Scientific Computing I

The Solution of Moderate Size Dense Linear Systems

Martin Köhler

Computational Methods in Systems and Control Theory (CSC) Max Planck Institute for Dynamics of Complex Technical Systems

Winter Term 2024/2025

Important Preliminaries

Important Preliminaries

Theorem 7.1 (LU decomposition)

Let $A \in \mathbb{R}^{n\times n}$ and for $k = 1, \ldots, n-1$, $A_k = A(1:k, 1:k) \in \mathbb{R}^{k\times k}$ the leading $k \times k$ sub-matrix.

1. *If $\forall k = 1, \ldots, n-1$ it holds $\det(A_k) \neq 0$, then $\exists L, U \in \mathbb{R}^{n\times n}$ such that*

$$A = LU$$

with

$$L = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix} \quad \textit{(unit lower triangular)}$$

and

$$U = \begin{pmatrix} \ast & \cdots & \ast \\ & \ddots & \vdots \\ & & \ast \end{pmatrix} \quad \textit{(upper triangular).}$$

2. *If $A = LU$ exists and A is regular then the LU factorization is unique.*
3. *If $A = LU$ as in (ii) then*

$$\det(A) = u_{11} \cdots u_{nn}$$

Important Preliminaries

Note that the simple regular 2×2 matrix $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ does not allow for an LU decomposition, but applying a single row permutation we get:

$$\tilde{A} := PA = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{where } P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Important Preliminaries

Note that the simple regular 2×2 matrix $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ does not allow for an LU decomposition, but applying a single row permutation we get:

$$\tilde{A} := PA = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \text{where } P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$\rightarrow$ we need a more general formulation of the LU decomposition:

Theorem 7.2

Let $A \in \mathbb{R}^{n \times n}$ regular. There exists a permutation matrix $P \in \mathbb{R}^{n \times n}$ such that

$$PA = LU$$

for L, U as in Theorem 7.1.

Important Preliminaries

Gaussian elimination is used to compute the L and U matrices. It consists of a triple loop procedure. The straight forward row-by-row elimination version reads:

Algorithm 7.1: Gaussian Elimination "kij"-formulation

Input: $A \in \mathbb{R}^{n \times n}$
Output: A overwritten by L, U

```
1 for k = 1 : n − 1 do
2     A(k + 1 : n, k) = A(k + 1 : n, k)/A(k, k);
3     for i = k + 1 : n do
4         for j = k + 1 : n do
5             A(i, j) = A(i, j) − A(i, k)A(k, j);
```

Important Preliminaries

Gaussian elimination is used to compute the L and U matrices. It consists of a triple loop procedure. The straight forward row-by-row elimination version reads:

Algorithm 7.1: Gaussian Elimination "kij"-formulation

Input: $A \in \mathbb{R}^{n \times n}$
Output: A overwritten by L, U

1 **for** $k = 1 : n-1$ **do**
2 $A(k+1:n, k) = A(k+1:n, k)/A(k,k)$;
3 **for** $i = k+1 : n$ **do**
4 **for** $j = k+1 : n$ **do**
5 $A(i,j) = A(i,j) - A(i,k)A(k,j)$;

- There are 5 other versions how to arrange the three loops: kji, ikj, ijk, jik, jki.
- The jki version is sometimes called **left looking LU**. ↗ important for sparse matrices.

Important Preliminaries

In order to obtain a more "matrix-valued" formulation of Algorithm 7.1, we rearrange the data in a clever way:

Algorithm 7.2: Outer product Gaussian Elimination

Input: $A \in \mathbb{R}^{n \times n}$ fulfilling Theorem 7.1
Output: $L, U \in \mathbb{R}^{n \times n}$ such that $A = LU$ as in Theorem 7.1 A is overwritten by the factors.

1 **for** $k = 1 : n - 1$ **do**
2 $\quad \text{rows} = k + 1 : n;$
3 $\quad A(\text{rows}, k) = A(\text{rows}, k)/A(k, k);$
4 $\quad A(\text{rows}, \text{rows}) = A(\text{rows}, \text{rows}) - A(\text{rows}, k)A(k, \text{rows});$

Important Preliminaries

In order to obtain a more "matrix-valued" formulation of Algorithm 7.1, we rearrange the data in a clever way:

Algorithm 7.2: Outer product Gaussian Elimination

Input: $A \in \mathbb{R}^{n \times n}$ fulfilling Theorem 7.1
Output: $L, U \in \mathbb{R}^{n \times n}$ such that $A = LU$ as in Theorem 7.1 A is overwritten by the factors.

1 **for** $k = 1 : n - 1$ **do**
2 $\quad$ rows$= k + 1 : n$;
3 $\quad$ $A(\text{rows}, k) = A(\text{rows}, k)/A(k, k)$;
4 $\quad$ $A(\text{rows}, \text{rows}) = A(\text{rows}, \text{rows}) - A(\text{rows}, k)A(k, \text{rows})$;

Algorithm 7.2 is a rank-1 update, i.e., BLAS Level 2 operation formulation of the Gaussian elimination process. It involves $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ flops.

Important Preliminaries

If we want to solve a linear system $Ax = b$ with the help of the LU decomposition, we end up with the following algorithm:

Algorithm 7.3: Linear System solver using Gaussian Elimination and forward/backward substitution

---

Input: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n$
Output: $x \in \mathbb{R}^n$

1 Compute L, U as in Theorem 7.1, such that $A = LU$ (e.g. via Algorithm 7.2);
2 Solve $Ly = b$ by forward substitution;
3 Solve $Ux = y$ by backward substitution;

Cache/BLAS Exploitation

Triangular Systems

Cache/BLAS Exploitation

Triangular Systems

Consider

$$\left.\begin{aligned} a_{11}x_1 &= b_1, \\ a_{21}x_1 + a_{22}x_2 &= b_2. \end{aligned}\right\} \quad \leftrightarrow \quad \underbrace{\begin{bmatrix} a_{11} & \\ a_{21} & a_{22} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}}_{b}$$

In case $a_{11} \neq 0$ and $a_{22} \neq 0$ this leads to

$$x_1 = \frac{b_1}{a_{11}},$$
$$x_2 = \frac{b_2 - a_{21}x_1}{a_{22}} = \frac{b_2 - \frac{a_{21}}{a_{11}}b_1}{a_{22}}$$

In the i-th equation in a system $Lx = b$ in Algorithm 7.3 we find:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij}x_j}{l_{ii}} \overset{L \text{ unit diagonal}}{=} b_i - \sum_{j=1}^{i-1} l_{ij}x_j$$

Cache/BLAS Exploitation

Triangular Systems

Algorithm 7.4: Forward Substitution (Row Version)

Input: $L \in \mathbb{R}^{n \times n}$ (unit) lower triangular, $b \in \mathbb{R}^n$
Output: $y = L^{-1}b$ (stored in b)

1 $b(1) = \frac{b(1)}{L(1,1)}$;
2 **for** $i = 2 : n$ **do**
3 $\quad b(i) = \frac{b(i) - L(i,1:i-1)b(1:i-1)}{L(i,i)}$

Cache/BLAS Exploitation

Triangular Systems

Algorithm 7.4: Forward Substitution (Row Version)

Input: $L \in \mathbb{R}^{n \times n}$ (unit) lower triangular, $b \in \mathbb{R}^n$
Output: $y = L^{-1}b$ (stored in b)

1 $b(1) = \frac{b(1)}{L(1,1)}$;
2 **for** $i = 2 : n$ **do**
3 $\quad b(i) = \frac{b(i) - L(i,1:i-1)b(1:i-1)}{L(i,i)}$

- complexity: $\mathcal{O}(n^2)$ flops.
- the rounding error in each element x_i of the solution vector is smaller than $n \cdot u$.
- row-wise access to $L \leftrightarrow$ column major storage ↯

Cache/BLAS Exploitation

Triangular Systems

Algorithm 7.5: Forward Substitution (Column Version)

Input: $L \in \mathbb{R}^{n \times n}$ (unit) lower triangular, $b \in \mathbb{R}^n$
Output: $y = L^{-1}b$ (stored in b)

1 **for** $j = 1 : n-1$ **do**
2 $\quad b(j) = \frac{b(j)}{L(j,j)}$;
3 $\quad b(j+1:n) = b(j+1:n) - b(j)L(j+1:n, j)$;
4 $b(n) = \frac{b(n)}{L(n,n)}$;

Cache/BLAS Exploitation

Triangular Systems

Algorithm 7.5: Forward Substitution (Column Version)

Input: $L \in \mathbb{R}^{n \times n}$ (unit) lower triangular, $b \in \mathbb{R}^n$
Output: $y = L^{-1}b$ (stored in b)

1 **for** $j = 1 : n-1$ **do**
2 $\quad b(j) = \frac{b(j)}{L(j,j)}$;
3 $\quad b(j+1:n) = b(j+1:n) - b(j)L(j+1:n, j)$;
4 $b(n) = \frac{b(n)}{L(n,n)}$;

- only column-wise access to L
- can be built on top of **axpy** operations.
- with a unit diagonal L: only **axpy** operations.

Cache/BLAS Exploitation

Triangular Systems

Algorithm 7.5: Forward Substitution (Column Version)

Input: $L \in \mathbb{R}^{n \times n}$ (unit) lower triangular, $b \in \mathbb{R}^n$
Output: $y = L^{-1}b$ (stored in b)

1 **for** $j = 1 : n-1$ **do**
2 $\quad b(j) = \frac{b(j)}{L(j,j)}$;
3 $\quad b(j+1:n) = b(j+1:n) - b(j)L(j+1:n, j)$;
4 $b(n) = \frac{b(n)}{L(n,n)}$;

- only column-wise access to L
- can be built on top of **axpy** operations.
- with a unit diagonal L: only **axpy** operations.

$\rightarrow$ "backward" substitution for $Ux = b$ works in the same way

Triangular Systems with Multiple Right Hand Sides and BLAS Level 3 formulation

Cache/BLAS Exploitation

Triangular Systems with Multiple Right Hand Sides and BLAS Level 3 formulation

Let $B \in \mathbb{R}^{n \times q}$ leading to a family of linear systems $LX = B$ with $X \in \mathbb{R}^{n \times q}$. L is (unit) lower triangular and we consider the block substructure as in

$$\begin{bmatrix} L_{11} & 0 & \cdots & 0 \\ L_{21} & L_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}. \tag{1}$$

Cache/BLAS Exploitation

Triangular Systems with Multiple Right Hand Sides and BLAS Level 3 formulation

Let $B \in \mathbb{R}^{n \times q}$ leading to a family of linear systems $LX = B$ with $X \in \mathbb{R}^{n \times q}$. L is (unit) lower triangular and we consider the block substructure as in

$$\begin{bmatrix} L_{11} & 0 & \cdots & 0 \\ L_{21} & L_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}. \tag{1}$$

We apply Algorithm 7.5 with the $L(1,1)$ element replaced by the L_{11} block to get

$$\begin{bmatrix} L_{22} & 0 & \cdots & 0 \\ L_{32} & L_{33} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N2} & L_{N3} & \cdots & L_{NN} \end{bmatrix} \begin{bmatrix} X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} B_2 - L_{21}X_1 \\ B_3 - L_{31}X_1 \\ \vdots \\ B_N - L_{N1}X_1 \end{bmatrix}$$

after computing X_1 from $L_{11}X_1 = B_1$ by Algorithm 7.5.

Cache/BLAS Exploitation

Triangular Systems with Multiple Right Hand Sides and BLAS Level 3 formulation

Now, successively, continuing with $L_{22}X_2 = \tilde{B}_2$ and so forth, we derive the block forward elimination scheme given in Algorithm 7.6:

Algorithm 7.6: Block Forward Substitution

Input: L, B as in (1)
Output: X solving $LX = B$

1 **for** $j = 1 : N$ **do**
2 Solve $L_{jj}X_j = B_j$ for X_j;
3 **for** $i = j + 1 : N$ **do**
4 $B_i = B_i - L_{ij}X_j$

Cache/BLAS Exploitation

Triangular Systems with Multiple Right Hand Sides and BLAS Level 3 formulation

Now, successively, continuing with $L_{22}X_2 = \tilde{B}_2$ and so forth, we derive the block forward elimination scheme given in Algorithm 7.6:

Algorithm 7.6: Block Forward Substitution

Input: L, B as in (1)
Output: X solving $LX = B$

1 **for** $j = 1 : N$ **do**
2 Solve $L_{jj}X_j = B_j$ for X_j;
3 **for** $i = j + 1 : N$ **do**
4 $B_i = B_i - L_{ij}X_j$

- scalar/vector updates replaced by **GEMM** operations
- block size can be tuned w.r.t the hardware/CPU.
- similar for the "backward" substitution

BLAS Level 3 based Gaussian Elimination

Cache/BLAS Exploitation

BLAS Level 3 based Gaussian Elimination

The block formulation for the forward/backward substitution raises the obvious question:

Can we do something similar for the Gaussian elimination process?

Cache/BLAS Exploitation

BLAS Level 3 based Gaussian Elimination

The block formulation for the forward/backward substitution raises the obvious question:

Can we do something similar for the Gaussian elimination process?

We take the outer product Gaussian elimination in Algorithm 7.2 and , To this end, let $A \in \mathbb{R}^{n \times n}$ with partitioning

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{2}$$

Here $A_{11} \in \mathbb{R}^{r \times r}$, $A_{12} \in \mathbb{R}^{r \times (n-r)}$, $A_{21} \in \mathbb{R}^{(n-r) \times r}$, $A_{22} \in \mathbb{R}^{(n-r) \times (n-r)}$, for a blocking parameter $1 \leqslant r \leqslant n$.

Cache/BLAS Exploitation

BLAS Level 3 based Gaussian Elimination

Now we can compute $A_{11} = L_{11}U_{11}$, e.g., using Algorithm 7.2 and solve the triangular systems

$$L_{11}U_{12} = A_{12} \text{ for } U_{12},$$
$$L_{21}U_{11} = A_{21} \text{ for } L_{21}.$$

Then it follows:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & \tilde{A}_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{bmatrix},$$

where

$$\tilde{A}_{22} = A_{22} - L_{21}U_{12}. \tag{3}$$

Now if $\tilde{A}_{22} = L_{22}U_{22}$ were the LU of the updated $(2,2)$ block, then

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

Since we did not post special assumptions on the matrix A in Equation (2) other than the existence of the LU-decomposition, we can proceed with $\tilde{A}_{22}$ as above.

Cache/BLAS Exploitation

BLAS Level 3 based Gaussian Elimination

Algorithm 7.7: Panel Outer Product LU

Input: $A \in \mathbb{R}^{n \times n}$ as in Theorem 7.1, $r \geqslant 1$ block size
Output: $A = LU$ with L, U stored in A

1 $k = 1$;
2 **while** $k \leqslant n$ **do**
3 $\quad l = \min(n, k + r - 1)$;
4 $\quad$ Compute $A(k : l, k : l) = \tilde{L}\tilde{U}$ via Algorithm 7.2;
5 $\quad$ Solve $\tilde{L}Z = A(k : l, l + 1 : n)$ and store Z;
6 $\quad$ Solve $W\tilde{U} = A(l + 1 : n, k : l)$ and store W;
7 $\quad$ Perform the rank-r update: $A(l + 1 : n, l + 1 : n) = A(l + 1 : n, l + 1 : n) - WZ$;
8 $\quad k = l + 1$;

Cache/BLAS Exploitation

BLAS Level 3 based Gaussian Elimination

Algorithm 7.7: Panel Outer Product LU

Input: $A \in \mathbb{R}^{n \times n}$ as in Theorem 7.1, $r \geqslant 1$ block size
Output: $A = LU$ with L, U stored in A

1 $k = 1$;
2 **while** $k \leqslant n$ **do**
3 $\quad l = \min(n, k + r - 1)$;
4 $\quad$ Compute $A(k : l, k : l) = \tilde{L}\tilde{U}$ via Algorithm 7.2;
5 $\quad$ Solve $\tilde{L}Z = A(k : l, l + 1 : n)$ and store Z;
6 $\quad$ Solve $W\tilde{U} = A(l + 1 : n, k : l)$ and store W;
7 $\quad$ Perform the rank-r update: $A(l + 1 : n, l + 1 : n) = A(l + 1 : n, l + 1 : n) - WZ$;
8 $\quad k = l + 1$;

- takes $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ flops
- r is optimized w.r.t. the CPU and the memory hierarchy, typically $128 \leqslant r \leqslant 384$.

Pivoted LU Decomposition

We still have the restriction of Theorem 7.1. How to overcome this issue?

We still have the restriction of Theorem 7.1. How to overcome this issue?

From Theorem 7.2, we know that as long as $A \in \mathbb{R}^{n \times n}$ is regular, we can find a permutation matrix P such that

$$PA = LU.$$

We still have the restriction of Theorem 7.1. How to overcome this issue?

From Theorem 7.2, we know that as long as $A \in \mathbb{R}^{n \times n}$ is regular, we can find a permutation matrix P such that

$$PA = LU.$$

Idea: Swap the rows in A on-the-fly, such that $A(k,k) \neq 0$ for $k = 1, \ldots, n$.

Definition 7.3

A matrix $P \in \mathbb{R}^{n \times n}$ is called permutation matrix, iff it only has a one in each row and column.

- The product PA swaps the rows of the matrix A.
- The product AP swaps the columns of the matrix A.

Cache/BLAS Exploitation

Pivoted LU Decomposition

For a given step k we can select the row l for interchanging within $k \leqslant l \leqslant n$.

- $A(l,k) \neq 0$
- dividing by $A(l,k)$ must be "numerically" safe, i.e. coefficients $A(\cdot,k)/A(l,k)$ should be as small as possible to avoid overflows

Cache/BLAS Exploitation

Pivoted LU Decomposition

For a given step k we can select the row l for interchanging within $k \leqslant l \leqslant n$.

- $A(l,k) \neq 0$
- dividing by $A(l,k)$ must be "numerically" safe, i.e. coefficients $A(\cdot,k)/A(l,k)$ should be as small as possible to avoid overflows

Solution

We select l such that $l = \text{argmax}_{k \leqslant l \leqslant n} |A(l,k)|$.

If $A(l,k) = 0$ for some k, the matrix A is singular.

Cache/BLAS Exploitation

Pivoted LU Decomposition

For a given step k we can select the row l for interchanging within $k \leqslant l \leqslant n$.

- $A(l,k) \neq 0$
- dividing by $A(l,k)$ must be "numerically" safe, i.e. coefficients $A(\cdot,k)/A(l,k)$ should be as small as possible to avoid overflows

Solution

We select l such that $l = \text{argmax}_{k \leqslant l \leqslant n} |A(l,k)|$.

If $A(l,k) = 0$ for some k, the matrix A is singular.

Remark 1

The element $A(l,k)$ is called **pivot element** in step k.

Cache/BLAS Exploitation

Pivoted LU Decomposition

For a given step k we can select the row l for interchanging within $k \leqslant l \leqslant n$.

- $A(l,k) \neq 0$
- dividing by $A(l,k)$ must be "numerically" safe, i.e. coefficients $A(\cdot,k)/A(l,k)$ should be as small as possible to avoid overflows

Solution

We select l such that $l = \text{argmax}_{k \leqslant l \leqslant n} |A(l,k)|$.

If $A(l,k) = 0$ for some k, the matrix A is singular.

Remark 1

The element $A(l,k)$ is called **pivot element** in step k.

Remark 2

Selecting the pivot element only form the current column (or current row) is called **partial pivoting**.

Cache/BLAS Exploitation

Pivoted LU Decomposition

Integrating this approach in Algorithm 7.2 leads to

Algorithm 7.8: Outer product Gaussian Elimination with Partial Pivoting

Input: $A \in \mathbb{R}^{n \times n}$ fulfilling Theorem 7.2

Output: $P, L, U \in \mathbb{R}^{n \times n}$ such that $PA = LU$ as in Theorem 7.2 A is overwritten by the factors.

```
1 P=I;
2 for k = 1 : n − 1 do
3     Determine l, k ⩽ l ⩽ n such that |A(l, k)| = ||A(k : n, k)||_∞;
4     A(l, k : n) ↔ A(k, k : n), P(l, k : n) ↔ P(k, k : n);
5     rows= k + 1 : n;
6     A(rows, k) = A(rows, k)/A(k, k);
7     A(rows, rows) = A(rows, rows) − A(rows, k)A(k, rows);
```

Cache/BLAS Exploitation

Pivoted LU Decomposition

Now, this needs to be reformulated, such that it works on block of size r.

Cache/BLAS Exploitation

Pivoted LU Decomposition

Now, this needs to be reformulated, such that it works on block of size r.

$\rightarrow$ Instead of using the A_{11} in the first step, we have perform the pivoted LU step on

$$P_1 \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}.$$

Then the permutation P_1 needs to be applied to the remaining matrix

$$\begin{bmatrix} \tilde{A}_{12} \\ \tilde{A}_{22} \end{bmatrix} = P_1 \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$$

and compute U_{12} by solving

$$L_{11} U_{12} = \tilde{A}_{12}.$$

Finally, we update

$$\tilde{A} = \tilde{A}_{22} - L_{21} U_{12}$$

and resume the procedure on $\tilde{A}$ again.

Cache/BLAS Exploitation

Pivoted LU Decomposition

Algorithm 7.9: Panel Outer Product LU with Partial Pivoting

Input: $A \in \mathbb{R}^{n \times n}$ as in Theorem 7.2, $r \geqslant 1$ block size
Output: $PA = LU$ with L, U stored in A, P permutation matrix

1 $k = 1$;
2 $P = I$;
3 **while** $k \leqslant n$ **do**
4 $\quad l = \min(n, k + r - 1)$;
5 $\quad$ Compute $P_* A(k : n, k : l) = \begin{bmatrix} \tilde{L}_1 \\ \tilde{L}_2 \end{bmatrix} \tilde{U}$ via Algorithm 7.9;
6 $\quad P(k : n, k : n) \leftarrow P_* P(k : n, k : n)$;
7 $\quad$ Solve $\tilde{L}_1 Z = A(k : l, l + 1 : n)$ and store Z;
8 $\quad$ Perform the rank-r update: $A(l + 1 : n, l + 1 : n) = A(l + 1 : n, l + 1 : n) - \tilde{L}_2 Z$;
9 $\quad k = l + 1$;

Cache/BLAS Exploitation

Pivoted LU Decomposition

Algorithm 7.9: Panel Outer Product LU with Partial Pivoting

Input: $A \in \mathbb{R}^{n \times n}$ as in Theorem 7.2, $r \geqslant 1$ block size
Output: $PA = LU$ with L, U stored in A, P permutation matrix

1 $k = 1$;
2 $P = I$;
3 **while** $k \leqslant n$ **do**
4 $\quad l = \min(n, k + r - 1)$;
5 $\quad$ Compute $P_* A(k : n, k : l) = \begin{bmatrix} \tilde{L}_1 \\ \tilde{L}_2 \end{bmatrix} \tilde{U}$ via Algorithm 7.9;
6 $\quad P(k : n, k : n) \leftarrow P_* P(k : n, k : n)$;
7 $\quad$ Solve $\tilde{L}_1 Z = A(k : l, l + 1 : n)$ and store Z;
8 $\quad$ Perform the rank-r update: $A(l + 1 : n, l + 1 : n) = A(l + 1 : n, l + 1 : n) - \tilde{L}_2 Z$;
9 $\quad k = l + 1$;

$\rightarrow$ the pivoted LU decomposition requires $\frac{2}{3} n^3 + \mathcal{O}(n^2)$ flops, as well.

Cache/BLAS Exploitation

Pivoted LU Decomposition

Theorem 7.4

For a matrix $A \in \mathbb{R}^{n \times n}$ the exact number of flops for the LU is

$$\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n.$$

This is invariant under the choice of the parameter r.

Complete Pivoting

Cache/BLAS Exploitation

Complete Pivoting

The partial pivoting search only in one direction for a pivot element.

Cache/BLAS Exploitation

Complete Pivoting

The partial pivoting search only in one direction for a pivot element.

The pivot element could be selected from the whole $A(k:n, k:n)$ submatrix. This leads to the **pivot element** $A(l, j)$, fulfilling

$$(l, j) = \underset{k \leqslant l \leqslant n,\ k \leqslant j \leqslant n}{\operatorname{argmax}} |A_(l, j)|,$$

which is called **complete pivoting**.

Remark 3

Since we search in two dimensions for the pivot element, we have to permute rows and columns of A, thus our LU decomposition changes to

$$PAQ = LU,$$

where P and Q are permutation matrices.

Cache/BLAS Exploitation

Complete Pivoting

Remark 4

- The LU decomposition without pivoting is not numerically stable.
- The LU decomposition with partial or complete pivoting is numerically stable.
- The complete pivoting strategy is only used in rare cases, where a the partial pivoting might fail.
- The complete pivoting strategy could not be implemented in a practicable Level-3 enabled algorithm.

Iterative Refinement

Iterative Refinement

What to do, if the solution of a linear system (with LU decomposition) is still inaccurate?

Motivation

Iterative Refinement

Motivation

We assume that

- $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ and $\hat{x}$ is the computed solution of $Ax = b$,
- $r \in \mathbb{R}^n$, $r = b - A\hat{x}$ is the residual.

Now we compute $d = A^{-1}r$ (or solve $Ad = r$) and calculate $\tilde{x} = \hat{x} + d$. Using exact arithmetic, we obtain

$$A\tilde{x} = A(\hat{x} + d) = A\hat{x} + Ad = (b - r) + AA^{-1}r = b - r + r = b$$

Thus in exact arithmetic the updated $\tilde{x}$ would be the exact solution.

Since we **cannot** use exact arithmetic, we have to repeat this procedure.

Iterative Refinement

Motivation

Algorithm 7.10: iterative refinement

Input: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $\hat{x}$ an approximate solution
Output: $\hat{x}$ a solution (approximation)

1 **repeat**
2 $\quad r = b - A\hat{x}$;
3 $\quad$ solve $Ad = r$;
4 $\quad$ update $\hat{x} = \hat{x} + d$
5 **until** $\hat{x}$ **accurate enough**;

Iterative Refinement

Motivation

The literature distinguishes mainly 2 approaches:

1. fixed precision refinement
2. mixed precision refinement

Iterative Refinement

Motivation

The literature distinguishes mainly 2 approaches:

1. fixed precision refinement
2. mixed precision refinement

Fixed Precision Iterative Refinement

All steps in Algorithm 7.10 are computed in the same precision (u) and the same datatype.

Iterative Refinement

Motivation

The literature distinguishes mainly 2 approaches:

1. fixed precision refinement
2. mixed precision refinement

Fixed Precision Iterative Refinement

All steps in Algorithm 7.10 are computed in the same precision (u) and the same datatype.

Mixed Precision Iterative Refinement

For mixed precision refinement the residual r is computed in a higher precision ($\hat{u}$). Classically $\hat{u} = u^2$, i.e., u corresponds to single precision, and $\hat{u}$ then stands for double precision.

Iterative Refinement

Motivation

Example 7.5

The LAPACK routine **`XGERFS`** (**`X`**=c,d,s,z) solves a linear system with fixed precision iterative refinement. Furthermore, it provides estimates for the forward- and the backward-error, which appears during the solution if the liear system.

Iterative Refinement

Motivation

Example 7.5

The LAPACK routine **XGERFS** (**X**=c,d,s,z) solves a linear system with fixed precision iterative refinement. Furthermore, it provides estimates for the forward- and the backward-error, which appears during the solution if the liear system.

Example 7.6

The LAPACK routine **DSGESV** solves a linear systems with the help of mixed precision iterative refinement.

- the LU decomposition is done in single precision.
- solving $Ad = r$ is done in single precision,
- the residual $r = b - A\hat{x}$ and the update $\hat{x} = \hat{x} + d$ are performed in double precision.

Theoretic Background

Iterative Refinement

Theoretic Background

Let $A \in \mathbb{R}^{n \times n}$ be a square matrix. The absolute value of A is defined component-wise:

$$|A| = (|a_{ij}|)_{i,j=1,\ldots,n}.$$

Under the assumption

$$(A + \Delta A)\hat{x} = b \quad |\Delta A| \leqslant \mathrm{u}w \tag{4}$$

for W non-negative depending on A, n, and u (but not on b) one can prove the following two theorems:

Iterative Refinement

Theoretic Background

Theorem 7.7 (Mixed Precision Refinement)

Let $Ax = b$ be a non-singular linear system solved with a method satisfying (4) and residuals in double the working precision. Moreover

$$\eta = u \left\| |A^{-1}| \left(|A| + w \right) \right\|_\infty$$

If $\eta < 1 - \delta$ for δ large enough, then iterative refinement reduces the forward error by approximately a factor of η at each stage until

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \approx \mathsf{u}$$

Iterative Refinement

Theoretic Background

Theorem 7.8 (Fixed Precision Refinement)

Setting as in Theorem 7.7 but with residual computation in working precision. The same reduction holds, but with limit

$$\frac{\|x-\hat{x}\|_\infty}{\|x\|_\infty} \leqslant 2nu \underbrace{\frac{\left\||A^{-1}||A||x|\right\|_\infty}{\|x\|_\infty}}_{\mathrm{cond}(A,x)} \tag{5}$$

Iterative Refinement

Theoretic Background

Remark 5

- Equation (5) is essentially the best we can expect in fixed precision.
- Note that the solver need not be of LU type and $\hat{u}$ is not limited to u^2.
- When working in $\hat{u} = u^2$, i.e., system solves in single precision and residual in double precision, one can reuse the LU decomposition from the outer solve. That means the iterative refinement is of $\mathcal{O}(n^2)$ complexity, i.e., one order of magnitude cheaper than the actual solve and the amount of data copied is reduced due to single precision storage.
- Fixed precision iterative refinement may be used to stabilize unstable solvers for $Ax = b$, e.g., $LU = PA$ computed with poor pivoting.
- **rule of thumb:**
  machine precision: $10^{-d} = u$, $\kappa_\infty(A) \approx 10^q \rightsquigarrow k$ steps of mixed precision refinement lead to approximately $\min(d, k(d-q))$ correct digits in x.

Iterative Refinement as Splitting Method

Iterative Refinement

Iterative Refinement as Splitting Method

Splitting methods are iterative solvers for linear systems, that focus on an additive decomposition of the matrix A and compute a sequence x_i converging to x by

$$x_{i+1} = x_i + M(b, x_i),$$

where $M(\cdot, \cdot)$ is a mapping $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ which approximates the solution with the use of the additive decomposition.

Iterative Refinement

Iterative Refinement as Splitting Method

Splitting methods are iterative solvers for linear systems, that focus on an additive decomposition of the matrix A and compute a sequence x_i converging to x by

$$x_{i+1} = x_i + M(b, x_i),$$

where $M(\cdot, \cdot)$ is a mapping $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ which approximates the solution with the use of the additive decomposition.

We decompose A into $A = B + (A - B)$, this yields

$$\begin{aligned} \Rightarrow Ax = b &\Leftrightarrow B^{-1}(B + (A - B))x = B^{-1}b \Leftrightarrow (I + (B^{-1}A - I))x = B^{-1}b \\ &\Leftrightarrow x + (B^{-1}A - I)x = B^{-1}b \\ \rightsquigarrow x_{i+1} &= B^{-1}b - (B^{-1}A - I)x_i \qquad (*) \\ &= x_i + \underbrace{B^{-1}\underbrace{(b - Ax_i)}_{r_i}}_{d_i} \end{aligned}$$

Iterative Refinement

Iterative Refinement as Splitting Method

By setting

$$B = \hat{L}\hat{U} \quad \rightarrow \quad B^{-1} = (\hat{L}\hat{U})^{-1},$$

this reflects a refinement of the LU. From (*) we immediately find

$$x_{i+1} = B^{-1}b + B^{-1}(B - A)x_i.$$

As for the splitting methods in general, by the Banach fixed point theorem we then have that the iteration converges if $M := B^{-1}(B - A)$ is a contraction, i.e. $\rho(M) < 1$.