Problem 1

E) Show that the Hilbert matrix with dimension n

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & & \cdots & \frac{1}{n+1} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & & & \frac{1}{n+2} \\ \vdots & \vdots & \vdots & & & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & & \cdots & \frac{1}{2n-1} \end{bmatrix} = \left[\frac{1}{j+k-1}\right]_{j,k\le n}$$

is positive definite.

Notes:
1. To create a Hilbert matrix with dimension n in Python you can write `A=scipy.linalg.hilbert(n)`.
2. In Matlab, you can use the command `A=hilb(n)`.
3. Hilbert matrices are ill-conditioned, see the programming assignment.

Problem 2

E) The row-sum $R_i(A)$ and the column-sum $C_i(A)$ of a matrix $A = [a_{ik}] \in \mathbb{R}^{n\times n}$ are defined as follows:

$$R_i(A) = \sum_{k=1}^{n} |a_{ik}|, \qquad C_k(A) = \sum_{i=1}^{n} |a_{ik}|.$$

In the lecture, using an example, we showed that $\|A\|_\infty = \max_{i=1}^n R_i(A)$, thus the matrix norm induced by the ∞-norm is the row-sum norm. Show for an arbitrary 2×2-matrix, that the matrix norm induced by the 1-norm is the column sum norm, i.e. that

$$\|A\|_1 = \max_{k=1}^n C_k(A).$$

Problem 3 **4 Points**

For $p \in \{1, 2, \infty\}$, compute the norms $\|A\|_p$, $\|A^{-1}\|_p$ and the condition numbers $\operatorname{cond}_p(A)$ for the following matrices.

$$\text{E a)}\quad A = \begin{bmatrix} 4 & -13 \\ 8 & -1 \end{bmatrix}, \qquad \text{E b)}\quad A = \begin{bmatrix} 0 & -2 & 0 \\ 5 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \qquad \text{E c)}\quad A = I \quad \text{(Identity matrix).}$$

$$\text{H a)}\quad A = \begin{bmatrix} 22 & 3 \\ -18 & 13 \end{bmatrix}, \qquad \text{H b)}\quad A = \begin{bmatrix} 0 & -4 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 0 & 4 & 0 \end{bmatrix}.$$

Hints: The formula to compute the inverse of a 2×2 matrix is:

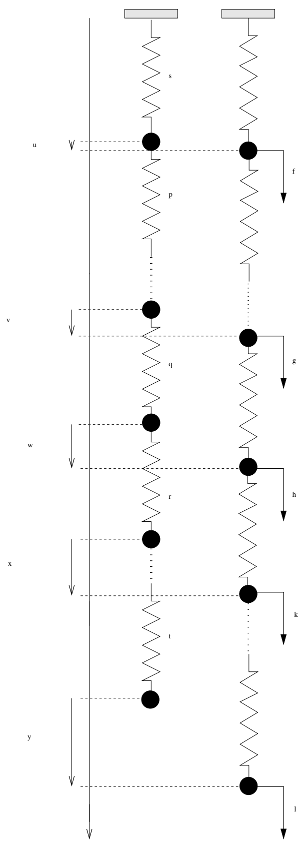
$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \Rightarrow \qquad A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

To compute the inverse of a block diagonal matrix, invert the block matrices:

$$A = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \qquad \Rightarrow \qquad A^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{bmatrix}.$$

Problem 4

E) A model from mechanics. Consider the spring mass system below with masses $m_k > 0$ attached to (massless) springs with stiffness constants $s_k > 0$. The left side shows the chain of springs when it is relaxed, i.e. when the effect of gravity is neglected. When the effect of gravity is taken into account, the masses m_k are pulled downwards by a force $f_k = g\, m_k$ ($g = 9.81$ gravity constant), the springs are stretched and the masses are displaced by u_k from their original positions.



Show: The relation between the forces f_k and the displacements u_k is given by

$$\underbrace{\begin{bmatrix} s_1+s_2 & -s_2 & 0 & & & & \\ -s_2 & s_2+s_3 & -s_3 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & -s_k & s_k+s_{k+1} & -s_{k+1} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & -s_{n-1} & s_{n-1}+s_n & -s_n \\ & & & & 0 & -s_n & s_n \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{k-1} \\ u_k \\ u_{k+1} \\ \vdots \\ u_{n-2} \\ u_{n-1} \\ u_n \end{bmatrix}}_{u} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_k \\ \vdots \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}.$$

Note: The stiffness matrix is positive definite. The physical explanation for this is that $\frac{1}{2}u^\top Su > 0$ is the elastic energy which is stored in the deformed springs.

Programming Assignment

(a) Let A be the Hilbert matrix from Problem 1 and let $b \in \mathbb{R}^n$ be the sum of the first and the last column of A. In Python-notation, to create the Hilbert vector from `A=hilbert(n)`, we write

`b=A[:,0]+A[:,n-1].`

In Matlab-notation, to create the Hilbert vector from `A=hilb(n)`, we write `b=A(:,1)+A(:,n)`.

The solution to the system of linear equations $Ax = b$ is given by

$$x = [\,1 \;\, 0 \;\, \dots \;\, 0 \;\, 1]^\top \in \mathbb{R}^n.$$

For large n the condition number of A is large. Then the equation $Ax = b$ cannot be solved exactly anymore on the computer.

Generate a table (i.e. a matrix) T with the following data.

- The first column of T has the numbers $n = 2, 3, \dots, 13$.
- The second column of T has the condition numbers of the Hilbert matrices with dimensions $n = 2, 3, \dots$, in the ∞-norm. Use the Python commands `numpy.linalg.norm` and `numpy.linalg.cond` or the Matlab commands `norm` and `cond`. Find more explanations on how to use these functions by typing `help(norm)` and `help(cond)` at the command prompt. To keep the table readable, multiply the condition numbers by 10^{-16}.
- The third column T has the relative error $\frac{\|x-x_b\|_\infty}{\|x\|_\infty}$, where x_b is the numerical solution of $Ax = b$, $A \in \mathbb{R}^{n\times n}$, computed in Python with the command `scipy.linalg.solve(A,b)` or in Matlab with the backlash command $x_b = A\backslash b$.
- The fourth column of T has the relative error $\frac{\|x-x_{lu}\|_\infty}{\|x\|_\infty}$, where x_{lu} is the numerical solution of $Ax = b$, obtained by first computing the LU-decomposition `[L,U,P]=lu(A)` and then solving the two systems of equations by forward substitution and by backward substitution. (You can use the Python command `solve` or the Matlab command `\`. Do not forget to multiply b with the permutation matrix P.)
- The fifth row of T has the relative error $\frac{\|x-x_{chol}\|_\infty}{\|x\|_\infty}$, where x_{chol} is the numerical solution of $Ax = b$, obtained by first computing the Cholesky decomposition, in Python with the command `L=numpy.linalg.cholesky(A)` or in Matlab with `R=chol(A)`, and then solving the systems by forward and backward substitution. (You can use `solve` or the backlash command. Note that R is an upper triangular matrix.)
- The sixth column shows $\|A\,x_{chol} - b\|_\infty$.

Which algorithm is the worst? What happens if you compute entries of the table for $n = 14$.

(b) Tridiagonal matrices appear in many applications (see e.g. the stiffness matrix of the spring-mass system in Problem 4). The computational effort to compute the Cholesky decomposition of such a (sparse) matrix can be reduced significantly. Every symmetric positive-definite tridiagonal matrix A has a Cholesky decomposition of the form:

$$\underbrace{\begin{bmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & b_3 & \ddots & \ddots & \\ & & & \ddots & \ddots & b_{n-1} \\ & & & & b_{n-1} & a_n \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} c_1 & & & & \\ d_1 & c_2 & & & \\ & d_2 & c_3 & & \\ & & d_3 & \ddots & \\ & & & \ddots & \\ & & & d_{n-1} & c_n \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} c_1 & d_1 & & & \\ & c_2 & d_2 & & \\ & & c_3 & d_3 & \\ & & & \ddots & \ddots \\ & & & & \ddots & d_{n-1} \\ & & & & & c_n \end{bmatrix}}_{L^T}$$

Write a function `tridiagsolve(a,b,f)`, in Python or Matlab, to solve `Au=f`, without saving the intermediate results in a matrix. Your function should first compute the coefficients c_j, d_j: First c_1, next d_1, then c_2, etc. Then solve the equation $Au = f$, by forward-substitution with the matrix L and by backward-substitution with the matrix L^T. However, do not explicitly compute the matrix L (to avoid saving lots of zeros).

Use this function to solve the equation system for the spring-mass system from Problem 4 with: $s_1 = s_2 = \ldots = s_n = 1$, $f_1 = f_2 = \ldots, = f_n = 1$ and for different values of n. How do the displacements $[u_1 \;\; u_2 \;\; \ldots \;\; u_n]^T$ depend on n?

Hint: The solution for $n = 3$ is $u = [3\ 5\ 6]$.

(c) Consider the matrix

$$duck = \begin{bmatrix} 0 & 1 & 1 & 1.5 & 2.5 & 3 & 3 & 4 & 2 & 2 & 1 & -3.7 & -5 & -4 & -3 & 0 \\ 1 & 0 & 3.5 & 4 & 4 & 3.5 & 3 & 3 & 2 & -1 & -2 & -2 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

The columns of this matrix are the points of a piecewise-linear polygonal curve that looks like a duck. To plot the duck in Python use: `import pylab as pl; pl.plot(duck[0,:],duck[1,:]); pl.show()`, in Matlab write: `plot(duck(1,:),duck(2,:))`.

Let $A \in \mathbb{R}^{2\times 2}$. Multiplying the duck from the left with a matrix

$$\texttt{A} * \texttt{duck}$$

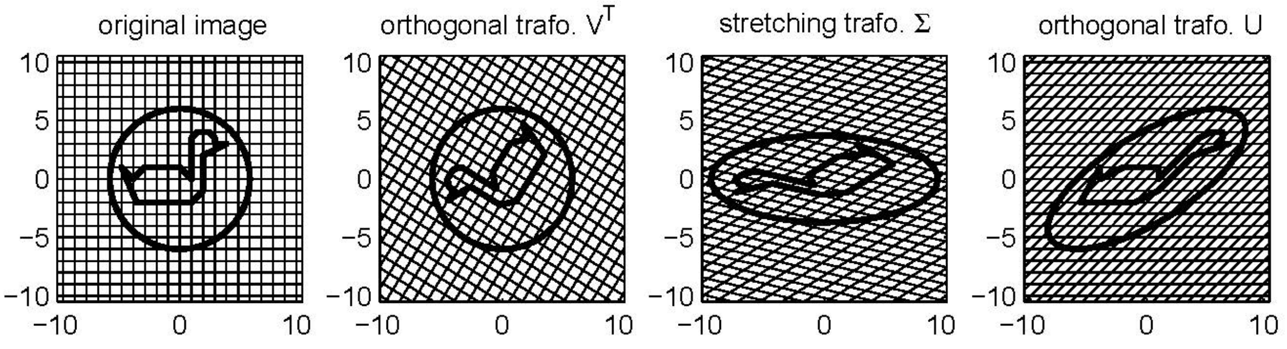
results in a new matrix, whose columns are the points of a transformed duck. The same method can be applied to transform a circle. First generate a matrix `circle`, whose columns are points on the circle (use polar coordinates $x = r\cos(t)$, $y = r\sin(t)$, $t \in [0, 2\pi]$). Then compute the points on the transformed circle

$$\texttt{A} * \texttt{circle}.$$

Write a program which

- computes the singular value decomposition of a given matrix $A = U\Sigma V^T$. The Python command is `scipy.linalg.svd(A)`, the Matlab command is `svd(A)`. See `help(svd)` for more explanations.
- generates a figure with the following plots:
  1) the original duck and a circle,
  2) the duck transformed by V^T and the circle transformed by V^T,
  3) the duck transformed by ΣV^T and the circle transformed by ΣV^T,
  4) the duck transformed by A and the circle transformed by A.

Your figure should look similar to the following graph (without the grid):


original image
orthogonal trafo. V^T
stretching trafo. Σ
orthogonal trafo. U