6 Lecture 6, Jan 21

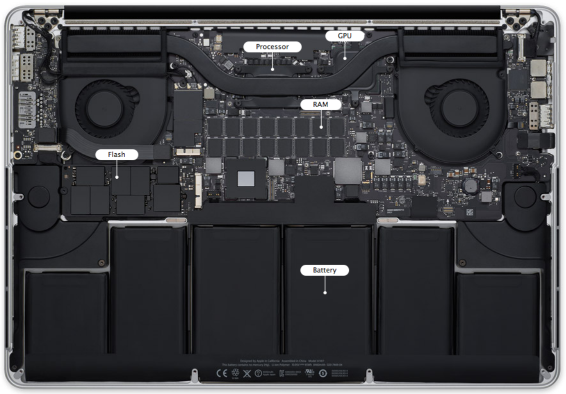
Last time

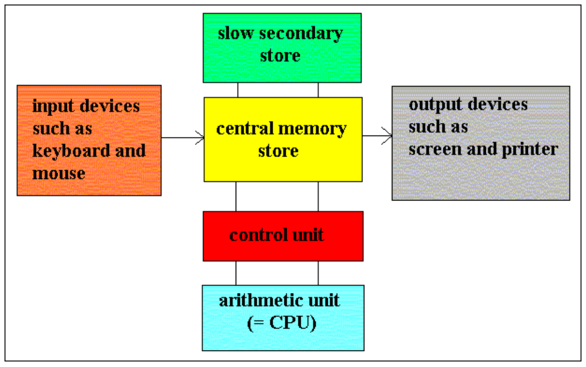
- Version control with Git.
- Algorithm, flop, computational complexity.
- Numerical linear algebra: introduction, BLAS.

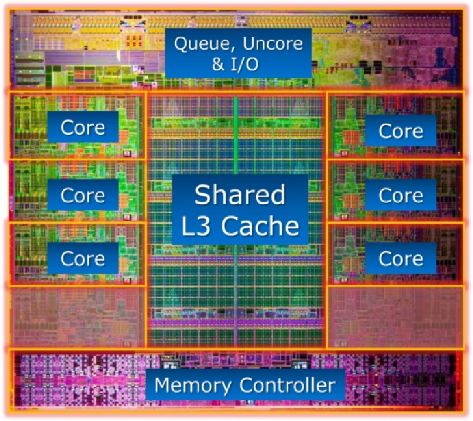
Today

- BLAS (con'td).
- Triangular systems.
- Gaussian elimination and LU decomposition.

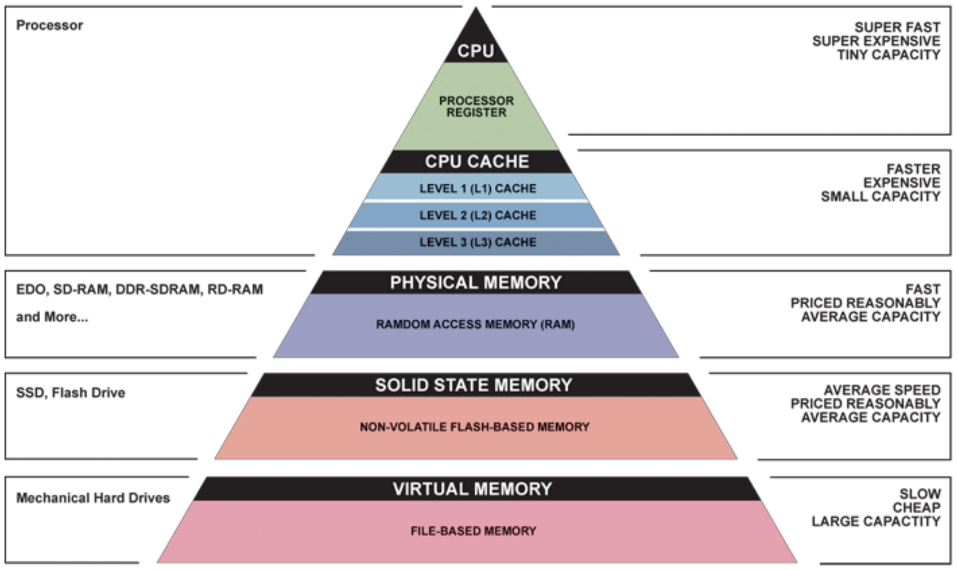
Computer architecture and high-level BLAS




slow secondary store
input devices such as keyboard and mouse
central memory store
output devices such as screen and printer
control unit
arithmetic unit (= CPU)



- Memory hierarchy:



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Upper the hierarchy, faster the memory accessing speed, and more expensive the memory units.

> Key to high performance is effective use of memory hierarchy. True on all architectures.

- Can we keep the super fast arithmetic units busy with enough deliveries of matrix data and ship the results to memory fast enough to avoid backlog? Answer: use high-level BLAS as much as possible.

- Why high-level BLAS?

BLAS	Dimension	Mem Refs	Flops	Ratio
Level 1: $\boldsymbol{y} \leftarrow \boldsymbol{y} + a\boldsymbol{x}$	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$	$3n$	$2n$	3:2
Level 2: $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A}\boldsymbol{x}$	$\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{A} \in \mathbb{R}^{n \times n}$	n^2	$2n^2$	1:2
Level 3: $\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$	$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{n \times n}$	$4n^2$	$2n^3$	2:n

- BLAS 1 tend to be *memory bandwidth-limited*. E.g., Xeon X5650 CPU has a theoretical throughput of 128 DP GFLOPS but a max memory bandwidth of 32GB/s.

- Higher level BLAS (3 or 2) make more effective use of arithmetic logic units (ALU) by keeping them busy.

- Message: Although we state many algorithms (solving linear equations, least squares, eigen-decomposition, SVD, ...) in terms of inner product and saxpy, the actual implementation may be quite different.

- A distinction between LAPACK and LINPACK is that LAPACK makes use of higher level BLAS as much as possible (usually by smart partitioning) to increase the so-called *level-3 fraction*.

Effect of data layout

- Data layout in memory effects execution speed too. It is much faster to move chunks of data in memory than retrieving/writing scattered data.

- Storage mode: column-major (FORTRAN, MATLAB, R) vs row-major (C/C++).

  When you call BLAS from C/C++, use CBLAS library instead of the traditional BLAS library implemented in Fortran.

- Take matrix multiplication as an example. Assume the storage is column-major, such as in FORTRAN. $\boldsymbol{C} \leftarrow \boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}$, where $\boldsymbol{A} \in \mathbb{R}^{m \times p}$, $\boldsymbol{B} \in \mathbb{R}^{p \times n}$, $\boldsymbol{C} \in \mathbb{R}^{m \times n}$. There are 6 variants of the algorithms according to the order in the triple loops. We pay attention to the innermost loop, where the vector calculation occurs,

```
jki or kji:        for i = 1:m
                       C(i,j) = C(i,j) + A(i,k)B(k,j)
                   end

ikj or kij:        for j = 1:n
                       C(i,j) = C(i,j) + A(i,k)B(k,j)
                   end

ijk or jik:        for k = 1:p
                       C(i,j) = C(i,j) + A(i,k)B(k,j)
                   end
```

and the associated *stride* when accessing the three matrices in memory (assuming column-major storage)

Variant	A Stride	B Stride	C Stride
jki or kji	Unit	0	Unit
ikj or kij	0	Non-Unit	Non-Unit
ijk or jik	Non-Unit	Unit	0

Apparently the variants jki or kji are preferred.

- Message: data storage mode effects algorithm implementation too.
- A numerical experiment in JULIA: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/matmul_loop.html`.

Solving linear equations

We consider algorithms for solving linear equations $\boldsymbol{Ax} = \boldsymbol{b}$, a ubiquitous task in statistics. Idea: turning original problem into an "easy" one, e.g., triangular system.

Triangular system

- *Forward substitution* to solve $\boldsymbol{Lx} = \boldsymbol{b}$, where $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ is lower triangular

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ &\vdots \\ x_m &= b_m - a_{m1}x_1 - a_{m2}x_2 - \dots - a_{m,m-1}x_{m-1})/a_{mm} \end{aligned}$$

n^2 flops ($n^2/2$ multiplications/divisions and $n^2/2$ additions/substractions) and $\boldsymbol{L}$ is accessed by row.

- *Back substitution* to solve $\boldsymbol{Ux} = \boldsymbol{b}$ where $\boldsymbol{U} \in \mathbb{R}^{n \times n}$ is upper triangular

$$\begin{bmatrix} a_{11} & \dots & a_{1,m-1} & a_{1m} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & a_{m-1,m-1} & a_{m-1,m} \\ 0 & \dots & 0 & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_{m-1} \\ b_m \end{bmatrix}$$

$$\begin{aligned} x_m &= b_m/a_{mm} \\ x_{m-1} &= (b_{m-1} - a_{m-1,m}x_m)/a_{m-1,m-1} \\ x_{m-2} &= (b_{m-2} - a_{m-2,m-1}x_{m-1} - a_{m-2,m}x_m)/a_{m-2,m-2} \\ &\vdots \\ x_1 &= b_1 - a_{12}x_2 - a_{13}x_3 - \dots - a_{1m}x_m)/a_{11} \end{aligned}$$

  n^2 flops ($n^2/2$ multiplications/divisions and $n^2/2$ additions/substractions) and $\boldsymbol{U}$ is accessed by row.

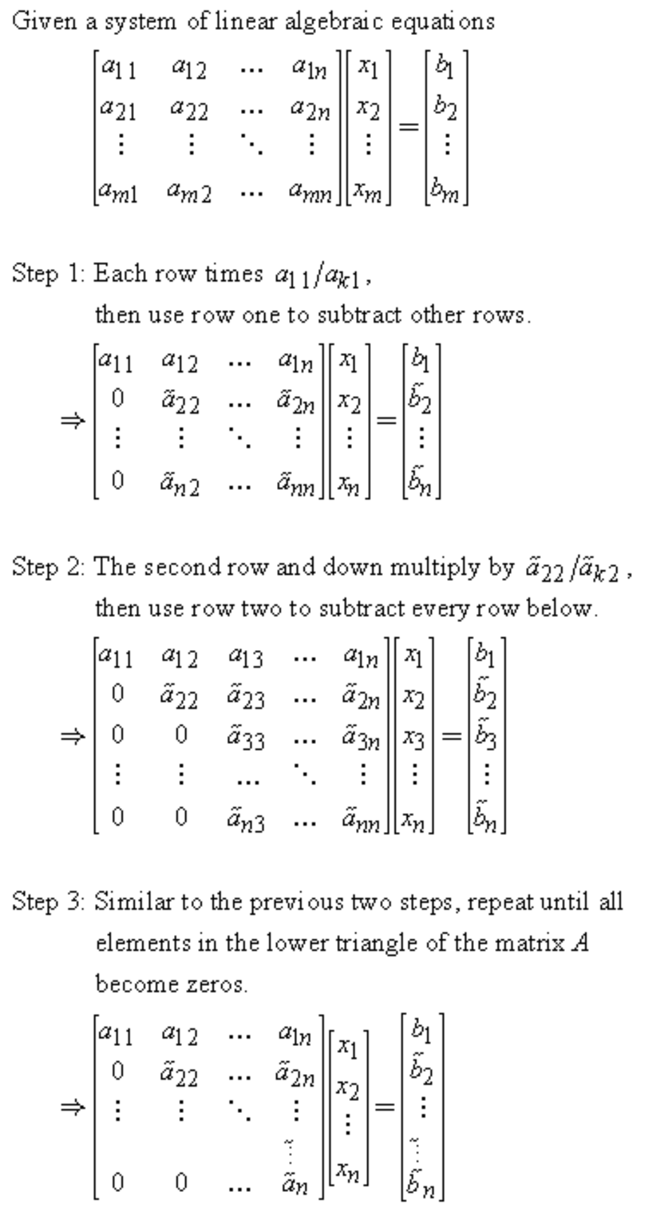
- Column version: reverse the order of looping.

- BLAS level 2 function: `?trsv` (triangular solve with one right hand side)

- BLAS level 3 function: `?trsm` (matrix triangular solve, i.e., multiple right hand sides)

- In R, `forwardsolve()` and `backsolve()` (wrappers of `dtrsm`).
- In Julia, `A \ b` uses forward or backward substitution when $\boldsymbol{A}$ is a triangular matrix. Or we can simply call BLAS functions directly.
- Eigenvalues of $\boldsymbol{L}$ are diagonal entries $\lambda_i = \ell_{ii}$. $\det(\boldsymbol{L}) = \prod_i \ell_{ii}$.
- A *unit triangular matrix* is a triangular matrix with all diagonal entries being 1.
- The algebra of triangular matrices (HW2)
  - The product of two upper (lower) triangular matrices is upper (lower) triangular.
  - The inverse of an upper (lower) triangular matrix is upper (lower) triangular.
  - The product of two unit upper (lower) triangular matrices is unit upper (lower) triangular.
  - The inverse of a unit upper (lower) triangular matrix is unit upper (lower) triangular.

Gaussian elimination and LU decomposition



Given a system of linear algebraic equations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Step 1: Each row times a_{11}/a_{k1},
then use row one to subtract other rows.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \tilde{a}_{n2} & \cdots & \tilde{a}_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{b}_n \end{bmatrix}$$

Step 2: The second row and down multiply by $\tilde{a}_{22}/\tilde{a}_{k2}$,
then use row two to subtract every row below.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \cdots & \tilde{a}_{2n} \\ 0 & 0 & \tilde{a}_{33} & \cdots & \tilde{a}_{3n} \\ \vdots & \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \tilde{a}_{n3} & \cdots & \tilde{a}_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \vdots \\ \tilde{b}_n \end{bmatrix}$$

Step 3: Similar to the previous two steps, repeat until all
elements in the lower triangle of the matrix A
become zeros.

$$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & \tilde{a}_{22} & \cdots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ & & & \tilde{\vdots} \\ 0 & 0 & \cdots & \tilde{a}_{n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{\vdots} \\ \tilde{b}_n \end{bmatrix}$$

- History: It is one by-product of Gauss's efforts to re-discover the dwarf planet Ceres using method of least squares. No linear algebra in 1800!
- Solve $\boldsymbol{Ax} = \boldsymbol{b}$ for a general matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$.
- Idea: a series of *elementary operations* that turn $\boldsymbol{A}$ into a triangular system.

- We consider the square $\boldsymbol{A}$ case first.
- *Elementary operator matrix* $\boldsymbol{E}_{jk}(c)$ is the identity with the 0 in position (j,k) replaced by c. For any vector $\boldsymbol{x}$,

$$\boldsymbol{E}_{jk}(c)\boldsymbol{x} = (x_1, \ldots, x_{j-1}, x_j + cx_k, x_{j+1}, \ldots, x_n)^\intercal.$$

  Applying $\boldsymbol{E}_{jk}(c)$ to both sides of the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ replaces the j-th equation $\boldsymbol{a}_{j*}^\intercal \boldsymbol{x} = b_j$ by $\boldsymbol{a}_{j*}^\intercal \boldsymbol{x} + c\boldsymbol{a}_{k*}^\intercal \boldsymbol{x} = b_j + cb_k$. For $j > k$, $\boldsymbol{E}_{jk}(c) = \boldsymbol{I} + c\boldsymbol{e}_j\boldsymbol{e}_k^\intercal$ is unit lower triangular and full rank. $\boldsymbol{E}_{jk}^{-1}(c) = \boldsymbol{E}_{jk}(-c)$.
- Zeroing the first column

$$\begin{aligned}
\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b} \\
\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b} \\
&\vdots \\
\boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})\boldsymbol{b}
\end{aligned}$$

  where $c_i^{(1)} = -a_{i1}/a_{11}$. Denote $\boldsymbol{M}_1 = \boldsymbol{E}_{n1}(c_n^{(1)})\cdots\boldsymbol{E}_{31}(c_3^{(1)})\boldsymbol{E}_{21}(c_2^{(1)})$.
- Then zero the k-th column for $k = 2, \ldots, n-1$ sequentially. This results in a transformed linear system $\boldsymbol{U}\boldsymbol{x} = \tilde{\boldsymbol{b}}$, where $\boldsymbol{U} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{A}$ is upper triangular and $\tilde{\boldsymbol{b}} = \boldsymbol{M}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{b}$. $\boldsymbol{M}_k$ has the shape

$$\boldsymbol{M}_k = \boldsymbol{E}_{n,k}^{(k)}\cdots\boldsymbol{E}_{k+1,k}^{(k)} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & c_{k+1}^{(k)} & 1 & & \\ & & \vdots & & \ddots & \\ & & c_n^{(k)} & & & 1 \end{pmatrix},$$

  where $c_i^{(k)} = -\tilde{a}_{ik}^{(k-1)}/\tilde{a}_{kk}^{(k-1)}$. $\boldsymbol{M}_k$ is unit lower triangular and full rank. $\boldsymbol{M}_k$ are called the *Gauss transformations*.
- Let $\boldsymbol{L} = \boldsymbol{M}_1^{-1}\cdots\boldsymbol{M}_{n-1}^{-1}$. We have the decomposition

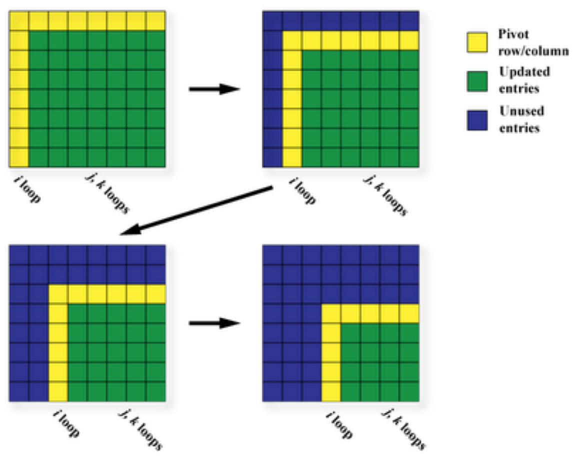
$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}.$$

  $\boldsymbol{M}_k$ is unit upper triangular, so $\boldsymbol{M}_k^{-1}$ and thus $\boldsymbol{L}$ is unit lower triangular.

- Where is $\boldsymbol{L}$? Note $\boldsymbol{M}_k = \boldsymbol{I} + (0, \ldots, 0, c_{k+1}^{(k)}, \ldots, c_n^{(k)})^\mathsf{T} \boldsymbol{e}_k^\mathsf{T}$. By Sherman-Morrison, $\boldsymbol{M}_k^{-1} = \boldsymbol{I} - (0, \ldots, 0, c_{k+1}^{(k)}, \ldots, c_n^{(k)})^\mathsf{T} \boldsymbol{e}_k^\mathsf{T}$. So the entries of $\boldsymbol{L}$ are simply $\ell_{ik} = -c_i^{(k)}$, $i > k$, the negative of multipliers in GE.

- The whole LU procedure is done in place, i.e., $\boldsymbol{A}$ is overwritten by $\boldsymbol{L}$ and $\boldsymbol{U}$.

- Implementation: outer product LU (kij loop), block outer product LU (higher level-3 fraction), Crout's algorithm (jki loop), ...



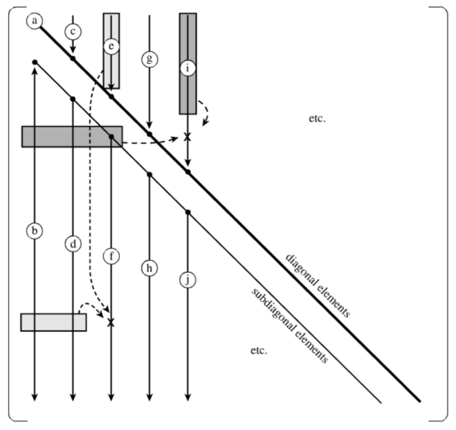


Figure 2.3.1. Crout's algorithm for *LU* decomposition of a matrix. Elements of the original matrix are modified in the order indicated by lower case letters: a, b, c, etc. Shaded boxes show the previously modified elements that are used in modifying two typical elements, each indicated by an "x".

- LU decomposition exists if the principal sub-matrix $A(1:k, 1:k)$ is non-singular for $k = 1, \ldots, n-1$. If the LU decomposition exists and $\boldsymbol{A}$ is non-singular, then the LU decomposition is unique and $\det(\boldsymbol{A}) = \prod_{i=1}^n u_{ii}$.

- This *forward elimination* or *LU decomposition* costs $2(n-1)^2 + 2(n-2)^2 + \cdots + 2 \cdot 1^2 \approx \frac{2}{3}n^3$ flops ($2n^3/3$ multiplications and $2n^3/3$ additions).

- Given LU, one right hand side costs $2n^2$ flops (one backward substitution and then forward substitution).

- For matrix inversion, there are n right hand sides $\boldsymbol{e}_i$. However, taking advantage of zeros reduces $2n^3$ flops to $\frac{4}{3}n^3$. So matrix inversion costs $\frac{2}{3}n^3 + \frac{4}{3}n^3 = 2n^3$ flops in total.

- We do *not* compute matrix inverse unless (i) it is necessary to compute standard errors, (2) number of right hand sides is much larger than n, (3) n is small.

- LU decomposition of a rectangular matrix $\boldsymbol{A} \in \mathbf{R}^{m\times n}$ exists if $\boldsymbol{A}(1:k, 1:k)$ is non-singular for $k = \min\{m, n\}$. For example,

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

and

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

Slight modification to the algorithm.

Pivoting for LU

- What if we encounter a *pivot* $\tilde{a}_{kk}^{(k-1)}$ being 0 or close to 0 due to underflow?
- Think about $\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Does it have a solution for arbitrary $\boldsymbol{b}$? Does GE work?
- Work on the example

$$\begin{aligned} 0.0001x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2, \end{aligned}$$

which has solution $x_1 = 1.0001$ and $x_2 = 0.9999$. Suppose we have 3 digits of precision. After first step of elimination, we have (catastrophic cancellation happens)

$$\begin{aligned} 0.0001x_1 + x_2 &= 1 \\ -10,000x_2 &= -10,000 \end{aligned}$$

and the solution by back substitution is $x_2 = 1.000$ and $x_1 = 0.000$.

- Message: zero or very small pivots cause trouble.
Solution: *pivoting*.
- *Partial pivoting*: at the k-th stage the equation with $\max_{i=k}^{n} |\tilde{a}_{ik}^{(k-1)}|$ is moved into the k-th row. Thus we have $\boldsymbol{M}_{n-1}\boldsymbol{P}_{n-1}\cdots\boldsymbol{M}_1\boldsymbol{P}_1\boldsymbol{A} = \boldsymbol{U}$.

- With partial pivoting, it can be shown that

$$\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U},$$

where $\boldsymbol{P} = \boldsymbol{P}_{n-1} \cdots \boldsymbol{P}_1$, $\boldsymbol{L}$ is unit lower triangular with $|\ell_{ij}| \le 1$, and $\boldsymbol{U}$ is upper triangular.

- $\det(\boldsymbol{P}) \det(\boldsymbol{A}) = \det(\boldsymbol{U}) = \prod_{i=1}^{n} u_{ii}$.

- To solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, we solve two triangular systems

$$\boldsymbol{L}\boldsymbol{y} = \boldsymbol{P}\boldsymbol{b} \quad \text{and} \quad \boldsymbol{U}\boldsymbol{x} = \boldsymbol{y},$$

costing $2n^2$ flops.

- *Complete pivoting*: Do both row and column interchanges so that the largest entry in the sub matrix $\boldsymbol{A}(k:n, k:n)$ is permuted to the (k,k)-th entry. This yields the decomposition $\boldsymbol{P}\boldsymbol{A}\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{U}$, where $|\ell_{ij}| \le 1$.

- Warning: In the actual LAPACK implementation, we do not really need to interchange rows/columns for pivoting. Just keep track of the indices we have interchanged.

- Gaussian elimination with partial pivoting is one of the most commonly used methods for solving general linear systems. Complete pivoting is stable but costs more computation. Partial pivoting is stable most of times.

- LAPACK: `?GETRF` does $\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ (LU decomposition with partial pivoting).

- In R, `solve()` implicitly performs LU decomposition (wrapper of LAPACK routine `DGESV`). `solve()` allows specifying a single or multiple right hand sides. If none, it computes the matrix inverse. The `matrix` package contains `lu()` function.