Advanced Numerical Analysis

Lecture 8

Spring 2025

Daniel Kressner

Fooling LU factorization into existence?

Consider

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

LU factorization does not exist because leading 1×1 principal submatrix is 0.

Maybe, can be fixed by slightly perturbing A...

Fooling LU factorization into existence?

$$A = \begin{pmatrix} 10^{-16} & 1 & 1 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

Now, LU factorization exists and Python returns

```
L =
   1.0000e+00                0                0
            0   1.0000e+00                0
   1.0000e+16  -1.0000e+16   1.0000e+00
U =
   1.0000e-16   1.0000e+00   1.0000e+00
            0   1.0000e+00  -1.0000e+00
            0             0  -2.0000e+16
```

The presence of $\pm 10^{16}$ in L should worry you!

Fooling LU factorization into existence?

Solving the linear system $A\mathbf{x} = \mathbf{b}$ for $b = [2, 2, 1]^T$ with these triangular factorization gives the "solution"

```
x =
         0
         2
         0
```

Up to $\pm 10^{-16}$, exact solution is

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}.$$

This is computed very accurately with Python's linear systems solver!

Morale of the story:

Not only zero pivots but also (very) small pivots need to be avoided.

LU factorization with pivoting

Algorithm 4.13

for $k = 1, \ldots, n-1$ **do**
  Search $i \in \{k, \ldots, n\}$ such that $|a_{ik}| \geq |a_{i'k}|$ for all $i' \in \{k, \ldots, n\}$
  Define P_k as permutation matrix that exchanges rows i and k.
  Exchange rows: $A \leftarrow P_k A$
  for $i = k+1, \ldots, n$ **do**
    $a_{ik} \leftarrow \dfrac{a_{ik}}{a_{kk}}$
    for $j = k+1, \ldots, n$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik} a_{kj}$
    end for
  end for
end for
Set U to upper triangular part of A.
Set L to lower triangular part of A and diagonal entries 1.
Set $P := P_{n-1} P_{n-2} \cdots P_2 P_1$.

Demonstration of Algorithm 4.13 for matrix:

$$A = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 4 & 4 & 4 & 4 \\ 2 & 8 & 8 & 8 \\ 1 & 3 & 7 & 7 \end{pmatrix}.$$

Initialization:

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

Step $k = 1$ in Algorithm 4.13:

- Select entry of largest magnitude in column 1 of A:

$$A = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 4 & 4 & 4 & 4 \\ 2 & 8 & 8 & 8 \\ 1 & 3 & 7 & 7 \end{pmatrix}$$

- Swap rows $1 \leftrightarrow 2$ of A and P:

$$A \leftarrow \begin{pmatrix} 4 & 4 & 4 & 4 \\ 2 & 3 & 4 & 5 \\ 2 & 8 & 8 & 8 \\ 1 & 3 & 7 & 7 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Reduction of column 1:

$$A \leftarrow \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1/2 & 1 & 2 & 3 \\ 1/2 & 6 & 6 & 6 \\ 1/4 & 2 & 6 & 6 \end{pmatrix}$$

(Red parts correspond to factor L.)

Step $k = 1$ in Algorithm 4.13:

- Select entry of largest magnitude in column 2 below diagonal:

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

- Swap rows $2 \leftrightarrow 3$ of A and P:

$$A \leftarrow \begin{pmatrix} & 4 & 4 & 4 & 4 \\ 1/2 & 6 & 6 & 6 \\ 1/2 & 1 & 2 & 3 \\ 1/4 & 2 & 6 & 6 \end{pmatrix}, \quad P \leftarrow \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Reduction of column 2:

$$A \leftarrow \begin{pmatrix} & 4 & 4 & 4 & 4 \\ 1/2 & & 6 & 6 & 6 \\ 1/2 & 1/6 & 1 & 2 \\ 1/4 & 1/3 & 4 & 4 \end{pmatrix}$$

(Red parts correspond to factor L.)

Step $k = 1$ in Algorithm 4.13:

- Select entry of largest magnitude in column 3 below diagonal:

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

- Swap rows $3 \leftrightarrow 4$ of A and P:

$$A = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1/2 & 6 & 6 & 6 \\ 1/4 & 1/3 & 4 & 4 \\ 1/2 & 1/6 & 1 & 2 \end{pmatrix}, \quad P \leftarrow \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

- Reduction of column 3:

$$A \leftarrow \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1/2 & 6 & 6 & 6 \\ 1/4 & 1/3 & 4 & 4 \\ 1/2 & 1/6 & 1/4 & 1 \end{pmatrix}$$

(Red parts correspond to factor L. These parts are also swapped!)

Result:

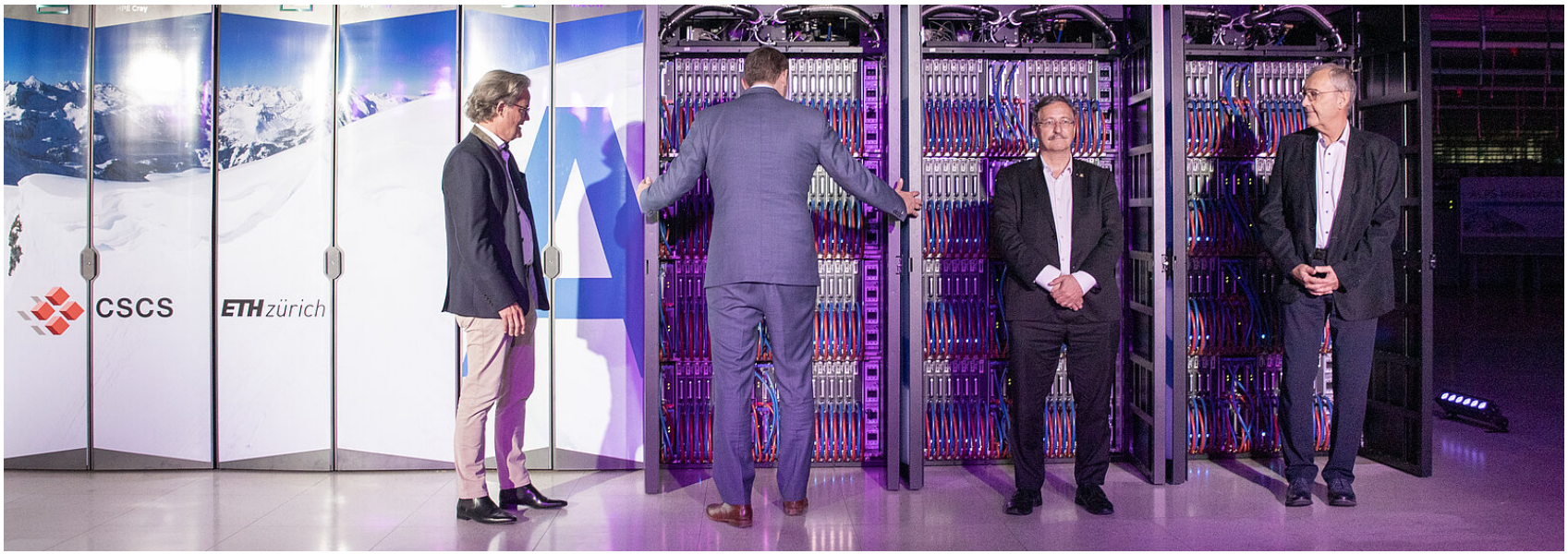
$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1 & 0 & 0 \\ 1/4 & 1/3 & 1 & 0 \\ 1/2 & 1/6 & 1/4 & 1 \end{pmatrix}, U = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 0 & 6 & 6 & 6 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`P,L,U = scipy.linalg.lu(A)` returns[1]

```
>>> P
array([[0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]])
>>> L
array([[1.        , 0.        , 0.        , 0.        ],
       [0.5       , 1.        , 0.        , 0.        ],
       [0.25      , 0.33333333, 1.        , 0.        ],
       [0.5       , 0.16666667, 0.25      , 1.        ]])
>>> U
array([[4., 4., 4., 4.],
       [0., 6., 6., 6.],
       [0., 0., 4., 4.],
       [0., 0., 0., 1.]])
```

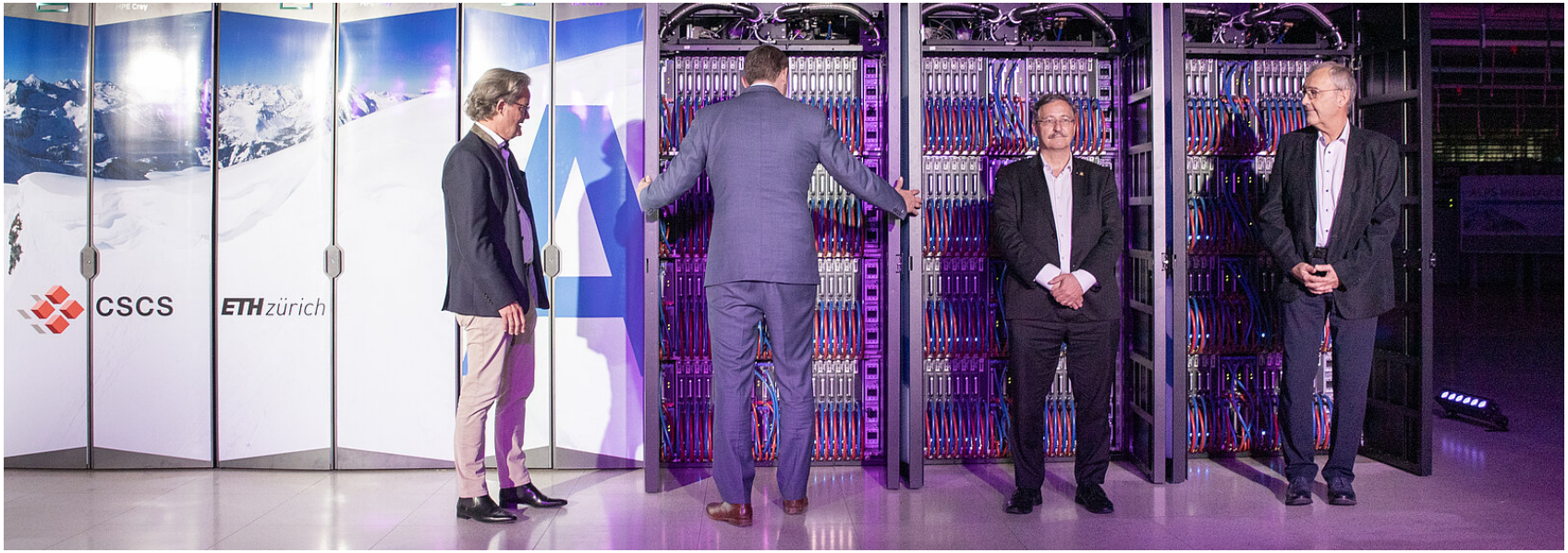
[1] By setting different options, direct computation of P can be avoided.

Fun Fact 1: Top500 list



`https://www.cscs.ch/publications/news/2024/new-research-infrastructure-alps-supercomputer-inaugurated`: ... This makes 'Alps' one of the fastest computers in the world. In the Top500 list of supercomputers from June 2024, it was ranked 6th in the first expansion stage.

Fun Fact 1: Top500 list



`https://www.cscs.ch/publications/news/2024/new-research-infrastructure-alps-supercomputer-inaugurated`: ... This makes 'Alps' one of the fastest computers in the world. In the Top500 list of supercomputers from June 2024, it was ranked 6th in the first expansion stage.

Top500 position determined by High Performance LINPACK benchmark, which applies LU to solve a (LARGE) dense system of linear equations:

$$A\mathbf{x} = \mathbf{b}.$$

Fun Fact 2: Famous open problem

- Gaussian elimination with *complete* pivoting also swaps the rows in order to ensure that the pivot is the largest element in the entire remaining matrix (not only in the column that is currently processed).
- Complete pivoting is safer than Algorithm 4.13 in terms of pivot growth.
- For problems of practical interest, Algorithm 4.13 works just fine. Virtually nobody uses Gaussian elimination with complete pivoting in favor of Algorithm 4.13, but there is an interesting open question concerning the worst-case pivot growth.