned} L_i L_j &= (I + \ell_i e_i^\top)(I + \ell_j e_j^\top) \\ &= I + \ell_i e_i^\top + \ell_j e_j^\top + \ell_i e_i^\top \ell_j e_j^\top \\ &= I + \ell_i e_i^\top + \ell_j e_j^\top + \ell_i (e_i^\top \ell_j) e_j^\top \\ &= I + \ell_i e_i^\top + \ell_j e_j^\top, \end{aligned}$$

   where we have exploited $e_i^\top \ell_j = 0$, which follows from $\ell_{i,j} = 0$ for all $i \le j$. The second statement follows along the same lines.

Ex 94 Linear Algebra, Singular Values

Frobenius Norm, Trace and Singular Values
The Frobenius norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined by

$$\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$$

and the trace of a matrix $B \in \mathbb{R}^{n\times n}$ by

$$\mathrm{tr}(B) = \sum_{i=1}^{n} b_{ii}.$$

1. Show that for a matrix $A \in \mathbb{R}^{m\times n}$ we have

$$\|A\|_F^2 = \mathrm{tr}(A^T A).$$

2. Show that the trace is symmetric, i.e., for $A, B \in \mathbb{R}^{m\times n}$ we find
$$\text{tr}(A^\top B) = \text{tr}(B^\top A) = \text{tr}(BA^\top).$$
Remark: $\mathbb{R}^{m\times n} \times \mathbb{R}^{m\times n} \to \mathbb{R}, (A,B) \mapsto \text{tr}(B^\top A)$ defines an inner product on $\mathbb{R}^{m\times n}$ and the Frobenius norm $\|A\|_F = \sqrt{\text{tr}(A^T A)}$ denotes the corresponding norm. For example, Cauchy-Schwarz inequality holds.

3. Use these results and the singular value decomposition to show that the Frobenius norm of a matrix $A \in \mathbb{R}^{m\times n}$ can be expressed in terms of its singular values, i.e.,
$$\|A\|_F^2 = \sum_{i=1}^{\min(m,n)} \sigma_i^2 \quad \left(= \sum_{i=1}^{r} \sigma_i^2 = \|\Sigma\|_F^2 \right).$$

Solution:

1. First note that by definition of the matrix product we have
$$(A^\top A)_{jj} = \sum_{i=1}^m a'_{ji} a_{ij} = \sum_{i=1}^m a_{ij} a_{ij} = \sum_{i=1}^m a_{ij}^2.$$
Therefore, by definition of the trace and the Frobenius norm, we have
$$\text{tr}(A^\top A) = \sum_{j=1}^n (A^\top A)_{jj} = \sum_{j=1}^n \sum_{i=1}^m a_{ij}^2 = \|A\|_F^2.$$

2. By definition of the trace and the matrix product and exploiting the commutativity of the product in $\mathbb{R}$, we find
$$\begin{aligned}
\text{tr}(A^\top B) &= \sum_{i=1}^n (A^\top B)_{ii} \\
&= \sum_{i=1}^n \sum_{k=1}^m a'_{ik} b_{ki} = \left(\sum_{i=1}^n \sum_{k=1}^m b'_{ik} a_{ki} = \sum_{i=1}^n (B^\top A)_{ii} = \text{tr}(B^\top A) \right) \\
&= \sum_{i=1}^n \sum_{k=1}^m b_{ki} a'_{ik} \\
&= \sum_{k=1}^m \sum_{i=1}^n b_{ki} a'_{ik} \left(= \sum_{i=1}^n (BA^\top)_{kk} = \text{tr}(BA^\top) \right)
\end{aligned}$$

3. Let $A = U\Sigma V^\top$, then
$$\begin{aligned}
\|A\|_F^2 = \text{tr}(A^\top A) &\overset{SVD}{=} \text{tr}((U\Sigma V^\top)^\top U\Sigma V^\top) \overset{\text{reverse prod. of transpose}}{=} \text{tr}(V\Sigma^\top U^\top U \Sigma V^\top) \\
&\overset{\text{U ortho.}}{=} \text{tr}(V\Sigma^\top \Sigma V^\top) \overset{\text{symmetry of tr}}{=} \text{tr}(\Sigma V^\top V \Sigma^\top) \\
&\overset{\text{V ortho.}}{=} \text{tr}(\Sigma\Sigma^\top) \overset{\text{Def. tr}}{=} \sum_{i=1}^n (\Sigma\Sigma^\top)_{ii} \\
&\overset{\text{Def. mat. prod.}}{=} \sum_{i=1}^m \sum_{j=1}^n \Sigma_{ij}\Sigma^\top_{ji} = \sum_{i=1}^m \sum_{j=1}^n \Sigma_{ij}^2 \overset{\Sigma \in \mathbb{R}^{m\times n} \text{ diag.}}{=} \sum_{i=1}^{\min(m,n)} \Sigma_{ii}^2.
\end{aligned}$$

Ex 95 Linear Algebra, Singular Values

An ill-conditioned Diagonal Matrix
For $n \in \mathbb{N}$ consider the diagonal matrix
$$D_n = \text{diag}\left(1, \frac{1}{2}, \dots, \frac{1}{n}\right) = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{n} \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Tasks:

1. Is D_n invertible? Explain your answer.
2. For a given $b \in \mathbb{R}^n$, determine the corresponding $x_b \in \mathbb{R}^n$, such that $D_n x_b = b$. Is x_b uniquely defined? Explain your answer.
3. Determine the spectrum $\sigma(D_n)$ of D_n.
4. Find a singular value decomposition of D_n.
5. What is the condition number $\text{cond}_2(D_n)$ of D_n? Determine $\lim_{n\to\infty} \text{cond}_2(D_n)$.
6. Let us assume $b \in \mathbb{R}^n$ is the true right-hand side and $\tilde{b}$ would be our measured right-hand side, which is prone to some error. For simplicity let us assume $\tilde{b} = b + \varepsilon e$ for some fixed small $\varepsilon > 0$ and $e = (1, \ldots, 1)^T \in \mathbb{R}^n$ (i.e., each component of b is equally perturbed by ε). Consider the difference $\Delta x := x_b - x_{\tilde{b}}$ and estimate the relative error $\frac{\|\Delta x\|}{\|x\|}$. What happens for large n?

Solution:

1. Yes, because diagonal entries are nonzero (then, e.g., $\det(D_n) \neq 0$).
2. We find
$$x_b = (b_1, 2b_2, \ldots, nb_n),$$
which is uniquely determined because D_n is invertible.
3. We find
$$0 = \det(D_n - \lambda I) = \prod_{i=1}^{n} (d_{ii} - \lambda) \quad \Leftrightarrow \quad \lambda \in \{1, \frac{1}{2}, \ldots, \frac{1}{n}\}.$$
4. Set $V := U := I_n$ (orthogonal) and $\Sigma := D_n$ (diagonal with positive entries), then obviously
$$D_n = U \Sigma V^T.$$
5. We find
$$\text{cond}_2(D_n) = \frac{\sigma_{max}}{\sigma_{min}} = \frac{1}{\frac{1}{n}} = n \quad \to \infty \text{ (as } n \to \infty).$$
6. From the lecture
$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}_2(D_n) \frac{\|\Delta b\|}{\|b\|} = \frac{\varepsilon n \sqrt{n}}{\|b\|}.$$
Thus for fixed b and ε, the relative error can get arbitrarily large as n increases.

Some Background

A column in A contains the measured features (e.g., age and height) for a particular sample (e.g., a person) and a row contains all measured values for a particular feature. Thus, let $a_i \in \mathbb{R}^n$ denote a row of A, then by assuming that the mean $a_i^\top \mathbf{1}$ is zero (without loss of generality, otherwise center the data) for all features, we have

$$\frac{1}{n-1} a_i^\top a_j = \begin{cases} \text{“statistical variance in feature } i\text{”} & i = j \\ \text{“statistical covariance between feature } i \text{ and } j\text{”} & i \neq j. \end{cases}$$

Furthermore we observe that the matrix
$$\frac{1}{n-1} AA^\top = \frac{1}{n-1} \left(a_i^\top a_j\right)_{ij}$$
contains these covariances (it is therefore often called *sample covariance matrix*) and using the SVD $A = U\Sigma V^\top$ we find

$$\frac{1}{n-1} AA^T = \frac{1}{n-1} U \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_r^2 \end{pmatrix} U^T = \frac{1}{n-1} \sum_{i=1}^{r} \sigma_i^2 u_i u_i^T.$$

The first few summands explain most of $\frac{1}{n-1}AA^T$, i.e., the sample covariance matrix, and the singular vectors $u_1, \dots, u_r$ are called principal components.

Geometrically we have the following interpretation:

$$A = \begin{array}{cc} m \text{ feats} & \overset{n \text{ samples}}{\longrightarrow} \\ \downarrow & \begin{pmatrix} | & & | & & | \\ a_1 & \cdots & a_i & \cdots & a_n \\ | & & | & & | \end{pmatrix} \end{array} = U\Sigma V^T = \underbrace{\begin{pmatrix} | & & | \\ u_1 & \cdots & u_m \\ | & & | \end{pmatrix}}_{\text{orthonormal basis}} \underbrace{(\Sigma V^T)}_{\text{coordinates of } a_i \text{ in terms of this basis}}$$

Thus, each sample $a_i \in \mathbb{R}^m$ is a linear combination of $u_1, \dots, u_m$ with coefficients $(\Sigma V^T)_i$.

Relation to "total least squares": The least squares problem

$$\min_{x \in \mathbb{R}} \|Ax - b\|^2,$$

where we want to minimize the error in the *dependent* variables, can be reformulated as the constrained optimization problem

$$\min_{r,x \in \mathbb{R}} \|r\|^2$$
$$s.t. \quad r = Ax - b.$$

If we also want to encounter errors in the *independent* variables we arrive at the problem

$$\min_{r,s,x \in \mathbb{R}} \| \begin{pmatrix} r \\ s \end{pmatrix} \|^2$$
$$s.t. \quad (A + s)x = b + r.$$

This problem is called the total *least squares problem* and errors on both dependent and independent variables are considered. One can show that the solution of this problem is the low-rank approximation which we yield from cropping the singular value decomposition. See for details: `https://eprints.soton.ac.uk/263855/1/tls_overview.pdf`

Solution:

Ex 96 Least Squares Problems

Minimum Norm Least Squares with Pseudoinverse

Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Consider the SVD $A = U\Sigma V^\top$ and set $A^+ = V\Sigma^+U^\top$, where $\Sigma^+ = \text{diag}(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0)$ is the pseudoinverse of a diagonal matrix as derived in the lecture. Show that

$$x^+ := A^+b = \arg \min_{x \in \{x:\ A^\top Ax = A^\top b\}} \|x\|_2^2,$$

i.e., x^+ is the minimum norm least squares solution.
Hint: First consider the simple case that A is diagonal and then use the SVD for the general case.

Solution:

(1) Special Case: Diagonal matrix

Let us start with the simple case: $A \in \mathbb{R}^{m\times n}$ diagonal

$$A = \begin{pmatrix} a_{11} & & & & 0 \\ & \ddots & & & \\ & & a_{rr} & & \\ & & & 0 & \\ & & & & \ddots \\ 0 & & & & 0 \end{pmatrix} \in \mathbb{R}^{m\times n}, \quad a_{ii} \neq 0, \quad A^TA = \begin{pmatrix} a_{11}^2 & & & & 0 \\ & \ddots & & & \\ & & a_{rr}^2 & & \\ & & & 0 & \\ & & & & \ddots \\ 0 & & & & 0 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

Normal equation:

$$A^TAx = A^Tb = \begin{pmatrix} a_{11}b_1 \\ \vdots \\ a_{rr}b_r \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \Leftrightarrow \quad \begin{matrix} a_{11}^2 x_1 = a_{11}b_1 \\ \vdots \\ a_{rr}^2 x_r = a_{rr}b_r \\ 0 \cdot x_i = 0 \\ (i > r) \end{matrix} \quad \Leftarrow \quad \begin{matrix} x_1 = \frac{b_1}{a_{11}} \\ \vdots \\ x_r = \frac{b_r}{a_{rr}} \\ x_{r+1} = 0 \\ \vdots \\ x_n = 0 \end{matrix}$$

$$\Rightarrow \quad x^+ = \begin{pmatrix} \frac{1}{a_{11}}b_1 \\ \vdots \\ \frac{1}{a_{rr}}b_r \\ 0 \\ \vdots \end{pmatrix} 0 = A^+b, \quad A^+ = \begin{pmatrix} \frac{1}{a_{11}} & & & & 0 \\ & \ddots & & & \\ & & \frac{1}{a_{rr}} & & \\ & & & 0 & \\ & & & & \ddots \\ 0 & & & & 0 \end{pmatrix} \in \mathbb{R}^{n\times m}$$

Note: The x_i for $i > r$ can be chosen arbitrarily, but setting them to zero gives the smallest vector.

(2) General Case

By using the SVD $A = U\Sigma V^\top$ we find

$$A^TA = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T\Sigma V^T,$$

so that the normal equation reads as

$$(*) \quad A^TAx = A^Tb \;\Leftrightarrow\; V\Sigma^T\Sigma(V^Tx) = V\Sigma^T(U^Tb)$$

$$\overset{V^T\cdot|}{\Leftrightarrow} \quad \underbrace{\Sigma^T\Sigma(V^Tx) = \Sigma^T(U^Tb)}_{\text{(normal equation for}(\Sigma,\ U^Tb))} \quad (\sharp)$$

Consequently, x solves $(*)$ if and only if $y := V^Tx$ solves $(\sharp)$. Since V is orthogonal both solutions have the same norm, more precisely,

$$\|x\|_2^2 = x^\top x = x^\top(V^\top V)x = \|Vx\|_2^2 = \|y\|_2^2.$$

For diagonal matrices we have shown that $y^+ = \Sigma^+U^Tb$ is the smallest solution of $(\sharp)$. Thus, $x^+ := Vy^+ = \Sigma^+U^Tb$ is the smallest solution of $(*)$, i.e., the minimum norm least squares solution.

All in all: Since orthogonal matrices (here U and V) are not only invertible but also isometric and the SVD $A = U\Sigma V^\top$ always exists, we could rely on the result for diagonal matrices (here Σ).

Ex 97 Linear Algebra, Singular Values, Python

Computation of the SVD

1. Assume you can only solve eigenvalue problems. Implement a Python function `U,V,sigma = svd(A)`, which computes a reduced SVD of some matrix $A \in \mathbb{R}^{m\times n}$ by using `numpy.linalg.eigh(A)`.
2. Test your routine on some examples.

Solution:

```
import numpy as np
import scipy.linalg as linalg
import matplotlib.pyplot as plt


def svd(A):
    """
    Compute reduced SVD of A by using linalg.eigh
    """
    tol_zero = 1e-13
    m, n = np.shape(A)
    S = np.block([[np.zeros((m, m)), A],
                  [A.T, np.zeros((n, n))]])
    sigma, W = linalg.eigh(S)
    # we are only interested in the positive eigenvalues
    singular_values = sigma[sigma > tol_zero]
    singular_vectors = W[:, sigma > tol_zero]
    # split the eigenvectors w = (u,v) and normalize
    U = singular_vectors[:m, :]
    U_norms = linalg.norm(U, axis=0)
    U = U @ np.diag(1. / U_norms)
    #
    V = singular_vectors[m:, :]
    V_norms = linalg.norm(V, axis=0)
    V = V @ np.diag(1. / V_norms)

    return U, V, singular_values


def test_example(m):
    """
    Compute SVD of 1st order finite difference matrix
    (discretization of the 1d gradient operator)
    The U and V are then related to the DCT and DST, respectively
    (D*T: Discrete Cosine/Sine Transform)
    """
    n = m+1
    A = np.eye(m, n, k=0) - np.eye(m, n, k=1)
    U, V, sigma = svd(A)
    A_reconstructed = np.round(U @ np.diag(sigma) @V.T, 10)
    print("Test:\n\t A = U * sigma * V.T \n\t is",
          np.allclose(A, A_reconstructed))
    X = np.linspace(0, 1, m+1)
    plt.figure()
    plt.title("V (DST)")
    plt.plot(X, V[:, 0:6])
    plt.show()
    plt.figure()
    plt.title("U (DCT)")
    plt.plot(X[1:], U[:, 0:6])
    plt.show()

    return None


if __name__ == "__main__":

    m = 100
    test_example(m)
```

Ex 98 Linear Algebra, Singular Values, Python

Using the SVD for Image Compression

Use the code snippet below (or an appropriate Python library) to load an image of your choice (extension .png or .jpg) as a gray scaled image $A \in \mathbb{R}^{m \times n}$.

1. Find a scipy routine to compute the SVD $U\Sigma V^T = A$.

2. Plot the singular values.

3. For several $1 \leq k \leq \text{rank}(A)$:

   a) Compute the *truncated SVD*: Use only the first k columns of U, the first k singular values $\sigma_1, \dots, \sigma_k$ from Σ and the first k rows of V^T to reconstruct A and plot the resulting image A_k using `plt.imshow(A_k, cmap='gray')`.

   b) For each k, compute the total number of floats that need to be stored for the truncated SVD A_k and compare it to the total number of floats that need to be stored for the full image A.

```
    """
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    print(np.shape(img))
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img
```

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as linalg


def load_image_as_gray(path_to_image):
    """
    loads image and returns luma transform as numpy array
    """
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    print(np.shape(img))
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img


def generate_video(U, Vt, sigma):
    from matplotlib import animation
    dpi = 250
    idlist = range(120)
    frames = []  # for storing the generated images
    fig = plt.figure()
    for i in idlist:
        aux = np.zeros((m, 2*n+50))
        aux[:, :n] = U[:, :i] @ np.diag(sigma[:i]) @ Vt[:i, :]
        aux[:, -n:] = A
        storage = np.round(100 * i * (1. + m + n) / (m * n), 2)
        frames.append([plt.imshow(aux, cmap='gray'),
                       plt.text(1, 1,
                                "k={}, storage={}%".format(str(idlist[i]),
                                   str(storage)),
                                horizontalalignment='left',
                                verticalalignment='top',
                                color="white"),
                                plt.title(
                                        "Truncated SVD A_k    VS    Original A"
                                        )])
    ani = animation.ArtistAnimation(fig,
                                    frames,
                                    interval=180,
                                    blit=True,
                                    repeat_delay=1000)
    ani.save('movie.mp4', dpi=dpi)
    plt.show()
    return None


if __name__ == "__main__":

    path_to_image = 'kalle.jpg'
    A = load_image_as_gray(path_to_image)

    U, sigma, Vt = linalg.svd(A)
    m, n = np.shape(A)

    # 1 plot singular values
    plt.figure()
    plt.subplot(3, 3, 1)
    plt.title(r'$\sigma$')
    x = np.linspace(1, sigma.size, sigma.size)
    plt.semilogy(x,
                 sigma,
                 'o-',
                 label='sigma',
                 markersize=3)
    plt.semilogy([100, 100, 0],
                 [0, sigma[100],
                  sigma[100]],
                 '-',
                 lw=2,
                 color='red')
    plt.grid(True)

    # 2 plot truncated svd images
    K = [1, 3, 5, 10, 20, 50, 100]
    for k in K:
        truncated_svd = U[:, :k] @ np.diag(sigma[:k]) @ Vt[:k, :]
        print("\nrank = {} \nrelative storage (truncated-SVD/A): {}%".format(
                k, np.round(100 * k * (1. + m + n) / (m * n), 2)))
        plt.subplot(3, 3, K.index(k)+2)
        plt.imshow(truncated_svd, cmap='gray')
    #     TODO: fix title issue
    #     plt.title("\nrank = {} \nrelative storage: {}%".format(
    #             k, np.round(100 * k * (1. + m + n) / (m * n), 2)))
        plt.tight_layout(pad=0.4, w_pad=0.05, h_pad=0.5)

    # 3 plot original
    plt.subplot(3, 3, 9)
    plt.imshow(A, cmap='gray')
    plt.title("original")

#     generate_video(U, Vt, sigma)
```

Ex 99 Linear Algebra, Singular Values

Equivalent Definitions of the Matrix Rank

Let $A \in \mathbb{R}^{m\times n}$, then use the SVD $A = U\Sigma V^\top$ to show that the following statements are equivalent:

i) The maximum number of linearly independent columns of A is r.

ii) The dimension of the image of A is r, i.e., $\dim(\text{im}(A)) = r$.

iii) The number of positive singular values of A is r.

As a consequence (since $A^\top = V\Sigma^\top U^\top$), we find $\text{rank}(A) = r = \text{rank}(A^\top)$, i.e., "column rank = row rank". The dimension formulas for A and $A^\top$ then read as

$$\begin{aligned} n &= \text{rank}(A) + \text{nullity}(A) = \text{rank}(A^\top) + \text{nullity}(A), \\ m &= \text{rank}(A^\top) + \text{nullity}(A^\top) = \text{rank}(A) + \text{nullity}(A^\top). \end{aligned}$$

Solution:

i) $\Rightarrow$ ii): Pick r independent columns of $A = [a_1, \dots, a_n]$, say $a_{i_1}, \dots, a_{i_r}$. Then by i) any other column of A can be written as linear combination of $a_{i_1}, \dots, a_{i_r}$, so that

$$\text{Im}(A) = \text{span}(a_1, \dots, a_n) = \text{span}(a_{i_1}, \dots, a_{i_r}).$$

Thus the $a_{i_1}, \dots, a_{i_r}$ are a basis of length r for $\text{Im}(A)$, implying ii) by the definition of "dimension".

ii) $\Rightarrow$ i): Let $\dim(\text{Im}(A)) = r$. Since any basis of a subspace has the same length, any basis of $\text{Im}(A)$ has length

r.
Now assume A has more than r independent columns. Then by the reasoning from above, these columns would yield another basis of $\text{Im}(A)$ but with length $> r$, which would contradict the fact that any basis has length r. Therefore implying i): the maximum number of independent columns in A is r.

ii) $\Leftrightarrow$ **iii)** We show that

$$\dim(\text{Im}(A)) = \text{number of positive singular values of } A.$$

Let us consider the reduced SVD $A = U_r\Sigma_r V_r^\top$ where r denotes the number of (positive) singular values, Σ_r is an invertible diagonal matrix and U_r and $V_r^\top$ have independent (even orthonormal) columns and rows, respectively. Thus

$$\text{Im}(A) = \text{Im}(U_r\Sigma_r V_r^\top).$$

We show that $\Sigma_r V_r^\top$ is surjective, i.e., $\text{rank}(\Sigma_r V_r^\top) = r$. This easily follows from the fact that $(\Sigma_r V_r^\top)^\top = V_r\Sigma_r$ has independent columns and thus

$$\text{nullity}(V_r\Sigma_r) = 0 \Leftrightarrow \text{rank}(\Sigma_r V_r^\top) = r.$$

Therefore by the lemma on the image of a product matrix we obtain

$$\text{Im}(A) = \text{Im}(U_r\Sigma_r V_r^\top) = \text{Im}(U_r).$$

Since the columns of U_r are independent they are a basis of length r for $\text{Im}(A)$, i.e., $\dim(\text{Im}(A)) = r$=number of (positive) singular values.

Ex 100

Consider the matrix

$$A = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 1 \end{pmatrix}.$$

1. Derive its singular value decomposition (SVD).
   (*Hint:* Compute $U\Sigma V^T$ to check your result.)
2. Write A as a sum of rank-1 matrices by using the singular values and vectors.
3. Is A invertible? Use the SVD to answer this question.

Solution:

$A = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 1 \end{pmatrix}$

SVD-Recipe: $\lambda \in \sigma(A^TA),\ \lambda \neq 0,\ \tilde{v}$ eigenvector

(i) $\sigma := \sqrt{\lambda}$

(ii) $v := \frac{\tilde{v}}{\|\tilde{v}\|}$

(iii) $u := \frac{1}{\sigma}Av$

1. Compute SVD and test:
   - $A^TA = \begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix}$
   - Eigenvalues:
     $\det(A^TA - \lambda I) = (\frac{3}{2} - \lambda)^2 - \frac{1}{4} = 0 \quad \Leftrightarrow \quad \lambda \in \{1, 2\}$
   - Eigenvectors:

     a)

$$(A^TA - \underbrace{\lambda_1}_{=2} I)\tilde{v} = 0 \quad \Leftrightarrow \quad \begin{pmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}\tilde{v} = 0$$

$$\Leftrightarrow \quad \tilde{v}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

b)

$$(A^TA - \underbrace{\lambda_2}_{=1} I)\tilde{v} = 0 \quad \Leftrightarrow \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tilde{v} = 0$$
$$\Leftrightarrow \quad \tilde{v}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

- $\sigma_1 := \sqrt{\lambda_1} = \sqrt{2},\ \sigma_2 := \sqrt{\lambda_2} = 1,$
  $v_1 := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix},\ v_2 := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix},$
  $u_1 := \frac{1}{\sigma_1} A v_1 = \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$
  $u_2 := \frac{1}{\sigma_2} A v_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} \frac{2}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- Thus:
$$\Sigma = \begin{pmatrix} \sqrt{2} & 0 \\ 0 & 1 \end{pmatrix}, \quad V = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
- Test:
$$U\Sigma V^T = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \underbrace{\begin{pmatrix} \sqrt{2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}_{= \begin{pmatrix} \sqrt{2} & \sqrt{2} \\ 1 & -1 \end{pmatrix}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ \sqrt{2} & \sqrt{2} \end{pmatrix} = A \ \checkmark$$

2.

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T = \frac{\sqrt{2}}{\sqrt{2}} \begin{pmatrix} 0 \\ 1 \end{pmatrix} (1, 1) + \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \end{pmatrix} (1, -1)$$
$$= \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} + \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 0 & 0 \end{pmatrix} \quad \checkmark$$

3. Yes, since $\sigma_1 \neq 0 \neq \sigma_2$.

Ex 101 Linear Algebra, Singular Values

Computation of the SVD

For a matrix $A \in \mathbb{R}^{m \times n}$ define

$$S := \begin{pmatrix} 0 & A \\ A^\top & 0 \end{pmatrix}.$$

1. Why does S only have real eigenvalues?
2. Determine all eigenpairs of S. Explain your answer.
3. Derive the reduced SVD of A from the eigenpairs of S.

Solution:

1. Because S is symmetric.
2. We already identify r eigenpairs for S, namely,

$$(\sigma_1, \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}), \ldots, (\sigma_r, \begin{pmatrix} u_r \\ v_r \end{pmatrix}),$$

   where $(\sigma_i, \begin{pmatrix} u_i \\ v_i \end{pmatrix})$ are the r singular values and vectors of A, respectively.

   Also, we easily find that

$$(-\sigma_1, \begin{pmatrix} -u_1 \\ v_2 \end{pmatrix}), \ldots, (-\sigma_r, \begin{pmatrix} -u_r \\ v_r \end{pmatrix})$$

are eigenpairs of S.
For the remaining $(m-r)+(n-r)$ eigenpairs take orthonomal bases $u_{r+1},\dots,u_m \in \ker A^\top$ and $v_{r+1},\dots,v_n \in \ker A$, then the $(0, \begin{pmatrix} u_i \\ 0 \end{pmatrix})$ and $(0, \begin{pmatrix} 0 \\ v_i \end{pmatrix})$ give the remaining eigenpairs (with eigenvalue 0).

3. Take the all eigenpairs (σ, w) of S corresponding to positive eigenvalues $\sigma > 0$. Split the eigenvectors $w = (u, v)$.

Ex 102 Linear Algebra, Singular Values

Compute the SVD
Consider the matrix

$$A = \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix}.$$

i) Compute its SVD $A = U\Sigma V^T$.

ii) Write A as a sum of rank-1 matrices.

iii) Is A invertible?

Hint: For i) follow this recipe:

1. Compute the eigenvalues λ_i with eigenvectors v_i of A^TA. Index the eigenvalues so that $\lambda_1 \geq \dots \geq \lambda_r > 0$, where $r :=$"number of positive eigenvalues". Normalize the eigenvectors v_i.
2. For $i = 1,\dots,r$: Set $\sigma_i := \sqrt{\lambda_i}$ and $u_i := \frac{1}{\sigma_i} A v_i$. [Until here we will already have the reduced SVD]
3. Extend the bases:
   - If $r < n$: Find orthonormal $v_{r+1},\dots,v_n \in \ker(A)$ by solving $Av_i = 0$ and orthogonalizing.
   - If $r < m$: Find orthonormal $u_{r+1},\dots,u_m \in \ker(A^T)$ by solving $A^T u_i = 0$ and orthogonalizing.

Solution:

$$A = \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix}, \quad A^TA = \begin{pmatrix} 25 & 20 \\ 20 & 25 \end{pmatrix}$$

i) COMPUTE SVD:

1. Compute $\sigma(A^TA)$ and corresponding eigenvectors.
   - Eigenvalues:

$$0 \overset{!}{=} \det(A^TA - \lambda I) = \det \begin{pmatrix} 25-\lambda & 20 \\ 20 & 25-\lambda \end{pmatrix} = (25-\lambda)^2 - 400$$
$$\Leftrightarrow \quad 25 - \lambda = \pm\sqrt{400} = \pm 20$$
$$\Leftrightarrow \quad \lambda_1 = 45, \quad \lambda_2 = 5.$$

   - Eigenvectors v_1 and v_2 are solutions of $(A^TA - \lambda_i I)v_i = 0$.
     a)

$$(A^TA - \lambda_1 I) = \begin{pmatrix} -20 & 20 \\ 20 & -20 \end{pmatrix}$$
$$\left(\begin{array}{cc|c} -20 & 20 & 0 \\ 20 & -20 & 0 \end{array}\right) \rightsquigarrow \left(\begin{array}{cc|c} -20 & 20 & 0 \\ 0 & 0 & 0 \end{array}\right)$$
$$\Rightarrow \quad -x_1 + x_2 = 0 \quad \Rightarrow \quad x_1 = x_2$$

$$\Rightarrow \quad v_1 \in \{s \begin{pmatrix} 1 \\ 1 \end{pmatrix} : s \in \mathbb{R}\}$$

Choose $s = \frac{1}{\sqrt{2}}$ and $v_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

b)

$$(A^TA - \lambda_2 I) = \begin{pmatrix} 20 & 20 \\ 20 & 20 \end{pmatrix}$$
$$\left(\begin{array}{cc|c} 20 & 20 & 0 \\ 20 & 20 & 0 \end{array}\right) \rightsquigarrow \left(\begin{array}{cc|c} 20 & 20 & 0 \\ 0 & 0 & 0 \end{array}\right)$$
$$\Rightarrow \quad x_1 + x_2 = 0 \quad \Rightarrow \quad x_1 = -x_2.$$

$\Rightarrow \quad v_2 \in \{s \begin{pmatrix} -1 \\ 1 \end{pmatrix} : s \in \mathbb{R}\}$

Choose $s = \frac{1}{\sqrt{2}}$ and $v_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

2. Set $\sigma_1 := \sqrt{45} = 3\sqrt{5}$, $\sigma_2 := \sqrt{5}$.
   Compute u_i:

$$u_1 = \frac{1}{\sigma_1} A v_1 = \frac{1}{3\sqrt{5}} \frac{1}{\sqrt{2}} \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{3\sqrt{1}} \begin{pmatrix} 3 \\ 9 \end{pmatrix} = \frac{1}{\sqrt{10}} \begin{pmatrix} 1 \\ 3 \end{pmatrix},$$
$$u_2 = \frac{1}{\sigma_2} A v_2 = \frac{1}{\sqrt{5}} \frac{1}{\sqrt{2}} \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{10}} \begin{pmatrix} -3 \\ 1 \end{pmatrix}.$$

3. Since $2 = r = m = n$, we are done.

ii) A as sum of rank-1 matrices:

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T = \underbrace{\frac{3\sqrt{5}}{\sqrt{10}\sqrt{2}}}_{\frac{3}{2}} \begin{pmatrix} 1 \\ 3 \end{pmatrix} (1\ 1) + \underbrace{\frac{\sqrt{5}}{\sqrt{10}\sqrt{2}}}_{\frac{1}{2}} \begin{pmatrix} -3 \\ 1 \end{pmatrix} (-1\ 1)$$
$$= \frac{3}{2} \begin{pmatrix} 1 & 1 \\ 3 & 3 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 3 & -3 \\ -1 & 1 \end{pmatrix}.$$

iii) A invertible?
Yes, since $\sigma_1 \neq 0 \neq \sigma_2$ and thus Σ is invertible implying that the product $U\Sigma V^T = A$ is invertible with inverse $A^{-1} = V\Sigma^{-1}U^T$.

Ex 103 Linear Algebra, Singular Values

The SVD and the Rank of a Matrix
Let $A \in \mathbb{R}^{n \times n}$ with SVD $A = U\Sigma V^T$ and define rank(A) := "number of positive singular values". Show that A is invertible $\iff$ rank$(A) = n$.

Solution:

$$\text{rank}(A) = n \quad \Leftrightarrow \quad \sigma_1, \ldots, \sigma_n \neq 0$$
$$\Leftrightarrow \quad \Sigma = \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{pmatrix} \text{ is invertible with } \Sigma^{-1} = \begin{pmatrix} \frac{1}{\sigma_1} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sigma_n} \end{pmatrix}$$
$$\Leftrightarrow A = U\Sigma V^T \text{ invertible with } A^{-1} = (U\Sigma V^T)^{-1} = V\Sigma^{-1}U^T.$$

Ex 104 Linear Algebra, Singular Values

The SVD of Rank-1-Matrices
Let $u \in \mathbb{R}^m \setminus \{0\}$ and $v \in \mathbb{R}^n \setminus \{0\}$ be nonzero vectors and define $A := uv^T$. Find a reduced SVD of A and shortly explain why rank$(A) = 1$.

Solution:

We will in all detail derive the full and reduced SVD by following our recipe (your answer may be shorter!):

We have
$$A^TA = \|u\|^2 vv^T.$$
Following the recipe for computing the SVD of A we determine the eigenpairs of A^TA. We find
$$A^TAv = \|u\|^2\|v\|^2 v,$$
which implies that v is an eigenvector to the positive (note: $u, v \neq 0$) eigenvalue $\|u\|^2\|v\|^2$.

- Since A^TA is symmetric we know all eigenvectors are mutually orthogonal.
- However, any vector x orthogonal to v is eigenvector to the eigenvalue 0, since
$$A^TAx = \|u\|^2 v \underbrace{v^Tx}_{=0} = 0 \cdot x.$$
Therefore the eigenvalues of A^TA are given by
$$\lambda_1 = \|u\|^2\|v\|^2,\ \lambda_2 = \cdots = \lambda_n = 0,$$
with precisely $r = 1$ positive one ($\Rightarrow$ rank$(A) = 1$). Thus we find the singular values and right-singular vector by
$$v_1 := \frac{v}{\|v\|},\ \sigma_1 = \|u\|\|v\| > 0.$$
Extend v_1 to the orthogonal matrix $V = \begin{pmatrix} | & \\ v_1 & \dots \\ | & \end{pmatrix} \in \mathbb{R}^{n\times n}$ with orthonormal columns, where $v_2, \dots, v_n \in \ker(A)$.
Also set
$$\Sigma = \text{diag}(\sigma_1, 0, \dots, 0) \in \mathbb{R}^{m\times n}.$$
- Following the recipe, the corresponding left-singular vector is given by
$$u_1 := \frac{Av_1}{\sigma_1} = \frac{1}{\|u\|\|v\|} uv^T \frac{v}{\|v\|} = \frac{u}{\|u\|} \underbrace{\frac{v^Tv}{\|v\|^2}}_{=1} = \frac{u}{\|u\|}.$$
Extend u_1 to the orthogonal matrix $U = \begin{pmatrix} | & \\ u_1 & \dots \\ | & \end{pmatrix} \in \mathbb{R}^{m\times m}$ with orthonormal columns, where $u_2, \dots, u_m \in \ker(A^T)$.
- All in all we then obtain the full and reduced (as sum of rank-1 matrices) SVD
$$\Rightarrow \quad A = U\Sigma V^T = \|u\|\|v\| u_1 v_1^T.$$

Ex 105 Linear Algebra, Singular Values

The SVD of Symmetric Matrices

Let $A \in \mathbb{R}^{n\times n}$ be symmetric with eigenvalues $\lambda_1, \dots, \lambda_n \in \mathbb{R}$.

1. What are the singular values of A?
2. How does the SVD look like? What can you say about the SVD if in addition A is positive definite?
3. Give a representation of the condition number and the rank of A in terms of its eigenvalues.

Solution:

REMARK: $A \in \mathbb{R}^{n\times n}$, (λ, v) eigenpair of A, then

$$A^k v = A^{k-1}\underbrace{(Av)}_{=\lambda v} = \dots = \lambda^k v \quad \forall k \in \mathbb{N}$$

$\Rightarrow (\lambda^k, v)$ is eigenpair of A^k.

1. A symmetric $\Rightarrow A^T A = A^2 \Rightarrow \sigma(A^T A) = \{\lambda^2 : \lambda \in \sigma(A)\}$,
   $\Rightarrow$ singular values are $\sigma = \sqrt{\lambda^2} = |\lambda|$ for $\lambda \in \sigma(A)$ nonzero

   ("Singular values of a symmetric matrix are the absolute values of its nonzero eigenvalues!")

2. How does the SVD look like?
   Let (λ_i, v_i) be eigenpairs of A, so that the v_i's are orthonormal. We set

$$\sigma_i = |\lambda_i|, \quad (i = 1, \dots, r)$$
$$u_i = \begin{cases} \frac{1}{\sigma_i} A v_i = \frac{\lambda_i}{|\lambda_i|} v_i & : \text{ for } i = 1, \dots, r \ \ (\lambda_i \neq 0), \\ v_i & : \text{ for } i = r+1, \dots, n \ (\lambda_i = 0). \end{cases}$$

   (note: we have shown in the lecture that the u_i as defined here are orthonormal and also note that $\ker(A) = \ker(A^\top)$).
   Then

$$A = V\Lambda V^\top \qquad \text{[eigendecomposition]}$$

$$\underbrace{= \underbrace{\begin{pmatrix} | & & | & | & & | \\ \frac{\lambda_1}{|\lambda_1|} v_1 & \dots & \frac{\lambda_r}{|\lambda_r|} v_r & v_{r+1} & \dots & v_n \\ | & & | & | & & | \end{pmatrix}}_{=:U} \underbrace{\begin{pmatrix} |\lambda_1| & & & & 0 \\ & \ddots & & & \\ & & |\lambda_r| & & \\ & & & \ddots & \\ 0 & & & & 0 \end{pmatrix}}_{=:\Sigma = \text{diag}(\sigma_1, \dots, \sigma_r)}}_{=V\Lambda} \underbrace{\begin{pmatrix} - & v_1^\top & - \\ & \vdots & \\ - & v_n^\top & - \end{pmatrix}}_{V^\top} \qquad [SVD]$$

   A symmetric and positive definite $\Rightarrow |\lambda_i| = \lambda_i > 0 \Rightarrow \frac{\lambda_i}{|\lambda_i|} = 1$. Thus:

   ("For spd matrices: SVD = Eigendecomposition!")

3. For the condition and the rank we obtain (by inserting $\sigma_i = |\lambda_i|$)

$$\text{cond}(A) = \frac{\max(\sigma_i)}{\min(\sigma_i)} = \frac{\max(|\lambda_i|)}{\min(|\lambda_i|)}$$

   and

$$\text{rank}(A) = |\{i : \sigma_i > 0\}| = |\{i : |\lambda_i| > 0\}| = |\{i : \lambda_i \neq 0\}|.$$

6 Iterative Methods

Ex 106

```
1 def fun(A,b, m=50):
2     n = A.shape[1]
3     x = np.zeros(n)
4     N = 1/A.diagonal()
5     for k in range(m):
6         x = x - N * (A @ x - b)
7         return x
```

1. Please describe what each line of the code does (please do not write into the pseudocode).
2. Which algorithm is implemented and what is its purpose? Which role does N play here?

Solution:

1.

$(1P)$ 1 function declaration with input $A, b, m := 50$

$(1P)$ 2 set $n :=$ number of columns of A

$(1P)$ 3 set $x = (0, \dots, 0)^T \in \mathbb{R}^n$

$(1P)$ 4 set $N = (\frac{1}{a_{11}}, \dots, \frac{1}{a_{nn}})^T$ (inverting diagonal elements)

5 /

$(1P)$ 6 for-loop from $k = 0$ to $k = m - 1$

$(1P)$ 7 update x by $x - D^{-1}(Ax - b),\ D = \begin{pmatrix} a_{11} & & 0 \\ & \ddots & \\ 0 & & a_{nn} \end{pmatrix}$

$(1P)$ 8 output value of x

2.

$(1P)$ Jacobi iteration

$(1P)$ solve $Ax = b$ iteratively

$(1P)$ N is preconditioner (with the "hope" $\rho(I - NA) < 1$)

Ex 107

1. Let $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$ be given.
   a) General linear iterations are of the form $x_{k+1} = (I - NA)x_k + Nb$ for some invertible matrix $N \in \mathbb{R}^{n\times n}$ and $x_0 \in \mathbb{R}^n$. Give a sufficient criteria for the convergence of the sequence $(x_k)_k$.
   b) How is the Jacobi iteration without relaxation defined?
   c) What is the purpose of the Jacobi iteration?
2. Assume you want to solve the system $Ax = b$, where

$$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

   a) Determine the iteration matrix $(I - NA)$ of the Jacobi iteration (without relaxation) in this example.
   b) Perform 3 iteration steps of the Jacobi iteration with initial guess $x_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, i.e., compute the iterates x_1, x_2, x_3.
   c) Does the Jacobi iteration converge here? Justify your answer.
   (*Hint: Look at task a.i again.*)

Solution:

1. a) (1P) $x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b$, where $D = \text{diag}(A) \in \mathbb{R}^{n\times n}$

   b) (1P) The purpose is to solve a linear system of the form $Ax = b$

   c) (1P) The iteration converges to a fixed point if the spectral radius $\rho(I - D^{-1}A) < 1$

2. a) Here we have

$$D^{-1} = \frac{1}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad D^{-1}A = \begin{pmatrix} 1 & -\frac{1}{2} \\ -\frac{1}{2} & 1 \end{pmatrix},$$

$$\text{thus} \quad I - D^{-1}A = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} \quad (2P)$$

   and therefore the Jacobi iteration is given here by

$$x_{k+1} = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix} x_k = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}^k x_0.$$

   b) We obtain

$$(1P)\ x_1 = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix}$$

$$(1P)\ x_2 = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{4} \end{pmatrix}$$

$$(1P)\ x_3 = \begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}\begin{pmatrix} 0 \\ \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{8} \\ 0 \end{pmatrix}$$

   c) (1P) Yes, the iteration converges.
   (1P) because $\rho(I - D^{-1}A) = \rho\left(\begin{pmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{pmatrix}\right) = \frac{1}{4} < 1$.

Ex 108

Weighted Jacobi, Gauß–Seidel and Sucessive Over–Relaxation
Let $A \in \mathbb{R}^{n\times n}$ be a matrix with nonzero diagonal entries $a_{ii} \neq 0$ and consider the splitting $A = L + D + U$ into lower triangular, diagonal and upper triangular part of A. Also recall that splitting methods are of the form

$$x^{k+1} = (I - NA)x^k + Nb,$$

where the matrix $M := I - NA$ is called iteration matrix.

Show the following:

1. **Weighted Jacobi:** $N = \theta D^{-1}$
   - For the iteration matrix we find

$$M_{Jac} := I - \theta D^{-1}A = (1-\theta)I - \theta D^{-1}(L+U)$$

   - The i-the component of $x^{k+1} = (I - NA)x^k + Nb$ is given by

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

2. **Gauß–Seidel:** $N = (L+D)^{-1}$
   - For the iteration matrix we find

$$M_{GS} := I - (L+D)^{-1}A = -(L+D)^{-1}U$$

- The i-the component of $x^{k+1} = (I - NA)x^k + Nb$ is given by

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

3. **Sucessive Over–Relaxation (variant of Gauß-Seidel):** $N = \theta \cdot (\theta L + D)^{-1}$
   - For the iteration matrix we find

$$M_{SOR} := I - \theta(\theta L + D)^{-1}A = (\theta L + D)^{-1}((1-\theta)D - \theta U).$$

   - The i-the component of $x^{k+1} = (I - NA)x^k + Nb$ is given by

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

   Remark: We observe that SOR for $\theta = 1$ is Gauß–Seidel and otherwise is a combination of the previous step x^k and the Gauß-Seidel update. For spd matrices it allows for $\theta > 1$ which is why it is called over–relaxation.

Hint: For 2. and 3. cast the formulas into the form $x^{k+1} = T^{-1}w$ for some lower triangular matrix T and some vector w and then use forward substitution.

Solution:

We first recall the forward substitution formula for inverting lower triangular matrices: Let $T = (\ell_{ij}) \in \mathbb{R}^{n\times n}$ be triangular and $w \in \mathbb{R}^n$, then the i-th component of $z = T^{-1}w$ is given by

$$z_i = \frac{1}{\ell_{ii}} \left(w_i - \sum_{j=1}^{i-1} \ell_{ij} z_j \right), \quad i \in \{1, \dots, n\}.$$

1. **Jacobi:**
   - Using the splitting $A = L + D + U$ we find

$$M_{Jac} := I - \theta D^{-1}A = I - \theta D^{-1}(L + D + U) = I - \theta(I + D^{-1}(L+U)) = (1-\theta)I - \theta D^{-1}(L+U)$$

   - The inverse of D is $D^{-1} = \text{diag}(\frac{1}{a_{11}}, \dots, \frac{1}{a_{nn}})$. Thus the i-th component of $\theta D^{-1}b$ is given by $\theta\frac{b_i}{a_{ii}}$. Now applying the definition of the matrix vector product we find for the i-th component of $\theta D^{-1}Ax^k$ that $\theta\frac{1}{a_{ii}}\sum_{j=1}^n a_{ij}x_j^k$. Combining this we obtain for the i-th of the Jacobi iterate the searched formula

$$\begin{aligned} x_i^{k+1} = x_i^k - \theta\frac{1}{a_{ii}}\sum_{j=1}^n a_{ij}x_j^k + \theta\frac{b_i}{a_{ii}} &= x_i^k - \theta x_i^k - \theta\frac{1}{a_{ii}}\sum_{j\neq i} a_{ij}x_j^k + \theta\frac{b_i}{a_{ii}} \\ &= (1-\theta)x_i^k + \frac{\theta}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}\, x_j^{k+1} \right). \end{aligned}$$

2. **Gauß–Seidel**
   - Using the splitting $A = L + D + U$ we find

$$M_{GS} := I - (L+D)^{-1}A = I - (L+D)^{-1}(L+D+U) = I - (I + (L+D)^{-1}U) = -(L+D)^{-1}U$$

   - We cast the formula into a lower triangular system:

$$\begin{aligned} x^{k+1} &= (I - NA)x^k + Nb = M_{GS}x^k + Nb \\ &= -(L+D)^{-1}Ux^k + (L+D)^{-1}b \\ &= (L+D)^{-1}(b - Ux^k) \end{aligned}$$

Now we consider $z = x^{k+1}$, $T = (L + D)$ and $w = b - Ux^k$ and apply forward substitution to obtain

$$\begin{aligned} x_i^{k+1} = z_i &= \frac{1}{\ell_{ii}}\left(w_i - \sum_{j=1}^{i-1} \ell_{ij} z_j\right), \quad i \in \{1, \dots, n\} \\ &= \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^k - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1}\right), \quad i \in \{1, \dots, n\}. \end{aligned}$$

3. **SOR**
   - We use
$$N = \theta \cdot (\theta L + D)^{-1} = \left(\tfrac{1}{\theta}\right)^{-1} (\theta L + D)^{-1} = \left(L + \tfrac{1}{\theta} D\right)^{-1}$$
     and the splitting
$$A = L + D + U = L + D + U \pm \tfrac{1}{\theta} D = (L + \tfrac{1}{\theta} D) + U + (1 - \tfrac{1}{\theta}) D.$$
     Then we find
$$\begin{aligned} M_{SOR} := I - NA &= I - \left(L + \tfrac{1}{\theta} D\right)^{-1} \left((L + \tfrac{1}{\theta} D) + U + (1 - \tfrac{1}{\theta}) D\right) \\ &= -\left(L + \tfrac{1}{\theta} D\right)^{-1} \left(U + (1 - \tfrac{1}{\theta}) D\right) \\ &= \left(L + \tfrac{1}{\theta} D\right)^{-1} \left(\tfrac{1-\theta}{\theta} D - U\right) \\ &= \theta \cdot (\theta L + D)^{-1} \left(\tfrac{1-\theta}{\theta} D - U\right) \\ &= (\theta L + D)^{-1} \left((1-\theta) D - \theta U\right). \end{aligned}$$
   - We cast the formula into a lower triangular system:
$$\begin{aligned} x^{k+1} &= (I - NA)x^k + Nb = M_{SOR} x^k + Nb \\ &= (\theta L + D)^{-1} \left((1-\theta) D - \theta U\right) x^k + \theta \cdot (\theta L + D)^{-1} b \\ &= (\theta L + D)^{-1} (\theta b - ((1-\theta) D - \theta U) x^k) \end{aligned}$$
     Now we consider $z = x^{k+1}$, $T = (\theta L + D)$ and $w = \theta b - (1-\theta) D x^k - \theta U x^k$ and apply forward substitution to obtain
$$\begin{aligned} x_i^{k+1} = z_i &= \frac{1}{\ell_{ii}}\left(w_i - \sum_{j=1}^{i-1} \ell_{ij} z_j\right), \quad i \in \{1, \dots, n\} \\ &= \frac{1}{a_{ii}}\left(\theta b_i - (1-\theta) a_{ii} x_i^k - \theta \sum_{j=i+1}^{n} a_{ij} x_j^k - \sum_{j=1}^{i-1} \theta a_{ij} x_j^{k+1}\right), \quad i \in \{1, \dots, n\} \\ &= (1-\theta) x_i^k + \frac{\theta}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^k - \sum_{j=1}^{i-1} a_{ij}\, x_j^{k+1}\right), \quad i \in \{1, \dots, n\}. \end{aligned}$$

Ex 109 Linear Systems, Krylov Subspace Methods

Arnoldi and Lanczos Iteration

Let $A \in \mathrm{GL}_n(\mathbb{R})$ and $b \in \mathbb{R}^n \setminus \{0\}$. Then consider the Arnoldi iteration as sketched in Algorithm 4 to produces an orthonormal basis $q_1, \dots, q_r$ of the r-th Krylov subspace $K_r(A, b)$ with $r \leq \max_{s \leq n} \dim(K_s(A, b))$. Further let $Q_r := [q_1, \dots, q_r] \in \mathbb{R}^{n \times r}$ and $H_r := Q_r^T A Q_r \in \mathbb{R}^{r \times r}$.

1. In the j-th step: Assume $q_1, \dots, q_j$ have been computed according to the Arnoldi iteration 4 and assume that $q_1, \dots, q_{j-1}$ are mutually orthonormal. Show that q_j is orthogonal to all $q_1, \dots, q_{j-1}$.
2. Derive an expression for the (ℓ, k)-th entries of H_r and find these numbers in the Arnoldi iteration. What structure does H_r have?
3. Now assume A is symmetric. How does H_r look in this case? How can you simplify the Arnoldi iteration?
4. How do the eigenvalues of H_n and A relate? Explain your answer.

```
INPUT: $A \in GL_n(\mathbb{R})$, $b \in \mathbb{R}^n$, $r \le n$
OUTPUT: orthonormal basis $q_1, \dots, q_r$ of the $r$-th Krylov subspace $K_r(A, b)$

$q_1 := \frac{b}{\|b\|_2}$
for $j = 2, \dots, r$ do
    $\widehat{q}_j := Aq_{j-1} - \sum_{\ell=1}^{j-1} q_\ell^\top (Aq_{j-1}) \cdot q_\ell$
    if $\|\widehat{q}_j\|_2 = 0$ then
        break
    end
    $q_j := \frac{\widehat{q}_j}{\|\widehat{q}_j\|_2}$
end
```

Algorithm 4: Arnoldi Iteration

Solution:

1. Let $k < j$. Since $q_k^\top q_j = \frac{1}{\|\widehat{q}_j\|_2} q_k^\top \widehat{q}_j$ it suffices to show that $q_k^\top \widehat{q}_j = 0$. Now let $v := Aq_{j-1}$, then
$$q_k^\top \widehat{q}_j = q_k^\top \left(v - \sum_{\ell=1}^{j-1} q_\ell^\top v \cdot q_\ell \right) = q_k^\top v - \sum_{\ell=1}^{j-1} q_\ell^\top v \cdot \underbrace{q_k^\top q_\ell}_{=\delta_{k\ell}} = q_k^\top v - q_k^\top v \cdot 1 = 0.$$

2. By definition of the matrix product we obtain, for $1 \le \ell, j \le r$,
$$H_r^{\ell j} = (Q_r^\top A Q_r)_{\ell j} = q_\ell^\top A q_j.$$
These are precisely the projection lengths that are computed during the Arnoldi iteration. Since by definition Aq_j can be uniquely generated by $q_1, \dots, q_{j+1}$ (i.e., $Aq_j = \sum_{\nu=1}^{j+1} \lambda_\nu q_\nu$ for some coordinates λ_ν), we have, for all $\ell > j+1$,
$$H_r^{\ell j} = q_\ell^\top A q_j = q_\ell^\top \left(\sum_{\nu=1}^{j+1} \lambda_\nu q_\nu \right) = \sum_{\nu=1}^{j+1} \lambda_\nu q_\ell^\top q_\nu = 0.$$
In particular, H_r is an upper *Hessenberg* matrix (having precisely one subdiagonal).

3. If A is symmetric, then $H_r = Q_r^\top A Q_r$ is symmetric, so that it simplifies to a tridiagonal matrix, i.e.,
$$H_r^{\ell j} = q_\ell^\top A q_j = 0$$
for all ℓ, j with $|\ell - j| > 2$. Then Arnoldi becomes Lanczos by accounting for the simplification
$$\begin{aligned}\widehat{q}_j &= Aq_{j-1} - \sum_{\ell=1}^{j-1} q_\ell^\top (Aq_{j-1}) \cdot q_\ell \\ &= Aq_{j-1} - \sum_{\ell=j-2}^{j-1} q_\ell^\top (Aq_{j-1}) \cdot q_\ell \\ &= Aq_{j-1} - q_{j-2}^\top (Aq_{j-1}) \cdot q_{j-2} - q_{j-1}^\top (Aq_{j-1}) \cdot q_{j-1}.\end{aligned}$$

4. Since $Q_n^T A Q_n \in \mathbb{R}^{n \times n}$ is orthogonally similar to A, it has the same eigenvalues as A.

Ex 110 Linear Systems, Splitting Methods

Convergence Speed of Linear Iterations

Let $M \in \mathbb{R}^{n \times n}$ be *symmetric* with $\rho(M) < 1$, let $N \in \mathbb{R}^{n \times n}$ and $x_0, b \in \mathbb{R}^n$. Consider the fixed point iteration
$$x_{k+1} = Mx_k + Nb$$
and show the following convergence result
$$\|x_k - x^*\|_2 \le \rho(M)^k \|x_0 - x^*\|_2,$$
where x^* is the associated fixed point. Thus, the smaller the spectral radius, the faster does the method converge.
Hint: For symmetric matrices $M \in \mathbb{R}^{n \times n}$ we have $\|Mx\|_2 \le \rho(M)\|x\|_2$ for all $x \in \mathbb{R}^n$.

Solution:

Since $\rho(M) < 1$, the iteration converges to the fixed point $x^* = Mx^* + Nb$. We use this representation in the formulas. We find

$$\|x^k - x^*\|_2 = \|Mx^{k-1} + Nb - (Mx^* + Nb)\|_2 = \|M(x^{k-1} - x^*)\|_2 \overset{\text{[Hint]}}{\leq} \rho(M) \underbrace{\|x^{k-1} - x^*\|_2}_{\leq \rho(M)\|x^{k-2} - x^*\|} .$$

By inserting the iteration instruction repeatedly we ultimately arrive at

$$\|x^k - x^*\|_2 \leq \rho(M)^k \|x^0 - x^*\|_2.$$

Ex 111 Linear Systems, Krylov Subspace Methods, Python

GMRES

1. **Arnoldi step:** For given orthonormal vectors $q_1, \dots, q_{r-1} \in \mathbb{R}^n$ considered as a matrix $Q_{r-1} = [q_1, \dots, q_{r-1}] \in \mathbb{R}^{n \times (r-1)}$, and a vector $v \in \mathbb{R}^n$, implement a helper function

   $$q_r, h_r := \texttt{Arnoldi\_step}(Q_{r-1}, v),$$

   which, according to the Arnoldi iteration, appends these vectors (i.e., the matrix Q_{r-1}) by an orthonormal vector q_r through orthogonalizing v against $q_1, \dots, q_{r-1}$ and also outputs the numbers $h_r := (h_{1,r-1}, \dots, h_{r,r-1})^\top \in \mathbb{R}^r$, where $h_{\ell,r-1} := q_\ell^\top v$ for $\ell \leq r$. You can then call `Arnoldi_step(Q; v)` within `GMRES(...)`.

2. **GMRES:** Implement a function

   $$x = \texttt{GMRES(A, b, x0, tol=1e-6, maxiter=None, N=None)},$$

   which takes as arguments

   - A : a function evaluating the matrix–vector product $v \mapsto A \cdot v$ for some matrix $A \in \mathbb{R}^{n \times n}$ (not as an array!)
   - b : a vector $b \in \mathbb{R}^n$
   - x0 : an arbitrary initial guess $x^0 \in \mathbb{R}^n$
   - tol : error tolerance as float, which is set to 10^{-6} by default
   - maxiter : optional maximum number of iterations, which is set to None by default
   - N : optional preconditioner as a function (not as an array), for which $N(v) \approx A^{-1}v$

   and then solves the system $Ax = b$ by applying the GMRES method as presented in the lecture (see pseudocode 5 below). It shall then return

   - x : the approximation to the solution.

   The iteration shall break if the residual is tolerably small, i.e.,

   $$\|Ax^k - b\|_2 < \texttt{tol}$$

   or the maximum number of iterations `maxiter` has been reached.

3. **Test** your solver on a random invertible tridiagonal matrix

   $$A = \begin{pmatrix} * & * & 0 & \cdots & 0 \\ * & * & * & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & * & * & * \\ 0 & \cdots & 0 & * & * \end{pmatrix} \in \mathbb{R}^{n \times n}$$

   and some right-hand side b and initial guess x_0 of your choice. Choose different n and check how many iterations you need (potentially many!).
   Hint: You can generate some random diagonals using `numpy.random.rand(n)` *and then use*

   ```
   scipy.sparse.diags(diagonals, offsets=[-1,0,1]).tocsr()
   ```

   to construct a sparse CSR matrix. You can add `np.ones(n)` *to the main diagonal in order to "strengthen" the diagonal of A and thereby to get a better conditioned system. Then implement a function* `A(x)` *that outputs the matrix-vector product* `A.dot(x)`.

4. **Preconditioner:** For the same system, run your GMRES routine with a preconditioner of your choice (for example Jacobi $N : v \mapsto D^{-1}v$). Do you observe any difference in the number of iterations needed? (You may not necessarily observe a difference!)

5. **Compare** your `GMRES` solver to `scipy.linalg.solve(A_csr.toarray(),b)` for large dimension $n \geq 10^5$ and measure the time needed in each case. Also, find a SciPy implementation of GMRES and compare to yours.

```
INPUT: $A \in GL_n(\mathbb{R})$, $b \in \mathbb{R}^n$
OUTPUT: approximation $x_r \in K_r(A, b)$ to the exact solution $A^{-1}b$

GMRES($A$, $b$, $x_0 = 0$, tol = 1e-6, maxiter=None, $N = I$ ):
$b := b - Ax_0$ //account for initial guess
$A := NA$, $b := Nb$ //account for preconditioner
//Initialization:
$q_1 := \frac{b}{\|b\|_2}$, $Q_1 := [q_1]$
$v := Aq_1$
$\tilde{q}_1 := \frac{v}{\|v\|_2}$, $\tilde{Q}_1 := [\tilde{q}_1]$, $\tilde{R}_1 = [\|v\|_2]$
for $r = 2, ..., \min(n, \text{maxiter})$ do
    //STEP 1: use Arnoldi to find column $q_r$ by orthogonalizing $v$ against $q_1, \ldots, q_{r-1}$
    $q_r, h_{r-1}$ := Arnoldi_step($Q_{r-1}; v$) //we don't need $h_{r-1}$ in this variant
    $Q_r := [Q_{r-1}, q_r]$
    $v := Aq_r$

    //STEP 2: use Arnoldi to find columns $\tilde{q}_r$, $\widetilde{r}_r$ by orthogonalizing $v$ against $\tilde{q}_1, \ldots, \tilde{q}_{r-1}$
    $\tilde{q}_r, \tilde{r}_r$ := Arnoldi_step($\widetilde{Q}_{r-1}; v$)
    $\widetilde{Q}_r := [\widetilde{Q}_{r-1}, \tilde{q}_r]$, $\widetilde{R}_r := [\widetilde{R}_{r-1}, \tilde{r}_r]$

    //STEP 3: solve auxiliary least squares problems to obtain coordinates
    $c_r$ := solve_triangular($\widetilde{R}_r, \widetilde{Q}_r^\top b$)
    $x_r := Q_r c_r$
    //Attention: Evaluate the original residual here:
    if $\|N^{-1}(Ax_r - b)\|_2 <$ tol then
        break
    end
end
return $x_0 + x_r$
```

Algorithm 5: GMRES

Solution:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np
import scipy.linalg
from numpy.linalg import norm
import scipy.linalg as linalg
import scipy.sparse as sparse
import scipy.sparse.linalg as spla
import scipy
#import matplotlib.pyplot as plt
from time import time


# --------------------------------------------------------------- #
#    One Arnoldi (Lanczos) step
# --------------------------------------------------------------- #
def Arnoldi_step(Q, v):
    """
    perform one step of Arnoldi iteration to orthogonalize v against the
    columns of Q

    Parameters:
    -----------
    Q : matrix containing the orthonormal vectors as columns
    v : vector

    Returns:
    --------
    qr : vector which is orthonormal to all columns in Q
    hr : vector containing the linear coefficients so that
         v = [Q,qr] @ hr
    """
    r = np.shape(Q)[1]
    hr = np.zeros(r+1)
    for j in range(r):
        hr[j] = np.dot(Q[:, j], v)
        v = v - hr[j] * Q[:, j]
    hr[-1] = norm(v)
    return v / hr[-1], hr


def Lanczos_step(Q, v):
    r = np.shape(Q)[1]
    hr = np.zeros(r+1)
    for j in range(max(0, r-2), r):
        hr[j] = np.dot(Q[:, j], v)
        v = v - hr[j] * Q[:, j]
    hr[-1] = norm(v)
    return v / hr[-1], hr


# --------------------------------------------------------------- #
#    GMRES
# --------------------------------------------------------------- #
def GMRES(A, b, x0, tol=1e-6, maxiter=None, N=None, sym=False):
    """
    solves a system Ax = b, where A is assumed to be invertible,
    using QR-based GMRES
    Parameters
    ----------
    A : python function
        for evaluating the matrix-vector product
    b : (n,) numpy.ndarray
         right-hand side
    x0: (n,) numpy.ndarray
         initial guess
    tol : float
          iteration stops if ||Axk - b|| < tol, tol = 1e-6
    maxiter : int (optional)
              maximum number of iterations
    N : python function (optional)
        for evaluating matrix-vector product of preconditioner
    sym : bool (optional)
          indicating whether A is symmetric or not
          if sym=True: Lanczos is used over Arnoldi

    Returns
    -------
    x : (n,) numpy.ndarray
        approximate solution to Ax=b, with ||Ax-b||<tol
    info : dict
    """
    # account for initial guess
    boriginal = b
    b = boriginal - A(x0)
    # account for preconditioner A(v) := A(N(v))
    if N:
        Aoriginal = A

        def A(x):
            return N(Aoriginal(x))
        b = N(b)

    # Initializing
    Q = b / norm(b)
    Q = Q[:, np.newaxis]
    v = A(Q[:, 0])
    tQ, tR = v / norm(v), norm(v)
    tQ = tQ[:, np.newaxis]
    n = len(b)
    xr = np.zeros(n)

    # if A is symmetric we (can) use Lanczos instead of Arnoldi
    if sym:
        def ortho(Q, v):
            return Lanczos_step(Q, v)
    else:
        def ortho(Q, v):
            return Arnoldi_step(Q, v)

    if maxiter:
        maxiter = min(maxiter, n+1)
    else:
        maxiter = n+1

    for r in range(1, maxiter):
        # STEP 1: Find next orthonormal basis vector of Krylov subspace
        qr, hr = ortho(Q, v)  # we do not use hr in our variant
        Q = np.hstack((Q, qr[:, np.newaxis]))
        v = A(Q[:, -1])

        # STEP 2: Find QR decomposition of AQ_r by appending previous one
        tqr, trr = ortho(tQ, v)
        tQ = np.hstack((tQ, tqr[:, np.newaxis]))
        tR = np.hstack((np.vstack((tR, np.zeros((1, r)))), trr[:, np.newaxis]))

        # STEP 3: Solve least squares problem involving
        #         AQ_k using its QR-decomposition
        cr = linalg.solve_triangular(tR, tQ.T@b)
        xr = Q @ cr

        # Evaluate current Error and break if its small enough
        lsq_err = norm(boriginal - Aoriginal(x0+xr))

        # collect some infos
        info = dict(iterationCount=r+1,
                    dimension=n,
                    residualNorm=lsq_err)
        if lsq_err < tol:
            break
    return x0 + xr, info


def main(compare_LU_dense=1, compare_Scipy=0):
    # generate the random system matrix and rhs, as well as initial guess
    start_time = time()
    n = 10000  # 10000 # 100000
#    A_sparseMatrix =   (2 * sparse.eye(n, k=0) -
#                        1 * sparse.eye(n , k=1) -
#                        1 * sparse.eye(n , k=-1))
    # we regularize the matrix to prevent ill-conditioned systems
    diagonals = [np.random.rand(n) + 1.5 * np.ones(n),
                 np.random.rand(n-1),
                 np.random.rand(n-1)]
    A_sparseMatrix = scipy.sparse.diags(diagonals, [0, 1, -1]).tocsr()
    print("\n Dimension:", n)
    print(f" Time to generate the matrix: {time()-start_time:0.2f} [s]")

    def A_function(x):  # The matrix vector product as a function
        return A_sparseMatrix.dot(x)
    b = np.random.rand(n)  # np.ones(n) #
    x0 = np.random.rand(n)  # np.zeros(n) # b#*100 #
    # GMRES parameters
    tol = 1e-06
    maxiter = 1000
    sym = False

    # --------------------- #
    #    Preconditioner
    # --------------------- #
    # Jacobi
    precondJacobiArray = sparse.diags(1. / A_sparseMatrix.diagonal())
    def precondJacobi(b): return precondJacobiArray.dot(b)
    # none
    def precondNone(b): return b
    # Gauss-Seidel
    precondGSinvArray = scipy.sparse.tril(A_sparseMatrix).tocsr()
    def precondGS(b): return spla.spsolve(precondGSinvArray, b)
    # choice
    preconditionerFunctionDict = {"none": precondNone,
                                  "Jacobi": precondJacobi,
                                  "GS": precondGS}
    # runtime parameter

    for preconditioner in ["none", "Jacobi", "GS"]:
        print("\n"+"*"*35+"\nPreconditioner: "+preconditioner+"\n"+"*"*35)

        preconditionerFunction = preconditionerFunctionDict[preconditioner]

        # ------------------------------ #
        #    Our GMRES
        # ------------------------------ #
        print("-" * 30 + "\n\t Our GMRES \n" + "-" * 30)
        start_time = time()
        xk, info = GMRES(A_function, b, x0=x0, tol=tol,
                         maxiter=maxiter, N=preconditionerFunction, sym=sym)
        print(f" Solving time = {time()-start_time:0.2f} [s]")
        print(" Number of iterations:\t", info["iterationCount"],
              f"\n Residual norm: {info['residualNorm']:0.2e}\n")
#         print(" 'Ax = b' is", np.allclose(A_function(xk), b))

        if compare_Scipy:
            # ------------------------------ #
            #    Scipy's GMRES
            # ------------------------------ #
            print("-" * 30 + "\n\t SciPy's GMRES \n" + "-" * 30)

            # callback for gmres
            class gmres_counter(object):
                def __init__(self):
                    self.niter = 0

                def __call__(self, rk=None):
                    self.niter += 1

            counter = gmres_counter()
            start_time = time()
            x, eC = \
            spla.gmres(A_sparseMatrix, b, x0=x0, tol=tol, maxiter=maxiter,
                       M=spla.LinearOperator((n, n), preconditionerFunction),
                       callback=counter)
            print(" Successful exit: ", eC == 0)
            print(f"\n Solving time = {time()-start_time:0.2f} [s]")
            print(" Number of iterations:\t", counter.niter)
            print(f" Residual norm: {norm(b - A_sparseMatrix.dot(x)):0.2e}")
#             print("\n 'Ax = b' is", np.allclose(A_sparseMatrix.dot(x), b))

    # -------------------------------- #
    #    Compare to Scipy's LU (dense)
    # -------------------------------- #
    if compare_LU_dense:
        print("\n" + "-" * 30 + "\n\t LU dense \n" + "-" * 30)
        start_time = time()
        x = linalg.solve(A_sparseMatrix.toarray(), b)
        print(f" Solving time = {time()-start_time:0.2f} [s]")
        print(f" Residual norm: {norm(b - A_sparseMatrix.dot(x)):0.2e}")


if __name__ == "__main__":
    main(compare_LU_dense=1, compare_Scipy=0)
```

Ex 112 Linear Systems, Splitting Methods, Python

Splitting Methods: relax. Richardson, relax. Jacobi, Gauß-Seidel and SOR

1. Implement a function

   `x, error, numiter = iter_solve(A, b, x0, method="Jacobi", theta=.1, tol=1e-08, maxiter=50)`

   which takes as arguments
   - A : a matrix $A \in \mathbb{R}^{n\times n}$
   - b : a vector $b \in \mathbb{R}^n$
   - x0 : an initial guess $x^0 \in \mathbb{R}^n$
   - method : optional parameter to choose between relax. Richardson, weighted Jacobi, Gauß-Seidel and SOR and which is set to "Jacobi" by default
   - theta : relaxation parameter θ which is set to 0.1 by default (note: Gauß–Seidel is SOR with theta=1.0)
   - tol : error tolerance as float, which is set to 10^{-8} by default
   - maxiter : maximum number of iterations, which is set to 50 by default

   and then solves the system $Ax = b$ by applying the specified iterative scheme. It shall then return
   - x : list of all iterates x^k
   - error : list containing all residuals $\|Ax^k - b\|_2$
   - numiter : number of iterations that have been performed

   The iteration shall break if the residual is tolerably small, i.e.,

   $$\|Ax^k - b\|_2 < \texttt{tol}$$

   or the maximum number of iterations maxiter has been reached. Implement the **element-based** formulas for the Jacobi, Gauß-Seidel and SOR method.

2. **2d:** Test all methods on the following two-dimensional setting:

   $$A = 4\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

   What is the exact solution x^*? Play around with the parameters x0, theta, tol and maxiter. Also create the following two plots for one fixed setting:
   - Plot the error $\|Ax^k - b\|_2$ for each iterate $x^k \in \mathbb{R}^2$, $k = 1, \dots, m$, for all methods into one plot (use different colors).
   - Plot the iterates $x^k \in \mathbb{R}^2$, $k = 1, \dots, m$, for all methods into one plot (use different colors).

3. **nd:** Next, test all methods on the higher–dimensional analogue

   $$A = n^2 \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \dots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n},$$

   for different dimensions $n \in \mathbb{N}$ and data $b, x^0 \in \mathbb{R}^n$ of your choice. Play around with the parameters.

Hint: It can happen that the iterations do not converge. Use small values for θ when you use the Richardson iteration. This will assure that $\rho(I - NA) < 1$.

Solution:

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")


# --------------------------------------------------------------#
#                    ITERATIVE SOLVER
# --------------------------------------------------------------#
def Richardson_step(A, b, theta, x):
    return x - theta * (A @ x - b)


def Jacobi_step(A, b, theta, x):
    n = len(x)
    xnew = np.zeros(n)
    for i in range(n):
        s1 = np.dot(A[i, :i], x[:i])
        s2 = np.dot(A[i, i + 1:], x[i + 1:])
        xnew[i] = (1. - theta) * x[i] + theta / A[i, i] * (b[i] - s1 - s2)
    return xnew


def SOR_step(A, b, theta, x):
    n = len(x)
    xnew = np.zeros(n)
    for i in range(n):
        s1 = np.dot(A[i, :i], xnew[:i])  # <-- here we use already the new info
        s2 = np.dot(A[i, i + 1:], x[i + 1:])
        xnew[i] = (1. - theta) * x[i] + theta / A[i, i] * (b[i] - s1 - s2)
    return xnew


def steepest_descent_step(A, b, theta, x):
    r = A@x - b
    theta = np.dot(r, r) / np.dot(A@r, r)
    return x - theta * (A @ x - b)


def conjugate_gradient(A, b, x0, maxiter=50, tol=1e-8):
    X = [x0]
    n = len(x0)
    error = []
    r = b - A @ X[0]
    p = r
    alpha_alt = np.dot(r, r)
    for numiter in range(max(maxiter, n)):
        error += [np.linalg.norm(A.dot(X[-1]) - b)]
        if error[-1] < tol:
            return X, error, numiter
        v = A @ p
        lambd = alpha_alt / np.dot(v, p)
        X += [X[-1] + lambd * p]
        r = r - lambd * v
        alpha_neu = np.dot(r, r)
        p = r + alpha_neu/alpha_alt * p
        alpha_alt = alpha_neu
    return X, error, numiter


def iter_solve(A, b, x0, method="Jacobi", theta=.1, maxiter=50, tol=1e-8):
    """
    solves a system Ax = b, where A is assumed to be invertible,
    with splitting methods

    Parameters
    ----------
    A : (n, n) numpy.ndarray
         system matrix
    b : (n,) numpy.ndarray
         right-hand side
    x0: (n,) numpy.ndarray
         initial guess
    method : string
             indicates method: "Jacobi" (=default), "Richardson", "GS", "SOR"
    theta : number (int or float)
            relaxation parameter (step length) default theta = 0.1
    tol : number (float)
            error tolerance, iteration stops if ||Ax-b|| < tol
    maxiter : int
        number of iterations that are performed , default m=50

    Returns
    -------
    X : list of length N (<=m), containing iterates
        columns represent iterates from x_0 to x_(N-1)
    error : list of length numiter containing norm of all residuals
    numiter : integer indicating how many iterations have been performed
    """

    if method in ["GS", "SOR", "Jacobi", "steepestDescent"] and \
       np.prod(A.diagonal()) == 0:
        print(f"WARNING: Method was chosen to be {method} \
              but A has zero diagonal entries!")
        return None
    elif method == "GS":
        theta = 1.0
        method = "SOR"
    elif method == "CG":
        return conjugate_gradient(A, b, x0, maxiter=maxiter, tol=tol)
    # choose the function to compute the iteration instruction
    # according to method
    step_instruction_dict = {"Jacobi": Jacobi_step,
                             "Richardson": Richardson_step,
                             "SOR": SOR_step,
                             "steepestDescent": steepest_descent_step}
    step_instruction = step_instruction_dict[method]

    # ITERATION
    X = [x0]
    error = []
    for numiter in range(maxiter):
        error += [np.linalg.norm(A.dot(X[-1]) - b)]
        if error[-1] < tol:
            return X, error, numiter
        X += [step_instruction(A, b, theta, X[-1])]

    return X, error, numiter


def main(A, b, x0, maxiter, methThet, plot=False, verbose=False):

    X = np.zeros((maxiter, 2, len(methThet)))
    colors = ['r', 'g', 'b', "y", "c", "m"]
    # ---------------------------------------------------------------#
    #                     PLOT error and iterates
    # ---------------------------------------------------------------#
    if plot:
        plt.figure()
    for i, method in enumerate(methThet):
        X, error, numiter = iter_solve(A, b, x0, method=method,
                                       theta=methThet[method],
                                       maxiter=maxiter)
        if verbose:
            print(method, "\n \t\t  >", f"NumIter = {numiter}",
                  f"\t residual = {error[-1]:0.2e}")
        if plot:
            plt.subplot(1, 2, 1)
            plt.plot(error, colors[i]+"-x")
            plt.title("Residual $||Ax_k - b||_2$")
            plt.legend(list(methThet.keys()))
            plt.subplot(1, 2, 2)
            X = np.array(X)
            plt.plot(X[:, 0], X[:, 1], colors[i] + "o-")
            plt.legend(list(methThet.keys()))
            plt.title("Iterates $x_k$")
            plt.axis("equal")
    if plot:
        plt.show()
        plt.axis("equal")
    return X, error, numiter


if __name__ == "__main__":
    # ---------------------------------------------------------------#
    #       2d EXAMPLE
    # ---------------------------------------------------------------#
    A = 4. * np.array([[2, -1],
                       [-1, 2]])
    b = np.zeros(2)
    x0 = np.array([5, 8])
    maxiter = 100
    methThet = {"Richardson": 0.1,
                "Jacobi": 1,
                "GS": -1,
                "SOR": 1.2,
                "steepestDescent": -1,
                "CG": -1}
    print("-"*40+"\n 2d EXAMPLE (numiter, error) \n"+"-"*40)
    X, error, numiter = main(A, b, x0, maxiter,
                             methThet, plot=True, verbose=True)

    # ---------------------------------------------------------------#
    #               HIGHER DIMENSIONAL EXAMPLE
    # ---------------------------------------------------------------#
    n = 150  # 10000 # 100000
    A = n ** 2 * (2 * np.eye(n, k=0) - np.eye(n, k=1) - np.eye(n, k=-1))
    b = np.random.rand(n)
    x0 = np.random.rand(n)  # b#
    firstMmethods = 4
    maxiter = 1000
    methThet = {"Richardson": 0.00001,
                "Jacobi": 0.001,
                "GS": -1,
                "SOR": 1.9,
                "steepestDescent": -1,
                "CG": -1}
    print("-"*40 + "\n nd EXAMPLE with n = {}\n".format(n) + "-"*40)
    X, error, numiter = main(A, b, x0, maxiter,
                             methThet, plot=0, verbose=True)
```

Ex 113

Richardson Iteration

Let $A \in \mathbb{R}^{n\times n}$ be a symmetric and positive definite matrix. Consider the (relaxed) Richardson iteration

$$x_{k+1} = (I - \theta A)x_k + \theta b$$

with (symmetric) iteration matrix $M_\theta := I - \theta A$.

1. Let λ_{max} (λ_{min}) denote the largest (smallest) eigenvalue of A. Show that the spectral radius of M_θ is given by

   $$\rho(M_\theta) = \max\{|1 - \theta\lambda_{max}|,\ |1 - \theta\lambda_{min}|\}.$$

   Hint: Note that A has only positive eigenvalues. Determine the spectrum of M_θ by observing that it is a scaled and shifted version of A.

2. Determine all $\theta \in \mathbb{R}$ for which the Richardson iteration converges.
   Hint: Use 1. and find all $\theta \in \mathbb{R}$ for which $\rho(M_\theta) < 1$.

3. Draw the function $\theta \mapsto \rho(M_\theta)$ to determine the optimal $\hat{\theta}$, which fulfills $\rho(M_{\hat{\theta}}) \leq \rho(M_\theta)$ for all $\theta \in \mathbb{R}$.
   Hint: We find $\hat{\theta} = \frac{2}{\lambda_{max}+\lambda_{min}}$.

4. Show that for the optimal relaxation parameter $\hat{\theta}$ it holds that

   $$\rho(M_{\hat{\theta}}) = \frac{\mathrm{cond}(A) - 1}{\mathrm{cond}(A) + 1},$$

   where $\mathrm{cond}(A) := \frac{\lambda_{max}}{\lambda_{min}}$ denotes the condition number of the symmetric matrix A. What impact does a large condition number (i.e., $\mathrm{cond}(A) \gg 1$) have on the convergence speed?

Solution:

We have $x^{k+1} = \underbrace{(I - \theta A)}_{=:M_\theta} x^k + \theta b = M_\theta x^k + \theta b$.

1. By definition we have $\rho(M_\theta) = \max_{\lambda\in\sigma(M_\theta)} |\lambda|$.
   a) Thus we first determine the spectrum of $M_\theta = I - \theta A$:

   $$\begin{aligned} \lambda \in \sigma(A) &\overset{\text{[scaling:]}\cdot(-\theta)}{\Rightarrow} -\theta\lambda \in \sigma(-\theta A) \\ &\overset{\text{[shift:]}+1}{\Rightarrow} 1 - \theta\lambda \in \sigma(I - \theta A) \end{aligned}$$

   $$\Rightarrow \quad \sigma(M_\theta) = \{1 - \theta\lambda :\ \lambda \in \sigma(A)\}$$

   b) Determine the maximum of this set:
   Since $\sigma(A) \subset (0, +\infty)$, we obtain

   $$\begin{aligned} & 1 - \theta\lambda_{\mathrm{max}} \leq 1 - \theta\lambda \leq 1 - \theta\lambda_{\mathrm{min}} && \forall\lambda \in \sigma(A) \\ \Rightarrow\ & |1 - \theta\lambda| \leq \max\{|1 - \theta\lambda_{\mathrm{max}}|,\ |1 - \theta\lambda_{\mathrm{min}}|\} && \forall\lambda \in \sigma(A) \\ \Rightarrow\ & \rho(M_\theta) = \max\{|1 - \theta\lambda_{\mathrm{max}}|,\ |1 - \theta\lambda_{\mathrm{min}}|\}. \end{aligned}$$

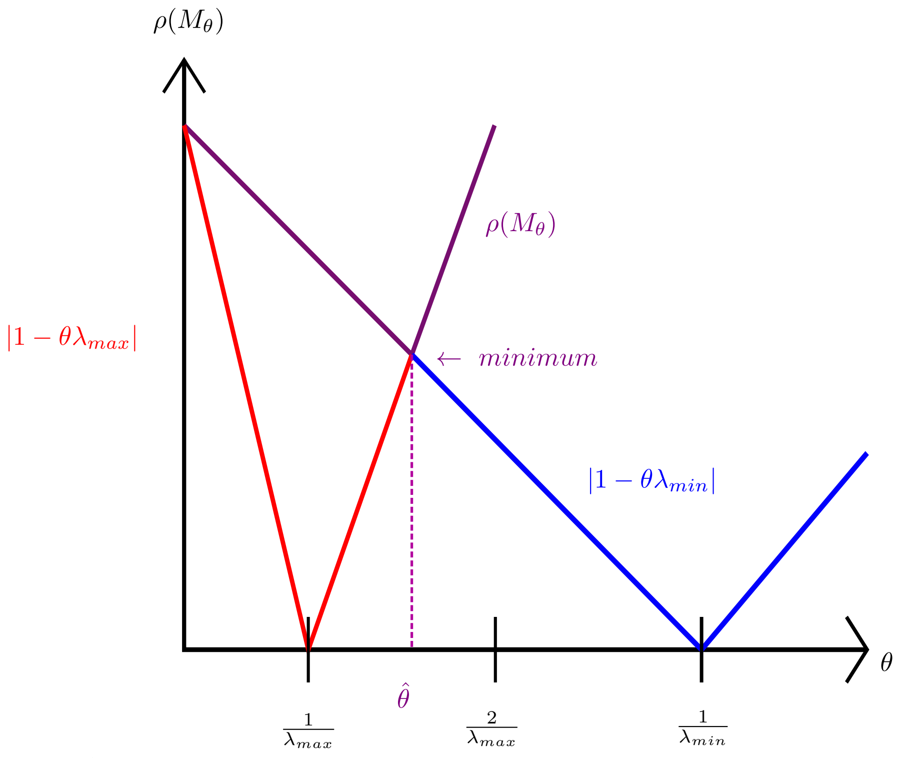
2.

   $$\begin{aligned} & \rho(M_\theta) = \max\{|1 - \theta\lambda_{\mathrm{max}}|,\ |1 - \theta\lambda_{\mathrm{min}}|\} < 1 \\ \Leftrightarrow\ & -1 \overset{a)}{<} 1 - \theta\lambda_{\mathrm{max}} \leq 1 - \theta\lambda_{\mathrm{min}} \overset{b)}{<} 1 \\ a)\ \Leftrightarrow\ & -2 < -\theta\lambda_{\mathrm{max}} \quad \Leftrightarrow \quad \theta < \frac{2}{\lambda_{\mathrm{max}}} \\ b)\ \Leftrightarrow\ & -\theta\lambda_{\mathrm{min}} < 0 \quad \Leftrightarrow \quad \theta > 0 \end{aligned}$$

   All in all:

   $$\rho(M_\theta) < 1 \quad \Leftrightarrow \quad \theta \in \left(0, \frac{2}{\lambda_{\mathrm{max}}}\right)$$

3. Let us plot $\theta \mapsto \rho(M_\theta) = \max\{|1-\theta\lambda_{\text{max}}|,\ |1-\theta\lambda_{\text{min}}|\}$



Thus: $\hat\theta$ is determined by the intersection

$$\begin{aligned} \text{"/"} \qquad -(1-\theta\lambda_{\text{max}}) &\overset{!}{=} 1-\theta\lambda_{\text{min}} \qquad \text{"\textbackslash"} \\ \Leftrightarrow \quad \theta\lambda_{\text{max}} - 1 &= 1-\theta\lambda_{\text{min}} \\ \Leftrightarrow \quad \theta &= \frac{2}{\lambda_{\text{min}}+\lambda_{\text{max}}} \end{aligned}$$

4. Inserting 3. into 2. gives

$$\begin{aligned} \rho(M_\theta) &= |1-\hat\theta\lambda_{\text{min}}| \quad (= |1-\hat\theta\lambda_{\text{max}}|) \\ &= \left|1 - \frac{2}{\lambda_{\text{min}}+\lambda_{\text{max}}}\lambda_{\text{min}}\right| \\ &= \left|\frac{\lambda_{\text{max}}-\lambda_{\text{min}}}{\lambda_{\text{min}}+\lambda_{\text{max}}}\right| \\ &= \frac{\lambda_{\text{max}}-\lambda_{\text{min}}}{\lambda_{\text{min}}+\lambda_{\text{max}}} \\ &= \frac{\lambda_{\text{min}}\left(\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}-1\right)}{\lambda_{\text{min}}\left(1+\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}\right)} \\ &= \frac{\text{cond}(A)-1}{\text{cond}(A)+1}. \end{aligned}$$

Thus:

$$\begin{aligned} &(1)\ \text{cond}(A) \gg 1 \quad \Rightarrow \quad \rho(M_\theta) \approx 1 \quad \to \quad \text{slow convergence} \\ &(2)\ \text{cond}(A) \approx 1 \quad \Rightarrow \quad \rho(M_\theta) \approx 0 \quad \to \quad \text{fast convergence} \end{aligned}$$

Ex 114 Linear Systems, Splitting Methods

Estimate Number of Iterations
Consider the linear system $Ax = b$ with

$$A = \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}.$$

Assume you want to solve this system with the Richardson, Jacobi and Gauß-Seidel method, respectively, without relaxation ($\theta = 1$). Estimate the number m of iterations that are needed for each method (if convergent) to reduce the error by a factor of $\varepsilon = 10^{-6}$, i.e.,

$$\|x_k - x\|_2 \leq \varepsilon \|x_0 - x\|_2.$$

Hint: Use the error estimate from previous exercises. You can later numerically verify your results.

Solution:

Since A is symmetric we can exploit the estimate

$$\|x_k - x\|_2 \leq \rho(M)^k \|x_0 - x\|_2,$$

where $\rho(M)$ denotes the spectral radius of the iteration matrix. Thus we want to find $m \in \mathbb{N}$ such that for all $k \geq m$, we have

$$\rho(M)^k \leq \varepsilon.$$

Applying logarithm we find that

$$m = \frac{\log(\varepsilon)}{\log(\rho(M))}.$$

Therefore let us compute the spectral radius for this case and each method. Consider the splitting $A = L + D + U$.

Richardson:
Here we have

$$M_R = I - A = \begin{pmatrix} -2 & 1 \\ 1 & -2 \end{pmatrix}.$$

Thus

$$0 = \det(M_R - \lambda I) = (2 + \lambda)^2 - 1$$

gives $\lambda \in \{-1, -3\}$, so that $\rho(M_R) = 3$. Therefore Richardson without relaxation does not converge!

Jacobi:
Here we have

$$M_J = I - D^{-1}A = I - \begin{pmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{3} \\ \frac{1}{3} & 0 \end{pmatrix}.$$

Thus

$$0 = \det(M_J - \lambda I) = \lambda^2 - \tfrac{1}{9}$$

gives $\lambda \in \{-\frac{1}{3}, \frac{1}{3}\}$, so that $\rho(M_J) = \frac{1}{3}$. Therefore Jacobi without relaxation does converge and

$$m = \frac{\log(\varepsilon)}{\log(\frac{1}{3})} \approx 13.$$

Gauß-Seidel:
Here we have

$$M_G = I - (D + L)^{-1}A$$

where

$$(D + L)^{-1} = \begin{pmatrix} \frac{1}{3} & 0 \\ \frac{1}{9} & \frac{1}{3} \end{pmatrix}$$

so that

$$M_G = I - \begin{pmatrix} \frac{1}{3} & 0 \\ \frac{1}{9} & \frac{1}{3} \end{pmatrix} \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{3} \\ 0 & \frac{1}{9} \end{pmatrix}.$$

Thus

$$0 = \det(M_G - \lambda I) = -\lambda(\tfrac{1}{9} - \lambda)$$

gives $\lambda \in \{0, \frac{1}{9}\}$, so that $\rho(M_G) = \frac{1}{9}$. Therefore Gauß-Seidel without relaxation does converge and

$$m = \frac{\log(\varepsilon)}{\log(\frac{1}{9})} \approx 7.$$

7 Least Squares Problems

Ex 115

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$. Assume you are given the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve?
2. Assume you are given the following data

z	-2	-1	0	1	2
y	-1	0	0	2	9

Solve the least squares problem

$$\min_{c_0,c_1\in\mathbb{R}} \sum_{i=1}^{5}(c_0+c_1z_i-y_i)^2.$$

Solution:

1. $\hat{x}$ solves the normal equation $(1P)$: $A^TA\hat{x} \overset{(1P)}{=} A^Ty$
2. In this case: $A = \begin{pmatrix} 1 & -2 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$ $(2P)$,

$$A^TA = \begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix} \quad (2P),$$

$$A^Ty = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} -1 \\ 0 \\ 0 \\ 2 \\ 9 \end{pmatrix} = \begin{pmatrix} 10 \\ 22 \end{pmatrix} \quad (2P)$$

Normal equation:

$$\begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix}\begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 10 \\ 22 \end{pmatrix} \quad \Leftrightarrow \quad c_0 = 2,\ c_1 = 2.2 \quad (1+1P)$$

$\Rightarrow \quad \hat{x} = \begin{pmatrix} 2 \\ 2.2 \end{pmatrix}$

Ex 116

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$. Assume you are given the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.
2. Assume you are given the following data

z	-3	-1	0	1	3
y	-3	-1,5	0	2,5	4

Solve the curve fitting problem

$$\min_{c_0,c_1\in\mathbb{R}} \sum_{i=1}^{5}(c_0+c_1z_i-y_i)^2,$$

i.e., determine the minimizing parameters c_0 and c_1.

Solution:

1. (1+1P): $\hat{x}$ solves the normal equation: $A^TA\hat{x} = A^Ty$
2. In this case:

$$(2P): \quad A = \begin{pmatrix} 1 & -3 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 3 \end{pmatrix},$$

$$(2P): \ A^TA = \begin{pmatrix} 5 & 0 \\ 0 & 20 \end{pmatrix} \quad ,$$

$$(2P): \ A^Ty = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -3 & -1 & 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} -3 \\ -1,5 \\ 0 \\ 2,5 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 25 \end{pmatrix}$$

Normal equation:

$$\begin{pmatrix} 5 & 0 \\ 0 & 20 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 25 \end{pmatrix} \quad \Leftrightarrow \quad c_0 = \frac{2}{5} = 0,4, \ c_1 = \frac{25}{20} = 1,25$$

$$\Rightarrow \quad (1+1P): \quad \hat{x} = \begin{pmatrix} 0,40 \\ 1,25 \end{pmatrix}, \quad c_0 = 0.40, \quad c_1 = 1.25$$

Ex 117

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Assume you are given the least squares problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.
2. Assume you are given the following data

z	-2	-1	0	1	2
y	3,5	2,5	1	0,5	-2,5

.

Solve the curve fitting problem

$$\min_{c_0, c_1 \in \mathbb{R}} \sum_{i=1}^{5} (c_0 + c_1 z_i - y_i)^2,$$

i.e., determine the minimizing parameters c_0 and c_1.

Solution:

1. $\hat{x}$ solves the normal $(1P)$ equation: $A^TA\hat{x} \overset{(1P)}{=} A^Ty$
2. In this case: $A = \begin{pmatrix} 1 & -3 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 3 \end{pmatrix} \quad (2P),$

$$A^TA = \begin{pmatrix} 5 & 0 \\ 0 & 20 \end{pmatrix}, \quad (2P)$$

$$A^Ty = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -3 & -1 & 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} 3,5 \\ 2,5 \\ 1 \\ 0,5 \\ -2,5 \end{pmatrix} = \begin{pmatrix} 5 \\ -20 \end{pmatrix} \quad (2P)$$

Normal equation:

$$\begin{pmatrix}5 & 0\\ 0 & 20\end{pmatrix}\begin{pmatrix}c_0\\ c_1\end{pmatrix} = \begin{pmatrix}5\\ -20\end{pmatrix} \quad \Leftrightarrow \quad c_0 = 1,\ c_1 = -1 \quad (1+1P)$$

$$\Rightarrow \quad \hat{x} = \begin{pmatrix}1\\ -1\end{pmatrix}$$

Ex 118

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$. Assume you are given the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.
2. Assume you are given the following data

z	-3	-1	0	1	3
y	-2,5	-1,5	0	2,5	4,5

   Solve the curve fitting problem

$$\min_{c_0,c_1\in\mathbb{R}} \sum_{i=1}^{5} (c_0 + c_1 z_i - y_i)^2.$$

Solution:

1. $\hat{x}$ solves the normal equation: $A^TA\hat{x} = A^Ty$
2. In this case: $A = \begin{pmatrix}1 & -2\\ 1 & -1\\ 1 & 0\\ 1 & 1\\ 1 & 2\end{pmatrix}$,

$$A^TA = \begin{pmatrix}5 & 0\\ 0 & 10\end{pmatrix} \quad ,$$

$$A^Ty = \begin{pmatrix}1 & 1 & 1 & 1 & 1\\ -2 & -1 & 0 & 1 & 2\end{pmatrix}\begin{pmatrix}-2,5\\ -1,5\\ 0\\ 2,5\\ 4,5\end{pmatrix} = \begin{pmatrix}3\\ 18\end{pmatrix}$$

Normal equation:

$$\begin{pmatrix}5 & 0\\ 0 & 10\end{pmatrix}\begin{pmatrix}c_0\\ c_1\end{pmatrix} = \begin{pmatrix}3\\ 18\end{pmatrix} \quad \Leftrightarrow \quad c_0 = \frac{3}{5} = 0,6,\ c_1 = 1,8$$

$$\Rightarrow \quad \hat{x} = \begin{pmatrix}0,6\\ 1,8\end{pmatrix}$$

Ex 119

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$. Assume you are given the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.
2. Assume you are given the following data

z	-3	-1	0	1	3
y	-2,5	-1,5	0	2,5	4,5

   Solve the curve fitting problem

$$\min_{c_0,c_1\in\mathbb{R}} \sum_{i=1}^{5} (c_0 + c_1 z_i - y_i)^2.$$

Solution:

1. $\hat{x}$ solves the normal equation: $A^T A\hat{x} = A^T y$
2. In this case: $A = \begin{pmatrix} 1 & -2 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix}$,

$$A^T A = \begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix} \quad ,$$

$$A^T y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} -2,5 \\ -1,5 \\ 0 \\ 2,5 \\ 4,5 \end{pmatrix} = \begin{pmatrix} 3 \\ 18 \end{pmatrix}$$

Normal equation:

$$\begin{pmatrix} 5 & 0 \\ 0 & 10 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 3 \\ 18 \end{pmatrix} \quad \Leftrightarrow \quad c_0 = \frac{3}{5} = 0,6, \; c_1 = 1,8$$

$$\Rightarrow \quad \hat{x} = \begin{pmatrix} 0,6 \\ 1,8 \end{pmatrix}$$

Ex 120

Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Assume you are given the least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve?
2. Assume you are given the following data

z	-2	-1	0	1	2
y	3	-1	0	1	4

.

Solve the curve fitting problem

$$\min_{c_0,c_1\in\mathbb{R}} \sum_{i=1}^{5} (c_0 + c_1 z_i^2 - y_i)^2,$$

i.e., determine the minimizing parameters c_0 and c_1.

Solution:

1. $\hat{x}$ solves the normal equation $(1P)$: $A^T A\hat{x} \overset{(1P)}{=} A^T y$
2. In this case: $A = \begin{pmatrix} 1 & 4 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 4 \end{pmatrix}$ $(2P)$,

$$A^T A = \begin{pmatrix} 5 & 10 \\ 10 & 34 \end{pmatrix} \quad (2P),$$

$$A^T y = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 4 & 1 & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} -1 \\ 0 \\ 0 \\ 2 \\ 9 \end{pmatrix} = \begin{pmatrix} 7 \\ 28 \end{pmatrix} \quad (2P)$$

Normal equation:

$$\begin{pmatrix} 5 & 10 \\ 10 & 34 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 7 \\ 28 \end{pmatrix}$$

Gaussian elimination yields

$$\begin{pmatrix} 5 & 10 \\ 0 & 14 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 7 \\ 14 \end{pmatrix}$$

$$\Leftrightarrow \quad c_0 = -\tfrac{2}{3}, \; c_1 = 1 \quad (1+1P)$$

$$\Rightarrow \quad \hat{x} = \begin{pmatrix} -\frac{2}{3} \\ 1 \end{pmatrix}$$

Ex 121

Least Squares

We are given a sample of size m of measurements $(z_i, y_i) \in \mathbb{R}^2$ for $i = 1, \dots, m$. Determine the minimizer c_0 of the problem

$$\min_{c_0 \in \mathbb{R}} \sum_{i=1}^{m} (c_0 - y_i)^2.$$

Solution:

Model: $f(x) = c_0$, with measurements (z_i, y_i), $i = 1, \dots, m$.

With the design matrix $A = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{m \times 1} = \mathbb{R}^m$ and $b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m$, we find

$$\min_{c_0 \in \mathbb{R}} \sum_{i=1}^{m} (c_0 - y_i)^2 = \min_{c_0 \in \mathbb{R}} \|A \cdot c_0 - b\|_2^2.$$

We have

$$A^T A = \sum_{i=1}^{m} 1 = m \quad \text{and} \quad A^T b = \sum_{i=1}^{m} y_i.$$

Thus by solving the normal equation we find

$$A^T A c_0 = A^T b \quad \Leftrightarrow \quad m \cdot c_0 = \sum_{i=1}^{m} y_i \quad \Leftrightarrow \quad c_0 = \frac{1}{m} \sum_{i=1}^{m} y_i.$$

With other words, the best constant fit in the least squares sense is the average of the data.

Ex 122

Minimum Norm Least Squares Solution: Example

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Assume you are given the least squares problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.
2. Assume you are given the following data

z	-1	1
y	2	-1

which you want to explain by a model $f : \mathbb{R} \to \mathbb{R}$ of the form

$$f_c(z) = c_0 + c_1 z + c_2 z^2.$$

Solve the least squares problem

$$\min_{c\in\mathbb{R}^3}\sum_{i=1}^{2}(f_c(z_i)-y_i)^2$$

to determine appropriate coefficients (c_0, c_1, c_2). If there are infinitely many solutions, pick the one with minimal norm.

Hint: Characterize the solution set $\hat{S}$ and derive a formula for the norm of a solution. Then use the fact, that the minimum (or maximum) of a quadratic function $p\colon \mathbb{R}\to\mathbb{R}$, $p(s) = a_0 + a_1 s + a_2 s^2$ can be found by setting the first derivative to zero, i.e., $0 = p'(s) = a_1 + 2a_2 s$.

Solution:

1. $\hat{x}$ solves the normal equation $A^T A\hat{x} = A^T b$.
2. We define

$$A := \begin{pmatrix} z_1^0 & z_1^1 & z_1^2 \\ z_2^0 & z_2^1 & z_2^2 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad b := \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix},$$
$$x = (c_0, c_1, c_2).$$

Then:

$$\min_{c\in\mathbb{R}^3}\sum_{i=1}^{2}(f_c(z_i)-y_i)^2 \quad = \quad \min_{x\in\mathbb{R}^3}\|Ax-b\|_2^2.$$

Thus we can equivalently solve the normal equation:

$$A^TAx = A^Tb \Leftrightarrow \underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix}}_{=\begin{pmatrix} 2 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 2 \end{pmatrix}} x = \underbrace{\begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 2 \\ -1 \end{pmatrix}}_{=\begin{pmatrix} 1 \\ -3 \\ 1 \end{pmatrix}}$$

$$\Leftrightarrow \begin{pmatrix} 2 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -3 \\ 1 \end{pmatrix} \begin{matrix} \text{(I)} \\ \text{(II)} \\ \text{(III)} \end{matrix}$$

$$\begin{matrix} \text{(I)}' = \frac{1}{2}\text{(I)} \\ \text{(II)}' = \frac{1}{2}\text{(II)} \\ \text{(III)}' = \text{(III)} - \text{(I)} \end{matrix} \quad \Leftrightarrow \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -\frac{3}{2} \\ 0 \end{pmatrix}$$

$$\text{(I)}' \Rightarrow x_1 + x_3 = \frac{1}{2} \Leftrightarrow x_3 = \frac{1}{2} - x_1$$

$$\text{(II)}' \Rightarrow x_2 = -\frac{3}{2}.$$

Thus the solution set is given by

$$\hat{S} := \{x\in\mathbb{R}^3 : A^TAx = A^Tb\} = \left\{\begin{pmatrix} s \\ -\frac{3}{2} \\ \frac{1}{2} - s \end{pmatrix} : s\in\mathbb{R}\right\}.$$

We now determine the solution with minimal norm:

a) For an arbitrary solution $x_s := \begin{pmatrix} s \\ -\frac{3}{2} \\ \frac{1}{2} - s \end{pmatrix}$ for some $s\in\mathbb{R}$ we find:

$$p(s) := \|x_s\|_2^2 = s^2 + \frac{9}{4} + \left(\frac{1}{2} - s\right)^2 = s^2 + \frac{9}{4} + \frac{1}{4} - s + s^2 = \frac{5}{2} - s + 2s^2.$$

b) We determine the parameter $s\in\mathbb{R}$ for which $\|x_s\|_2^2$ is minimal:

$$0 \overset{!}{=} p'(s) = -1 + 4s \quad \Leftrightarrow \quad s = \frac{1}{4}.$$

c) The minimum norm solution is therefore given by

$$x_{s=\frac{1}{4}} = \begin{pmatrix} \frac{1}{4} \\ -\frac{3}{2} \\ \frac{1}{2} - \frac{1}{4} \end{pmatrix} = \begin{pmatrix} 0.25 \\ -1.5 \\ 0.25 \end{pmatrix}.$$

The model function corresponding to the minimum norm solution is then given by

$$f_c(z) = 0.25 - 1.5z + 0.25z^2.$$

Ex 123 Least Squares Problems

Orthogonal Projection

Prove the following statement: Let $V \subset \mathbb{R}^m$ be a linear subspace and $b \in \mathbb{R}^m$. Then

$$\widehat{z} = \arg\min_{z \in V} \|z - b\|_2^2 \quad \Leftrightarrow \quad \widehat{z} - b \in V^\perp := \{w \in \mathbb{R}^n : w^\top v = 0 \ \ \forall v \in V\}.$$

Hint: You can use: For all $x, y \in \mathbb{R}^m$: $\|x + y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 \ \Leftrightarrow \ x^\top y = 0$.

Solution:

We use the hint with $x = \widehat{z} - b$ and $y := z - \widehat{z}$ for some $z \in V$ (note that $(z - \widehat{z}) \in V \ \forall z \in V$, since $\widehat{z} \in V$ and V subspace). More precisely, for a $\widehat{z} \in V$ we find

$$\begin{aligned} \widehat{z} - b \in V^\perp \quad & \Leftrightarrow \forall z \in V : (\widehat{z} - b)^\top z = 0 \\ & \Leftrightarrow \forall z \in V : (\widehat{z} - b)^\top (z - \widehat{z}) = 0 \\ & \Leftrightarrow \forall z \in V : \|z - b\|_2^2 = \|\widehat{z} - b\|_2^2 + \|\widehat{z} - z\|_2^2 \\ & \Leftrightarrow \forall z \in V : \|\widehat{z} - b\|_2^2 \leq \|z - b\|_2^2 \\ & \Leftrightarrow \widehat{z} = \arg\min_{z \in V} \|z - b\|_2^2. \end{aligned}$$

Ex 124 Linear Algebra, Singular Values, Python

Least Squares and *Total* Least Squares (Principal Components)

1. Use the function `numpy.random.multivariate_normal()` to create samples $(z_i, y_i) \in \mathbb{R}^2$ for $i = 1, \dots, 100$ from a 2-dimensional multivariate normal distribution with mean $\mu := (0, 0)$ and covariance

$$\Sigma := \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}.$$

2. Implement a function or find a Scipy function to solve the least squares problem

$$\min_{c \in \mathbb{R}} \sum_{i=1}^{100} (cz_i - y_i)^2$$

and plot the model $f(z) = cz$ and the data points into one plot.

3. Now consider the 2×100 matrix $A = [v_1, \cdots, v_n]$ with columns $v_i := (z_i, y_i)^\top \in \mathbb{R}^2$ and compute an SVD $A = U\Sigma V^\top$ of it. Draw the lines through the column vectors of U into the same plot. These are the principal components that explain most of the variance.

Solution:

```python
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt



if __name__ == "__main__":
    # generate random sample (multivariate normal distribution)
    mean = [0, 0]; cov = [[1, .7], [.7, 1]]
    Nrandom = 100
    x = np.random.multivariate_normal(mean, cov, Nrandom).T

    # PCA: compute svd
    U, sigma, V = linalg.svd(x)

    # least squares
    c, residuals, rank, singular_val = np.linalg.lstsq(x[0][np.newaxis].T,
                                                         x[1], rcond=None)

    # PLOT
    fig, ax = plt.subplots()

    # plot random sample as dots
    plt.plot(x[0], x[1], "o", alpha=.6, zorder=1)

    # plot least squares fit: f(z) = t * c
    t = np.linspace(-3, 3, 20)
    plt.plot(t, t * c, zorder=4)
    print("slope of the least squares fit:", c)

    # plot first principal component = line spanned by first column of U
    # each point (x,y) on this line is orthogonal (-U[0,1],U[0,0])
    plt.plot(t, t * (U[0, 1] / U[0, 0]), zorder=4, color='red')
    print("slope of PCA line:", U[0, 1] / U[0, 0])

    # plot styling
    plt.grid(alpha = 0.25)
    plt.xlim(xmin=-3, xmax=3)
    plt.ylim(ymin=-3, ymax=3)
    ax.set_aspect('equal')
    plt.legend(["Samples", "Linear Least Squares", "First Principal Component"])
    plt.show()
```

Ex 125 Least Squares Problems, Python

Ridge Regression and the Minimum Norm Solution

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Assume you are given the regularized least squares problem

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2 + \tfrac{\delta}{2}\|x\|_2^2.$$

1. Which equation does the solution x_δ of the above least squares problem solve?

2. Assume you are given the following data

z	-1	1
y	2	-1

which you want to explain by a model $f : \mathbb{R} \to \mathbb{R}$ of the form

$$f_c(z) = c_1 + c_2 z + c_3 z^2.$$

Implement a program to solve the regularized least squares problem

$$\min_{c\in\mathbb{R}^3} \sum_{i=1}^{2} (f_c(z_i) - y_i)^2 + \tfrac{\delta}{2} \sum_{j=1}^{3} c_j^2$$

to determine appropriate coefficients $x_\delta = (c_1^\delta, c_2^\delta, c_3^\delta)$ for *multiple* $\delta \in (0,1)$. Why is regularization an appropriate approach here?

3. Find a routine to compute the minimum norm least squares solution x^+ and compare it to your solutions x_δ. What do you observe for small δ?

4. Plot the measurements and the fitting polynomial corresponding to x_δ and x^+ into one figure. What do you observe for small δ?

   Hint: Use

   - scipy.linalg.solve to solve a linear system and set the correct flag that informs the function about the positivity of the matrix (see documentation),
   - numpy.linspace to create an array of multiple $\delta \in (0,1)$,
   - the plot routines from previous exercises if you want.

Solution:

```python
import numpy as np
import scipy.linalg as linalg
import matplotlib.pyplot as plt

if __name__ == "__main__":

    # given data
    data = np.array([[-1., 1],
                     [2, -1]])

    # Assembly
    # design matrix
    p = [0,1,2]
    z_i = data[0,:]
    z_i = z_i[np.newaxis].T
    A = z_i**p

    # right-hand side
    b = data[1,:]

    # minimum norm least squares solution
    x_0 = linalg.lstsq(A,b)[0] # note that lstsq uses the svd

    # create an array of delta's in (0,1)
    num_delta = 3
    Delta = np.linspace(0.0001,1, num_delta, endpoint=False)

    # loop over delta's from large to small
    print("del \t x_del  \t\t\t\t\t    ||x_del-x0|| ||x|| \t  ||Ax-b||")
    for delta in Delta[::-1]:

        # solve regularized least squares problem (reg. normal equation)
        A_delta = A.T@A + delta * np.eye(len(A.T@A))
        x_delta = linalg.solve(A_delta, A.T@b, assume_a='pos')

        print(np.round(delta, 4), "\t",
              np.round(x_delta, 6), "\t",
              np.round(linalg.norm(x_delta-x_0), 6), "\t",
              np.round(linalg.norm(x_delta), 6), "\t ",
              np.round(linalg.norm(A@x_delta - b), 6))

        # Plot measurements and fitting curve
        plt.figure()
        plt.title("Polynomial degree = "+str(max(p))+", delta = "+str(np.round(delta,4)))
        # measurements
        plt.plot(z_i, b, 'ro')
        # curve
        X = np.linspace(-2,2, 50)
        plt.plot(X, x_delta[0] + x_delta[1]*X + x_delta[2]*(X**2), 'b')
        plt.plot(X, x_0[0] + x_0[1]*X + x_0[2]*(X**2), 'cyan', linestyle = "--")
        plt.legend(["Samples", "Regularized Sol", "MinNorm Sol"])
        plt.show()
```

Ex 126 Linear Algebra

Projections and Least Squares
Let $a, b \in \mathbb{R}^n \setminus \{0\}$ be two nonzero vectors. Consider the 1-dimensional optimization task

$$\min_{c\in\mathbb{R}} \frac{1}{2}\|ca - b\|_2^2 =: f(c),$$

where $\|x\|_2 := \left(\sum_{i=1}^n x_i^2\right)^{\frac{1}{2}}$ denotes the Euclidean norm of a vector $x \in \mathbb{R}^n$.

1. Determine the parameter $c \in \mathbb{R}$ which minimizes f.
   Hint: As in high-school, compute the derivative f' of f with respect to c and solve the equation $f'(c) = 0$.
2. Compare your results to the projection of b onto a, i.e., $\text{proj}_a(b) := \frac{a^\top b}{\|a\|_2}\frac{a}{\|a\|_2}$.

Solution:

First we note that

$$f(c) = \frac{1}{2}\|ca - b\|_2^2 = \frac{1}{2}\left(c^2 a^\top a - 2ca^\top b - b^\top b\right)$$

Thus, for the derivative with respect to the scalar c, we find

$$f'(c) = ca^T a - a^T b.$$

Since $a \neq 0$ and therefore $a^\top a \neq 0$, we find

$$f'(\hat{c}) = 0 \quad \Leftrightarrow \quad \hat{c} = \frac{a^T b}{a^T a}.$$

By convexity of f we can conclude that $\hat{c}$ is a minimizer (you will learn this in the course "Numerical Optimization").

Remark: We will later identity the equation $ca^T a - a^T b = 0$ as the **normal equation**. The vector on the line span(a) closest to b in terms of the Euclidean norm is given by

$$\hat{c}a = \frac{a^T b}{a^T a}a = \frac{a^T b}{\|a\|_2}\frac{a}{\|a\|_2} = \text{proj}_a(b).$$

Ex 127 Least Squares Problems

Properties of $A^T A + \delta I$
Let $A \in \mathbb{R}^{m\times n}$ be any matrix.

1. Please show that $A^T A + \delta I$ is symmetric and positive definite for all $\delta > 0$.
2. Why is $A^T A + \delta I$ invertible for all $\delta > 0$?

Solution:

Symmetry (Recall: A matrix B is called symmetric, if $B^T = B$ holds). Here we find

$$(A^T A + \delta I)^T \quad = \quad (A^T A)^T + \delta I^T \quad = \quad A^T A + \delta I.$$

Positivity (Recall: A matrix B is called positive definite, if $x^T Bx > 0$ holds $\forall x \in \mathbb{R}^n \setminus \{0\}$). Here we find

$$x^T(A^T A + \delta I)x \quad = \quad \underbrace{x^T(A^T A)x}_{=\|Ax\|_2^2 \geq 0} + \underbrace{\delta}_{>0}\underbrace{x^T x}_{>0} \quad > \quad 0 \quad \text{for all } x \in \mathbb{R}^n \setminus \{0\}.$$

Thus, since $A^T A + \delta I$ is symmetric and positive definite for $\delta > 0$, it is invertible (see previous exercises).

Ex 128

8 Vector Spaces

Ex 129 Linear Algebra

Proof of Lemma 11.15
Let $f \in \mathrm{Hom}_{\mathbb{F}}(\mathbb{F}^n, \mathbb{F}^m)$ with representing matrix $A \in \mathbb{F}^{m\times n}$. Please show:

1. The sets $\ker(f)$ and $\mathrm{Im}(f)$ are subspaces of $\mathbb{F}^n$ and $\mathbb{F}^m$, respectively.
2. $\mathrm{rank}(A) = \dim(\mathrm{Im}(f))$.
3. $\ker(f) = \{0\} \quad \Leftrightarrow \quad f$ is injective.

Solution:

Let $f : \mathbb{F}^n \to \mathbb{F}^m$ be linear with matrix representation $A \in \mathbb{F}^{m\times n}$.

1. Show:
   a) $\ker(f) \subset \mathbb{F}^n$ subspace
   b) $\mathrm{Im}(f) \subset \mathbb{F}^m$ subspace

   Proof:
   a) i. $0 \in \ker(f)$
      ii. $\lambda \in \mathbb{F},\ u, v \in \ker(f)$, then $f(\lambda u + v) = \lambda \underbrace{f(u)}_{=0} + \underbrace{f(v)}_{=0} = 0$
      $\Rightarrow \quad \lambda u + v \in \ker(f)$
   b) i. $A_0 = 0 \in \mathrm{Im}(f)$
      ii.
$$\begin{aligned} & \lambda \in \mathbb{F},\ w_1, w_2 \in \mathrm{Im}(f) \subset W \\ \Rightarrow\ & \exists v_1, v_2 \in V:\ w_1 = f(v_1),\ w_2 = f(v_2) \\ \Rightarrow\ & \lambda w_1 + w_2 = \lambda f(v_1) + f(v_2) = f(\lambda v_1 + v_2) \\ \Rightarrow\ & \lambda w_1 + w_2 \in \mathrm{Im}(A) \end{aligned}$$

2. Show: $\mathrm{rank}(A) = \dim(\mathrm{Im}(f))$
   Proof:
   Recall:
   - $\mathrm{rank}(A_f) = |\{\sigma \neq 0 :\ \sigma \text{ singular value of } A\}|$
     $A = U\Sigma V^T,\ U \in \mathbb{R}^{m\times m},\ V \in \mathbb{R}^{n\times n}$ orthogonal, $\Sigma = \begin{pmatrix} \ddots & 0 \\ 0 & 0 \end{pmatrix}$
   - $\dim(V) :=$ length of a basis

   Let $\mathrm{rank}(A_f) =: k$, then
$$\begin{aligned} \mathrm{Im}(f) &= \mathrm{Im}(A_f) = \{A_f x :\ x \in \mathbb{F}^n\} = \{U \underbrace{\Sigma V^T x}_{=:\lambda = (\underbrace{\tilde\lambda}_{\in \mathbb{R}^k}, \underbrace{0, \dots, 0}_{m-k})} :\ x \in \mathbb{F}^n\} \\ &= \{U_k \tilde\lambda :\ \tilde\lambda \in \mathbb{F}^k\} \\ \Rightarrow \mathrm{Im}(f) &= \mathrm{span}(u_1, \dots, u_k) \text{ and } u_1, \dots, u_k \text{ linearly independent (even orthogonal)} \\ \Rightarrow \dim(\mathrm{Im}(f)) &= k = \mathrm{rank}(A_f) \end{aligned}$$

3. Show: $\ker(f) = \{0\} \quad \Leftrightarrow \quad f$ injective Proof:
$$\begin{aligned} \ker(f) = \{0\} &\Leftrightarrow \{f(x) = 0 \Rightarrow x = 0\} \\ &\Leftrightarrow \forall x, y \in \mathbb{F}^n:\ f(x) - f(y) = f(x-y) = 0 \Rightarrow x - y = 0 \\ &\Leftrightarrow f \text{ injective} \end{aligned}$$

□

Vector Space of Polynomials

Let $n \in \mathbb{N}$ and $P_n(\mathbb{R})$ be the set of all polynomials of degree $\leq n$ on $\mathbb{R}$, i.e., the set $P_n(\mathbb{R})$ contains all functions $p : \mathbb{R} \to \mathbb{R}$ of the form

$$p(x) = \sum_{k=0}^{n} \alpha_k x^k$$

for some $\alpha_0, \alpha_1, \dots, \alpha_n \in \mathbb{R}$. We define a summation and scalar multiplication:

$$\begin{aligned} +\colon P_n(\mathbb{R}) \times P_n(\mathbb{R}) \to P_n(\mathbb{R}),& \quad (p+q)(x) := p(x) + q(x), \\ \cdot\colon \mathbb{R} \times P_n(\mathbb{R}) \to P_n(\mathbb{R}),& \quad (r \cdot p)(x) := r \cdot p(x). \end{aligned}$$

1. **VR axioms:** Please show that $P_n(\mathbb{R})$ together with the above defined summation and scalar multiplication forms a vector space.
2. Let $k < m \in \mathbb{N}$. Compute
$$\lim_{x\to\infty} \frac{x^k}{x^m}, \quad \text{and} \quad \lim_{x\to\infty} \frac{\sum_{k=0}^{m-1} \alpha_k x^k}{x^m}$$
for arbitrary $\alpha_0, \dots, \alpha_m \in \mathbb{R}$.
3. **Monomials form a basis:** Please show that the set $B := \{q_0, \dots, q_n\}$ with
$$q_k : \mathbb{R} \to \mathbb{R}, \quad x \mapsto x^k,$$
is a basis of $P_n(\mathbb{R})$. What is the dimension of the vector space $P_n(\mathbb{R})$? *Hint: Part 2. basically provides the proof of linear independence.*
4. **Derivative as linear operator:** Show that the operator $\mathcal{D} : P_n(\mathbb{R}) \to P_{n-1}(\mathbb{R}), \quad p \mapsto p'$, which maps a polynomial to its first derivative, is an $\mathbb{R}$-linear function.
Hint: You can use that for $p(x) = \sum_{k=0}^{n} \alpha_k x^k$ we have $Dp(x) = p'(x) = \sum_{k=0}^{n-1} \alpha_{k+1}(k+1)x^k$.
5. **Matrix representation of the derivative:** Derive the matrix representation of $\mathcal{D}$ with respect to the bases of monomials, i.e., derive the matrix $\mathcal{M}_{\{q_0,\dots,q_{n-1}\}}^{\{q_0,\dots,q_n\}}(\mathcal{D})$.

Solution:

1. **VR1:** Show, that $(P_n(\mathbb{R}), +)$ is an abelian group.
Let $p(x) = \sum_{k=0}^{n} \alpha_k x^k$, $q(x) = \sum_{k=0}^{n} \beta_k x^k$ and $w(x) = \sum_{k=0}^{n} \gamma_k x^k$ be in $P_n(\mathbb{R})$.
   (i) Associativity:
$$((p+q)+w)(x) = \sum [(\alpha_k + \beta_k) + \gamma_k] x^k = \sum [\alpha_k + (\beta_k + \gamma_k)] x^k = (p + (q+w))(x)$$
   (ii) Neutral Element:
$0 := \sum_{k=0}^{n} 0 \cdot x^k$, then $\forall\ p \in P_n(\mathbb{R})$:
$$(0+p)(x) = \sum (0 + \alpha_k) x^k = p(x)$$
   (iii) Inverse element:
For $p(x) = \sum \alpha_k x^k$ define $-p(x) := \sum (-\alpha_k) x^k$,
$$\Rightarrow \quad (p + (-p))(x) = 0.$$
   (iv) Commutativity:
$$(p+q)(x) = \sum \underbrace{(\alpha_k + \beta_k)}_{=\beta_k + \alpha_k} x^k = (q+p)(x)$$

   VR2: Consistency properties: Let $r, s \in \mathbb{R}$.
   (i) $((r+s)p)(x) = \sum \underbrace{(r+s)\alpha_k}_{=r\alpha_k + s\alpha_k} x^k = (rp)(x) + (sp)(x)$

(ii)-(iv) ✓

2. Let $k < m \in \mathbb{N}$. Then

$$\frac{x^k}{x^m} = x^{k-m} = \frac{1}{x^{m-k}} \overset{x\to+\infty}{\longrightarrow} 0$$
$$\Rightarrow \quad \forall \alpha_0, \dots, \alpha_m : \quad \frac{\sum_{k=0}^{m-1} \alpha_k x^k}{x^m} = \sum_{k=0}^{m-1} \alpha^k \underbrace{\left(\frac{x^k}{x^m}\right)}_{\to 0} \overset{x\to+\infty}{\longrightarrow} 0.$$

In particular we can conclude that

$$\forall \alpha_0, \dots, \alpha_m : \quad \sum_{k=0}^{m-1} \alpha^k x^k \neq x^m$$

because otherwise limit $\equiv 1$.

3. a) Linear independence by 2.
Asssume $\exists \alpha_0, \dots, \alpha_n$ ($m :=$ max$\{k : \alpha_k \neq 0\}$) not all zero with $\sum_{k=0}^{m} \alpha_k q_k = 0$

$$\Rightarrow \quad \sum_{k=0}^{m} \alpha_k q_k = \sum_{k=0}^{m-1} \alpha_k q_k + \alpha_m q_m = 0$$
$$\Rightarrow \quad \sum_{k=0}^{m} \alpha_k x^k = (-\alpha_m) x^m$$

contradiction to 2.

b)

$$\text{span}\{q_0, \dots, q_n\} = P_n(\mathbb{R}) \quad \text{by definition}$$
$$\Rightarrow \quad \dim(P_n(\mathbb{R})) = n+1$$

4. Let $p = \sum_{k=0}^{n} \alpha_k q_k$ and $w = \sum_{k=0}^{n} \beta_k q_k$ and $\lambda \in \mathbb{R}$, then:

$$\begin{aligned} \mathsf{D}(\lambda p + w)(x) &= \left(\sum_{k=0}^{n} (\lambda \alpha_k + \beta_k) x^k\right)' = \sum_{k=0}^{n} (\lambda \alpha_k + \beta_k) k x^{k-1} \\ &= \lambda \sum_{k=0}^{n} \alpha_k k x^{k-1} + \sum_{k=0}^{n} \beta_k k x^{k-1} = \lambda \mathsf{D}(p)(x) + \mathsf{D}(q)(x) \end{aligned}$$

5. For $j = 0$ we have

$$\mathcal{D} q_0 = 0$$

and for $1 \leq j \leq n$ we have

$$\mathcal{D} q_j = j q_{j-1}.$$

These coordinates in the basis q_j are simply scaled unit vectors, thus we obtain the $n \times (n+1)$ matrix

$$\mathcal{M}^{\{q_0,\dots,q_n\}}_{\{q_0,\dots,q_{n-1}\}}(\mathcal{D}) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & \ddots & n \end{pmatrix}.$$

Additional remark: Let us consider the coordinate mapping

$$\Phi : \ P_n(\mathbb{R}) \to \mathbb{R}^{n+1}, \ p = \sum_{k=0}^{n} \alpha_k q_k \mapsto (\alpha_0, \dots, \alpha_n)^T = (\pi_0(p), \dots, \pi_n(p))$$

This Φ is linear, since π_j are linear (see lecture) and bijective since $\{q_0, \dots, q_n\}$ basis, $\Phi^{-1} : \mathbb{R}^{n+1} \to P_n(\mathbb{R})$, $(\alpha_0, \dots, \alpha_n)^T \mapsto p = \sum \alpha_k q_k$.
We get the following commutative diagram

$$\begin{tikzcd}
(\alpha_o, \dots, \alpha_n)^T & \begin{matrix} \mathbb{R}^{n+1} \\ \{e_1, \dots, e_{n+1}\} \end{matrix} \arrow[r, "\mathcal{M}(\mathcal{D})"] & \begin{matrix} \mathbb{R}^{n} \\ \{e_1, \dots, e_{n}\} \end{matrix} \arrow[d, "\Phi^{-1}"'] & (\alpha_1, 2\alpha_2, 3\alpha_3, \dots, n\alpha_n) =: \beta \\
p = \sum_{k=0}^{n} \alpha_k q_k & \begin{matrix} P_n(\mathbb{R}) \\ \{q_0, \dots, q_{n+1}\} \end{matrix} \arrow[u, "\Phi"] \arrow[r, "\mathcal{D}"] & \begin{matrix} P_{n-1}(\mathbb{R}) \\ \{q_0, \dots, q_{n}\} \end{matrix} & p' = \sum_{k=0}^{n-1} \beta_k q_k
\end{tikzcd}$$

9 Calculus

Ex 131 Derivatives

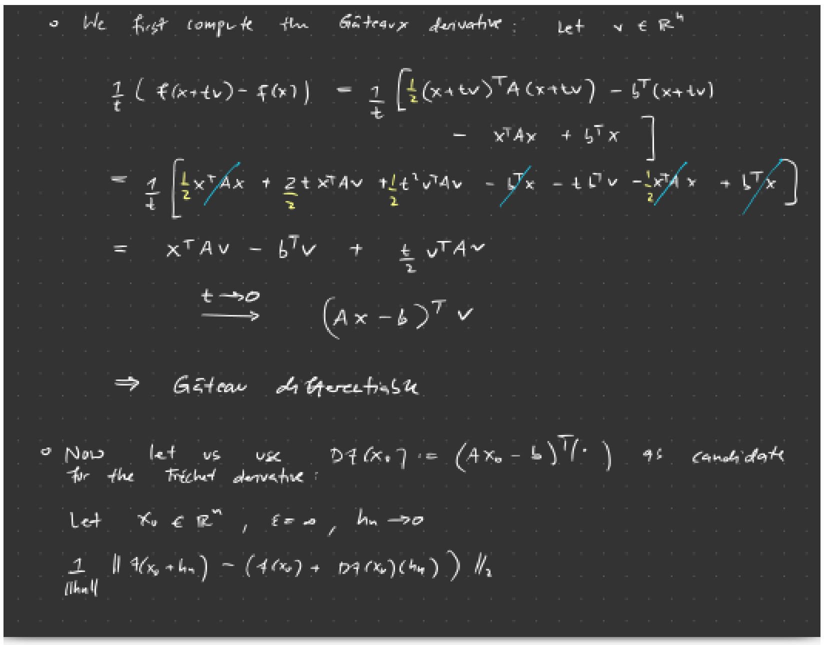
Derivatives
Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix and $b \in \mathbb{R}^n$ a vector . Show that the function

$$f : \mathbb{R}^n \to \mathbb{R}, x \mapsto \frac{1}{2}x^T Ax - b^T x$$

is Frechét differentiable and determine the gradient $\nabla f(x)$ of f at a point $x \in \mathbb{R}^n$.

Hint: Compute the directional (Gâteau) derivative and use the resulting expression as a candidate for the Fréchet derivative.

Solution:



- We first compute the Gâteaux derivative: Let $v \in \mathbb{R}^n$

$$\frac{1}{t}\left(f(x+tv) - f(x)\right) = \frac{1}{t}\Big[\tfrac{1}{2}(x+tv)^TA(x+tv) - b^T(x+tv) - x^TAx + b^Tx\Big]$$

$$= \frac{1}{t}\Big[\cancel{\tfrac{1}{2}x^TAx} + \tfrac{2}{2}t\,x^TAv + \tfrac{1}{2}t^2v^TAv - \cancel{b^Tx} - t\,b^Tv - \cancel{\tfrac{1}{2}x^TAx} + \cancel{b^Tx}\Big]$$

$$= x^TAv - b^Tv + \frac{t}{2}v^TAv$$

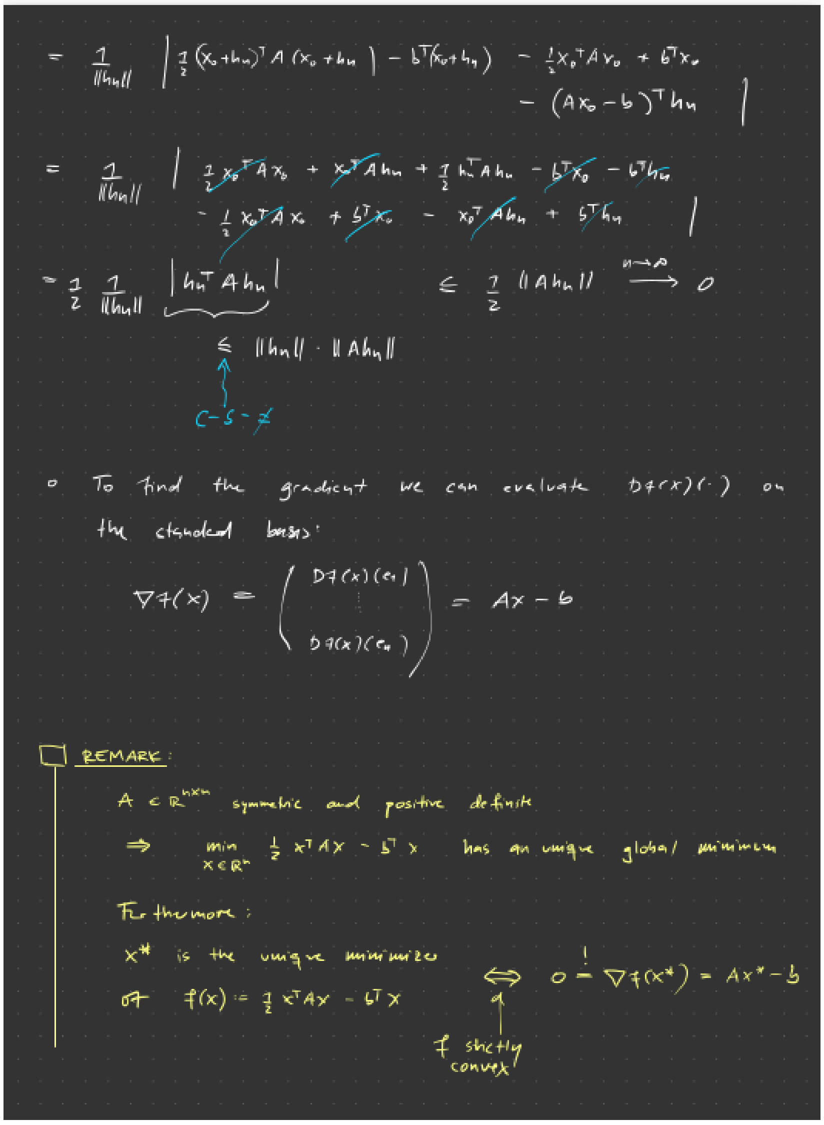
$$\xrightarrow{t \to 0} (Ax-b)^T v$$

$\Rightarrow$ Gâteau differentiable

- Now let us use $Df(x_0) := (Ax_0 - b)^T(\cdot)$ as candidate for the Fréchet derivative:

Let $x_0 \in \mathbb{R}^n$, $\varepsilon > 0$, $h_n \to 0$

$$\frac{1}{\|h_n\|}\left\| f(x_0+h_n) - \left(f(x_0) + Df(x_0)(h_n)\right)\right\|_2$$



$$= \frac{1}{\|h_n\|}\Big| \tfrac{1}{2}(x_0+h_n)^TA(x_0+h_n) - b^T(x_0+h_n) - \tfrac{1}{2}x_0^TAx_0 + b^Tx_0 - (Ax_0-b)^Th_n \Big|$$

$$= \frac{1}{\|h_n\|}\Big| \cancel{\tfrac{1}{2}x_0^TAx_0} + \cancel{x_0^TAh_n} + \tfrac{1}{2}h_n^TAh_n - \cancel{b^Tx_0} - \cancel{b^Th_n} - \cancel{\tfrac{1}{2}x_0^TAx_0} + \cancel{b^Tx_0} - \cancel{x_0^TAh_n} + \cancel{b^Th_n} \Big|$$

$$= \frac{1}{2}\frac{1}{\|h_n\|}\underbrace{\left|h_n^TAh_n\right|}_{\leq \|h_n\| \cdot \|Ah_n\|} \leq \frac{1}{2}\|Ah_n\| \xrightarrow{n\to\infty} 0$$

C-S-$\cancel{I}$

- To find the gradient we can evaluate $Df(x)(\cdot)$ on the standard basis:

$$\nabla f(x) = \begin{pmatrix} Df(x)(e_1) \\ \vdots \\ Df(x)(e_n) \end{pmatrix} = Ax - b$$

☐ REMARK:

$A \in \mathbb{R}^{n\times n}$ symmetric and positive definite

$\Rightarrow \quad \min\limits_{x\in\mathbb{R}^n} \frac{1}{2}x^TAx - b^Tx$ has an unique global minimum

Furthermore:

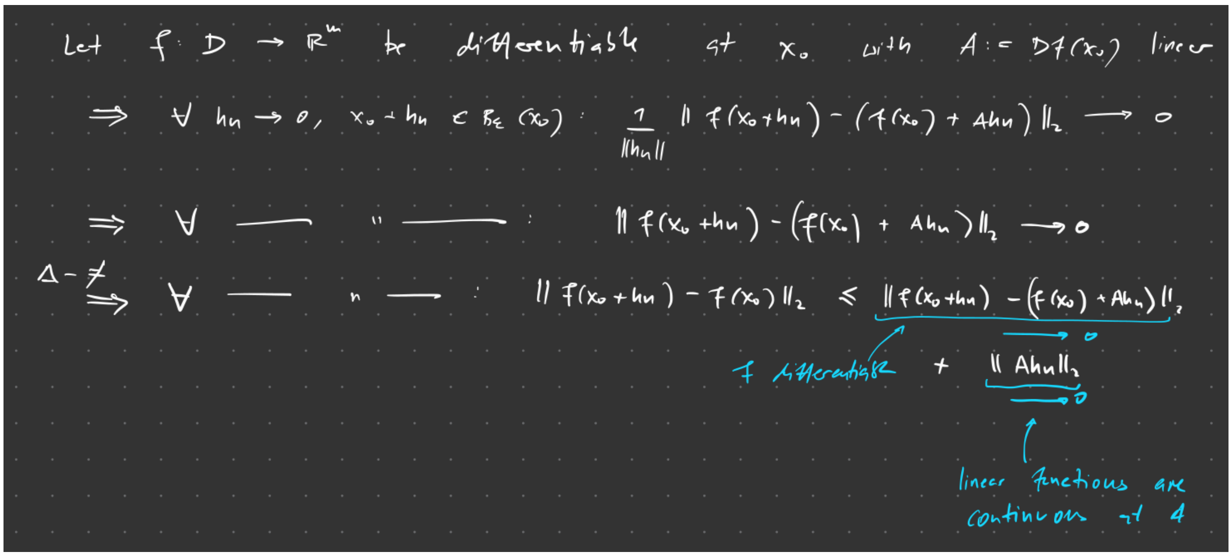
x^* is the unique minimizer of $f(x) := \frac{1}{2}x^TAx - b^Tx$ $\overset{f \text{ strictly convex}}{\Leftrightarrow}$ $0 \overset{!}{=} \nabla f(x^*) = Ax^* - b$

Ex 132 Derivatives

Differentiable implies Continuous

Let $D \subset \mathbb{R}^n$, $x_0 \in D$ with $B_\varepsilon(x_0) \subset D$ for some $\varepsilon > 0$. Let $f\colon D \to \mathbb{R}$ be (Frechét-) differentiable at x_0. Show that f is continuous at x_0.

Solution:



Let $f: D \to \mathbb{R}^m$ be differentiable at x_0 with $A := Df(x_0)$ linear

$\Rightarrow \forall h_n \to 0,\ x_0 + h_n \in B_\varepsilon(x_0)$: $\frac{1}{\|h_n\|} \| f(x_0 + h_n) - (f(x_0) + Ah_n) \|_2 \longrightarrow 0$

$\Rightarrow \forall$ —— " —— : $\| f(x_0 + h_n) - (f(x_0) + Ah_n) \|_2 \longrightarrow 0$

Δ-$\neq$ $\Rightarrow \forall$ —— " —— : $\| f(x_0 + h_n) - f(x_0) \|_2 \leq \underbrace{\| f(x_0 + h_n) - (f(x_0) + Ah_n) \|_2}_{\longrightarrow 0} + \underbrace{\| Ah_n \|_2}_{\longrightarrow 0}$

f differentiable

linear functions are continuous at 4

Ex 133

Heron's Method as Newton's Method

Consider the function $f\colon \mathbb{R} \to \mathbb{R},\ f(x) := x^2 - a$ for some nonnegative number $a \geq 0$. Apply Newton's method to the nonlinear system $f(x) = 0$ (root finding problem). Compare the resulting iterative scheme to Heron's algorithm from earlier sheets.

Solution:

Idea of Newton's method: [see result in the script]

Let $f: \mathbb{R}^n \to \mathbb{R}^n$ be „sufficiently smooth". We want to find a root $\hat{x} \in \mathbb{R}^n$ of f such that $f(\hat{x}) = 0$.

We try to do this iteratively. Therefore let $x^k \in \mathbb{R}^n$ be our approximation to $\hat{x}$ at step k.

How to choose a step Δx^k such that the iteration

$$x^{k+1} := x^k + \Delta x^k$$

converges to $\hat{x}$?

By Taylor approximation:

$$f(x^{k+1}) \approx \underset{\overset{!}{=}\ 0}{f(x^k) + J_f(x^k)\Delta x^k} \qquad \text{for } \|\Delta x^k\| \text{ small}$$

suggest to take

$$\Delta x^k := -J_f^{-1}(x^k)\,\Delta x^k$$

such that all in all:

$$x^{k+1} := x^k + \Delta x^k = x^k - J_f^{-1}(x^k)\, f(x^k)$$

Example: $f: \mathbb{R}^1 \to \mathbb{R}^1$, $f(x) := x^2 - a$, $a > 0$

$$f'(x) = 2x, \quad (f'(x))^{-1} = \frac{1}{2x}, \quad x \neq 0$$

$$\Rightarrow \quad x_{k+1} = x_k - (f'(x_k))^{-1} f(x_k)$$
$$= x_k - \frac{1}{2x_k}(x_k^2 - a)$$
$$= \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

Ex 134 Linear Systems, Splitting Methods, Python

Appetizer: Gradient, Steepest Descent and Conjugate Gradient Method

Background: Let $A \in \mathbb{R}^{n\times n}$ be symmetric and positive definite (spd). Then A is in particular invertible, so that the linear system $Ax = b$ has a unique solution $x^* \in \mathbb{R}^n$ for all $b \in \mathbb{R}^n$. Let us relate this linear system to an optimization problem. For this purpose we define for a fixed spd matrix A and fixed right-hand side b the function

$$f := f_{A,b} : \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto \tfrac{1}{2}x^T Ax - b^T x.$$

Then one can show the equivalence

$$Ax^* = b \qquad \Longleftrightarrow \qquad x^* = \arg\min_{x\in\mathbb{R}^n} f(x).$$

In words, x^* solves the linear system on the left-hand side if and only if x^* is the unique minimizer of the functional f. In fact, you will learn in the next semester that the condition $Ax^* = b$ is the necessary first-order optimality condition:

$$0 = \nabla f(x) = Ax - b.$$

Due to the convexity of f this condition is also sufficient. Consequently, solving linear systems which involve spd matrices is equivalent to solving the associated optimization problem above, i.e., minimizing the function $f(x) = \frac{1}{2}x^T Ax - b^T x$. Thus, in this context iterative methods for linear systems, such as the Richardson iteration, can also be interpreted as optimization algorithms. Let us consider the (relaxed) Richardson iteration for $Ax = b$, i.e., $x_{k+1} = (I - \theta A)x_k + \theta b$. After some minor manipulations and making use of $\nabla f(x_k) = Ax_k - b$ we arrive at the equivalent formulation

$$x_{k+1} = x_k - \theta\nabla f(x_k).$$

The latter is what is called a gradient method. A step from x_k into (an appropriately scaled) direction of the gradient $\nabla f(x_k)$ yields a decrease in the objective function f , i.e., $f(x_{k+1}) \leq f(x_k)$. Along the Richardson aka Gradient method the scaling (also called step size) θ is fixed (in a machine learning context the step size θ is called the *learning rate*). However, one could also choose a different θ_k in each iteration step. This gives the more general version

$$x_{k+1} = x_k - \theta_k\nabla f(x_k). \tag{1}$$

The well known method of *steepest descent* is given by choosing

$$\theta_k = \frac{r_k^\top r_k}{r_k^\top Ar_k}, \tag{2}$$

where $r_k := Ax_k - b$ is the k-th residual. This choice can be shown to be optimal in terms of convergence speed. Even more general, one can think of using a different preconditioner N_k in each iteration step

$$x_{k+1} = x_k - N_k\nabla f(x_k),$$

i.e., representing the gradient with respect to another inner product. This will later correspond to Newton-type optimization algorithms.

Task

Consider the following setting:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_0 = \begin{bmatrix} 4 \\ 1.4 \end{bmatrix}.$$

Convince yourself that A is spd. Determine the minimal and maximal eigenvalue of A, i.e., $\lambda_{\min}$ and $\lambda_{\max}$, respectively. What is the solution to $Ax = b$? Now extend your code (in particular `iter_solve()`) from previous sheets:

1. Implement the **steepest descent method** (1) by choosing the stepsize θ_k from (2) in each iteration step.
2. Find a way to apply the **conjugate gradient method** to solve a system $Ax = b$, where A is spd.
   Hint: You can either implement it on your own or find a SciPy routine (for the latter: you can collect all iterates x_k by using the callback interface).
3. **Test:** Solve the above problem with the following routines:
   a) Richardson method with $\theta = \frac{2}{\lambda_{\max}}$
   b) Richardson method with $\theta = 0.9 \cdot \frac{2}{\lambda_{\max}}$
   c) Richardson method with optimal $\theta = \frac{2}{\lambda_{\min}+\lambda_{\max}}$
   d) Steepest descent method
   e) conjugate gradient method

   Generate the following two plots:

   1. Plot the iterates x_k for all the runs into the same 2d plot (use different colors).
   2. Plot the function values $f(x_k) = \frac{1}{2}x_k^T Ax_k - b^T x_k$ for each iterate and all runs into a second plot (use different colors).

Solution:

```python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import importlib

# import iterative solvers from previous build
iterSolver = importlib.import_module("prog-LinearIterations_JacRich_solution")


if __name__ == "__main__":
    # ---------------
    # PROBLEM SETUP
    # ---------------
    eigmin, eigmax = 2, 10
    A = np.array([[eigmin, 0], [0, eigmax]])
    b = np.array([0, 0])
    x = np.array([0, 0])
    x0 = np.array([4, 1.4])

    theta_max = 2./eigmax
    theta_bad = 0.99 * theta_max
    theta_opt = 2 / (eigmin+eigmax)

    # ---------------
    # METHOD SETUP
    # ---------------
    maxiter = 200
    legend = ["Richardson max", "Richardson bad", "Richardson opt",
              "Steepest Descent", "Conjugate Gradient"]
    methods = ["Richardson", "Richardson", "Richardson",
               "steepestDescent", "CG"]
    theta = [theta_max, theta_bad,  theta_opt,
             -1, -1]
    colors = ["y", "c", "b",
              "g", "r"]
    numberRuns = len(methods)

    # ---------------
    # SOLVE
    # ---------------
    X = []
    F = []
    for i, method in enumerate(methods):
        _X = iterSolver.main(A, b, x0, maxiter,
                                {method: theta[i]}, plot=0, verbose=0)[0]
        X += [np.array(_X)]
        F += [[0.5 * np.dot(x, A.dot(x)) for x in _X]]

    # ------------- #
    # plot iterates
    # ------------- #
    fig, ax = plt.subplots()
    for i, _X in enumerate(X):
        plt.plot(_X[:, 0], _X[:, 1], colors[i]+"o-")
    # add level sets
    m = 200
    for k in range(m):
        e = Ellipse(xy=np.zeros(2), width=eigmax*0.8**k,
                    height=eigmin*0.8**k, angle=0, fill=False)
        ax.add_artist(e)
    plt.legend(legend)
    plt.axis('equal')
    plt.show()
    # -------------- #
    # plot objective
    # -------------- #
    plt.figure("objective-function")
    for i, f in enumerate(F):
        plt.plot(f, colors[i]+"x-")
    plt.legend(legend)
    plt.show()
```

10 Small Tour into Imaging

Ex 135 Linear Algebra, Image Processing

Discrete Convolution, Filters and Kronecker Product

1. *Definition: Let $f\colon \mathbb{Z} \to \mathbb{R}$ and $g\colon \mathbb{Z} \to \mathbb{R}$. Then the discrete convolution $*\colon \mathbb{R}^{\mathbb{Z}} \times \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}$ is defined by*
$$(f * g)(k) = \sum_{j\in\mathbb{Z}} f(j)g(k-j), \qquad k \in \mathbb{Z}.$$
Consider
$$f(i) := \begin{cases} 1 & : i \in \{0,1,2\} \\ 0 & : else \end{cases}, \qquad g(i) := \begin{cases} \frac{1}{3} & : i \in \{-1,0,1\} \\ 0 & : else \end{cases},$$
and compute the discrete convolution $f * g$.
Remark: You can think of f as a zero-padded finite signal and of g as a centered filter. Also, most values of $f * g$ are zero.

2. Consider f and g from above. Define a matrix $G \in \mathbb{R}^{3\times 3}$, so that
$$G \cdot \begin{pmatrix} f(0) \\ f(1) \\ f(2) \end{pmatrix} = \begin{pmatrix} (f*g)(0) \\ (f*g)(1) \\ (f*g)(2) \end{pmatrix}$$
Do you observe a structure in G?
Remark: The matrix G is the filter matrix for the average filter $\frac{1}{3}[1\ 1\ 1]$.

3. Now consider the 2d data
$$F := \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$
and the matrix G from above. Then compute the product
$$GFG^\top.$$
Remark: By computing this product we apply the 1d filter to each dimension of F, e.g., first column-wise and then row-wise: $(G(GF)^\top)^\top$. Thereby we obtain a 2d filter from the 1d filter.

4. *Definition: The Kronecker product $\otimes\colon \mathbb{R}^{m\times n} \times \mathbb{R}^{M\times N} \to \mathbb{R}^{mM\times nN}$ is defined by*
$$A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}.$$
Also let vec: $\mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ define the row-major vectorization (here start indexing at 0)
$$\textit{vec}(F)(k) := F(\lfloor \tfrac{k}{n} \rfloor, k \mod n), \quad 0 \le k < mn.$$
For G and F from above derive
$$\text{vec}(GFG^\top) \quad \text{and} \quad (G \otimes G)\text{vec}(F).$$
Remark: These two vectors should be the same. The right one defines the application of the 2d filter as a matrix–vector product.

5. *Definition: Let $F\colon \mathbb{Z}^2 \to \mathbb{R}$ and $W\colon \mathbb{Z}^2 \to \mathbb{R}$. Then the two-dimensional discrete convolution $*\colon \mathbb{R}^{\mathbb{Z}^2} \times \mathbb{R}^{\mathbb{Z}^2} \to \mathbb{R}^{\mathbb{Z}^2}$ is analogously defined by*
$$(F * W)(k,\ell) = \sum_{(i,j)\in\mathbb{Z}^2} F(i,j)W(k-i,\ell-j), \quad (k,\ell) \in \mathbb{Z}^2.$$
Consider
$$F(i,j) := \begin{cases} 1 & : (i,j) \in \{(1,0),(1,1),(1,2),(2,1)\} \\ 0 & : else \end{cases}, \qquad W(i,j) := \begin{cases} \frac{1}{9} & : -1 \le i,j \le 1 \\ 0 & : else \end{cases},$$
and compute the discrete convolution $F * W$. Compare the result to $GFG^\top$ from above.
Remark: This F is the zero-padded version of the 2d signal F from above and W is the 2d average filter.

6. Just for fun: Install the free software `https://www.gimp.org/downloads/`, open an image of your choice and check out the menu point "Filter". In particular, you can apply your own filters under "Filter > Generic > Convolution Matrix" (`https://docs.gimp.org/2.8/en/plug-in-convmatrix.html`).

Solution:

1. Intuition (formally): Thinking about f and g as infinite vectors and defining $g_k(j) := g(k-j)$ for a fixed k, then under abuse of notation we have
$$(f * g)(k) = f^\top g_k.$$
We have $g_0(j) = g(-j)$ (= reflection of g on y axis), so that g_k is a shift of g_0 of amount k (to the right if k is positive and to the left if k is negative). If g is even (=symmetric around the origin), then by definition $g(j) = g(-j)$.
Note that the support of f and g is $\{0, 1, 2\}$ and $\{-1, 0, 1\}$, respectively. Thus
$$(f*g)(k) = \begin{cases} f(0)g(-1-0) + f(1)g(-1-1) + f(2)g(-1-2) & : k = -1 \\ f(0)g(0-0) + f(1)g(0-1) + f(2)g(0-2) & : k = 0 \\ f(0)g(1-0) + f(1)g(1-1) + f(2)g(1-2) & : k = 1 \\ f(0)g(2-0) + f(1)g(2-1) + f(2)g(2-2) & : k = 2 \\ f(0)g(3-0) + f(1)g(3-1) + f(2)g(3-2) & : k = 3 \end{cases} = \begin{cases} \frac{1}{3} & : k = -1 \\ \frac{2}{3} & : k = 0 \\ 1 & : k = 1 \\ \frac{2}{3} & : k = 2 \\ \frac{1}{3} & : k = 3 \end{cases}$$
Observe that the convolution "increases the support" of f.

2. We define the *Toeplitz* matrix
$$G = \frac{1}{3}\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

3. Applying the filter G to each column:
$$GF = \frac{1}{3}\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix}$$
Now apply the filter G to each row of this intermediate result:
$$GFG^\top = (G(GF)^\top)^\top = \frac{1}{9}\left(\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix}\right)^\top = \frac{1}{9}\begin{pmatrix} 2 & 3 & 2 \\ 3 & 4 & 3 \\ 3 & 4 & 3 \end{pmatrix}$$

4. We find
$$\text{vec}(GFG^\top) = \frac{1}{9}\begin{pmatrix} 2 \\ 3 \\ 2 \\ 3 \\ 4 \\ 3 \\ 3 \\ 4 \\ 3 \end{pmatrix},$$
$$(G \otimes G) = \frac{1}{3}\begin{pmatrix} G & G & 0 \\ G & G & G \\ 0 & G & G \end{pmatrix} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}, \quad \text{vec}(F) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix},$$
$$(G \otimes G)\text{vec}(F) = \frac{1}{9}\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 2 \\ 3 \\ 4 \\ 3 \\ 3 \\ 4 \\ 3 \end{pmatrix}.$$

5. Neglecting zeroes we find

$$(F * W)(-1 \leq k \leq 3, -1 \leq \ell \leq 3) = \frac{1}{9}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 2 & 1 \\ 1 & 3 & 4 & 3 & 1 \\ 1 & 3 & 4 & 3 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

   Observe that the convolution "increases the support" of F.

6. GIMP demo.

Ex 136 Least Squares Problems, Python

More General Regularization Terms and Image Inpainting

Background:
Based on the idea of Tikhonov regularization we can use more general regularization terms by transforming the vector x with some matrix $G \in \mathbb{R}^{m \times n}$. Specifically, let us consider the more general regularized problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2 + \frac{\delta}{2}\|Gx\|_2^2, \quad \delta > 0 \text{ small.} \tag{1}$$

The corresponding "regularized" normal equation then reads as

$$(A^T A + \delta G^\top G)x = A^T b. \tag{2}$$

Observe that for $G = I$ we obtain the standard Tikhonov (or L^2-) regularization. We easily see that if $G^\top G$ is positive definite, then so is $A^T A + \delta G^\top G$ for positive δ, so that (2) is uniquely solvable.

We will apply this framework to the problem of image inpainting. Therefore assume you are given the deteriorated image b, which is obtained from the unknown original image x through the following masking operation

$$b_i = \begin{cases} x_i & i \in \texttt{indices}, \\ 0 & \text{else}, \end{cases} \tag{3}$$

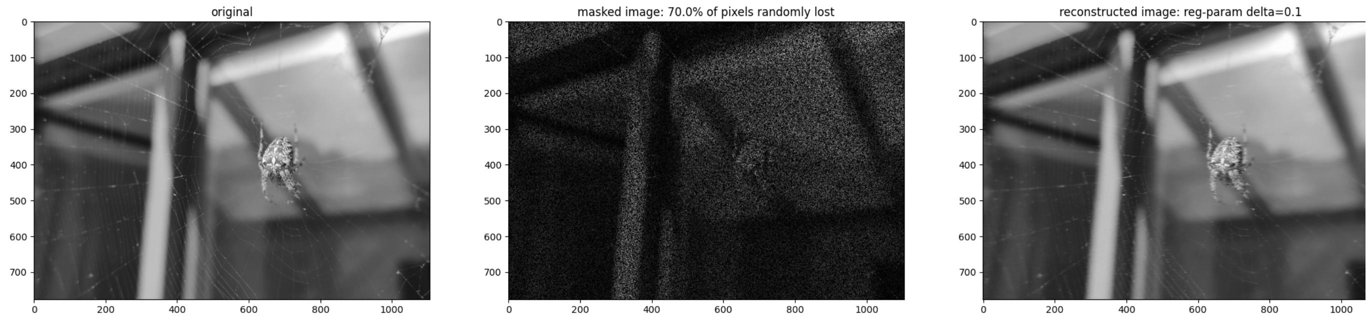
where `indices` is a list of random pixels. In words, the pixels in `indices` survived, the rest is set to zero and therefore lost. We want to recover those lost pixels. Note that the images are considered being flattened and thus vectors, so that the masking operation (3) can be written as a matrix-vector product

$$b = Ax$$

for some quadratic matrix A. You will see below that A is not of full rank, so that we cannot simply solve this equation. Instead, we will seek for solutions of the regularized least squares problem (1) by solving the linear equation (2).
For this purpose will stick to a particular G which is related to what is called Sobolev (or H^1-) regularization. Specifically we consider the 1-d finite difference quotient

$$G = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1) \times n}, \tag{4}$$

which has 1 on the main diagonal, -1 on the first lower off-diagonal, 0 everywhere else and is of dimension $(n+1) \times n$. Then, given the measured image b, the masking operator A (we assume that we know which pixels are original) and the regularization G we reconstruct the unknown image x by solving equation (2). See Figure below for an example experiment.


original
masked image: 70.0% of pixels randomly lost
reconstructed image: reg-param delta=0.1
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

Example experiment.

Step by Step
First note that due to the high dimensional image data you need to work with sparse matrices (`scipy.sparse`); specifically you should work with the CSR format.

1. **Choose original image:** Choose an image and load it in gray-scale as $H \times W$ `numpy.ndarray` using the code snippet:

```python
def load_image_as_gray(path_to_image):
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img
```

2. **Masking:** Write a function

   `b, indices = masking(img, percentage),`

   which takes as input an image `img` as $H \times W$ `numpy.ndarray` and a number `percentage` $\in (0, 1)$ which indicates the percentage of pixels that are randomly kept. It shall return the masked image b as an $n := (H \cdot W)$-dimensional vector and the list `indices` $\subset \{0, \dots, n-1\}$ indicating which pixels are original.

   - Ultimately, the image needs to become a vector. For this purpose you can use for example the method `.ravel()`. No matter which method/function you choose, be aware of how the vector is flattened (standard is often: row-major/C-style order).
   - To generate a random set of indices based on the parameter `percentage` have a look at the function `numpy.random.choice`.

3. **Solving:** Write a function

   `reconImg = inpainting(b, indices, delta, G),`

   which expects a deteriorated image b as vector of length n, a list `indices` $\subset \{0, \dots, n-1\}$ of length $\leq n$ indicating which pixels are original, a regularization parameter `delta` > 0 and a matrix `G` $\in \mathbb{R}^{m \times n}$. It then solves (2) and returns the solution as an n-dimensional vector `reconImg`.

   - The sparse $(n \times n)$ masking matrix A is zero everywhere except for $a_{ii} = 1$ for $i \in$`indices`. For example, you can easily implement this matrix with `sparse.coo_matrix` and then transform it to CSR format via its method `.tocsr()`.
   - You can implement the sparse matrix G from (4) with the help of the function `scipy.sparse.eye`.
   - You can then solve the system (2) with `scipy.sparse.linalg.spsolve`.

4. **Analysis:**

   - Play around with different choices for the parameters `delta` and `percentage`.
   - You can plot your images (original, masked, reconstructed) with

     `matplotlib.pyplot.imshow((...).reshape(H,W), cmap='gray').`

   - Bonus: Try to recover the image with standard Tikhonov regularization, i.e., $G = I$. Do you have an idea why this does not work here?
   - Bonus: With this choice of G, you get flattening artifacts in one dimension (horizontal or vertical depending on your flattening approach) and at the boundary of the reconstructed image. Do you have an idea why?

5. **Bonus:** Instead of removing randomized set of pixels, remove a patch of the image. Can you reconstruct it?

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg


def load_image_as_gray(path_to_image):
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img
```

```python
def masking(img, percentage):
    """Randomly sets (1-percentage)*100 % of the pixels to zero

    Parameters
    ----------
    img : (H, W) ndarray
          original image
    percentage : float
                 number in (0,1)

    Returns
    -------
    b : ndarray
        masked image of shape (H*W, 1)
    indices : list
              of length <=H*W, subset of {0,...,(H*W-1)}
              contains indices of pixels that were kept
    """
    # flatten image in C order, i.e., append row by row
    print("Image dimensions: ", img.shape)
    img = img.ravel()
    n = len(img)
    # masking operator
    indices = np.random.choice(np.arange(n), replace=False,
                               size=int(n * percentage))

    b = np.zeros(n)
    b[indices] = img[indices]

    return b, indices


def inpainting(b, indices, delta, G):
    """
    inpainting based on trivial 1-d Sobolev Regularization

    Parameters
    ----------
    b : ndarray
        of shape (n, 1)
    indices : list or array-like
              of length <= n, subset of {0,...,(n-1)}
              contains indices of pixels that were kept
    delta : float
            positive number, regularization parameter
    G : numpy.ndarray
        of dimension (m,n), determining regularization

    Returns
    -------
    reconImg : ndarray
               of shape (n, 1)
               reconstructed image
    """
    n = len(b)
    # masking operator with A = A.T@A (n,n)
    A = sparse.coo_matrix((np.ones(len(indices)),
                          (indices, indices)), shape=(n, n)).tocsr()
    # solve with scipy sparse (note that: A.T@A = A)
    reconImg = scipy.sparse.linalg.spsolve(A + delta * G.T@G, A.dot(b))
    return reconImg


if __name__ == "__main__":

    # INPUT PARAMETERS
    # --------------
    path_to_image = 'kalle.jpg'  # 'happy_dog.jpg' # the image
    percentage = 0.1  # we randomly keep 100*percentage % of the data
    delta = 0.001  # regularization parameter

    # ORIGINAL IMAGE
    # --------------
    img = load_image_as_gray(path_to_image)
    H, W = np.shape(img)
    # plot original image
    plt.figure("Image Inpainting")
    plt.subplot(1, 3, 1)
    plt.imshow(img, cmap='gray')
    plt.title("original")

    # MASKED IMAGE
    # -----------
    b, indices = masking(img, percentage)
    n = len(b)
    # plot noisy image
    plt.subplot(1, 3, 2)
    plt.imshow(b.reshape((H, W)), cmap='gray')
    plt.title("masked image: {}% of pixels randomly lost".format(
              (1-percentage)*100))

    # RECONSTRUCTED IMAGE
    # -------------------
    # difference quotient (k+1,k)
    G = sparse.eye(n+1, n, k=0) - sparse.eye(n+1, n, k=-1)  # sparse.eye(n,n) #
    reconImg = inpainting(b, indices, delta, G)
    # plot denoised image
    plt.subplot(1, 3, 3)
    plt.imshow(reconImg.reshape((H, W)), cmap='gray')
    plt.title("reconstructed image: reg-param delta={}".format(delta))
#    plt.savefig("Image_Inpainting")
```

Ex 137 Image Processing, Least Squares Problems, Python

Image Inpainting

1. Download the file noisy_image.npy (link “data” next to the link “Sheet 11”) and use numpy.load() to import it into your Python script. The shape is (823, 1238). This image is obtained from the original image by randomly setting 99% (!) of the pixel values to zero (=black).

2. Apply the techniques from the lecture to recover the original image:

   a) Estimate the applied masking matrix A from the given noisy image.

   b) Solve a regularized least squares problem. Note that the solving step may take a few seconds due to the high–dimensional data.

   (Can you identify the objects on the image?)

Solution:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse as sparse


def masking_matrix(image_shape, percentage):
    """Randomly sets (1-percentage)*100 % of the pixels to zero, i.e.
    percentage*100% of the pixels are kept"""
    import numpy as np
    import scipy.sparse as sparse
    dim = np.prod(image_shape)
```

```python
    indices = np.random.choice(np.arange(dim), replace=False, size=int(dim * percentage))
    A = sparse.coo_matrix((np.ones(len(indices)),
                           (indices, indices)), shape=(dim, dim)).tocsr()
    A = A.tocsr()
    return A


def generate_data(original, noisy, percentage):
    original_img = np.array(Image.open("steinpilz.png").convert("L"))
    original_img = original_img[::2,::2]
    img_shape = np.shape(original_img)
    masking_matrix(img_shape, percentage)

    A = masking_matrix(img_shape, percentage)
    b = A.dot(original_img.ravel()).reshape(img_shape)
    np.save("noisy_image", b)
    np.save("original_image", original_img)
    return None

def estimate_masking(data):

    data = data.ravel()
    indices = np.where(data != 0)[0]
    dim = np.prod(np.shape(data))
    A = sparse.coo_matrix((np.ones(len(indices)),
                           (indices, indices)), shape=(dim, dim)).tocsr()
    A = A.tocsr()
    return A

def difference_matrix(image_shape):
    """returns difference matrix D; shape = edges x nodes (graph=squared grid)"""
    H, W = image_shape
    n = H * W
    D = sparse.eye(n+1, n, k=0) - sparse.eye(n+1, n, k=-1)
    D_horizontal = sparse.kron(sparse.eye(H), sparse.eye(W-1, W, k=0) - sparse.eye(W-1, W, k
=1))
    D_vertical = sparse.kron(sparse.eye(H-1, H, k=0) - sparse.eye(H-1, H, k=1), sparse.eye(W))
    D = sparse.vstack((D_horizontal, D_vertical))
    D = D.tocsr()
    return D

def reg_lstsq(A, b, D, reg_param=0.1, verbose=1):
    """A csr nxn, b vector nx1, D csr pxn, delta float"""
    import scipy.sparse.linalg
    _delta = reg_param
#    n = len(b)
    if verbose:
        print("start solving")
    recon_img = scipy.sparse.linalg.gmres(A.T @ A + _delta * D.T @ D, A.dot(b),maxiter=30)[0]
    if verbose:
        print("finished solving")
    # scipy.sparse.linalg.spsolve(A.T@A + _delta * D.T@D, A.dot(b))#
    return recon_img

def experiment(data_fname, reg_param, save=False):

    b = np.load(data_fname)
    img_shape = np.shape(b)
    print("Resolution:", img_shape)

    A = estimate_masking(b)
    D = difference_matrix(img_shape)
    recon_img = reg_lstsq(A, b.ravel(), D, reg_param=reg_param)

    plt.figure("Image Inpainting")
    plt.subplot(1, 2, 1)
    plt.imshow(b.reshape(img_shape), cmap='gray')
    plt.title("noisy image")
    plt.axis(False)

    plt.subplot(1, 2, 2)
    plt.imshow(recon_img.reshape(img_shape), cmap='gray')
    plt.axis(False)
    plt.title("reconstructed image: reg-param delta={}".format(reg_param))
    plt.show()

    if save:
        plt.savefig("Image_Inpainting", dpi='figure')

    return None




if __name__ == "__main__":

    generate_data(original="steinpilz.png", noisy="noisy_image", percentage=0.01)
    original = np.load("original_image.npy")
    plt.imshow(original, cmap="gray")
    experiment("noisy_image.npy", 0.02, save=False)
```