

Advanced algorithms

computer arithmetic:
number encodings and operations,
LUP decomposition, finding inverse matrix

Jiří Vyskočil, Radek Mařík

2011

Natural Number Encodings

- The most common representation of natural numbers is the following binary encoding:

$$\text{value of a number} = \sum_{i=0}^n b_i \times 2^i$$

where n is a number of bits of the number and b_i is a value of i -th bit.

- BCD (Binary Coded Decimal) representation with each decimal digit represented by its own four-bit binary sequence (nibble)
 - It is not as effective as the previous representation (all combinations of binary bit sequences are not used)
 - BCD format are still important and continue to be used in financial, commercial, and industrial computing.

Integer Number Encodings

- complement representation of negative numbers (the most common):

$$\text{value of a number } N = \begin{cases} \sum_{i=0}^{n-1} b_i \times 2^i & \text{for } b_n = 0, \text{ thus } N \in \langle 0; 2^{n-1} - 1 \rangle \\ -1 - \sum_{i=0}^{n-1} (1 - b_i) \times 2^i & \text{for } b_n = 1, \text{ thus } N \in \langle -2^{n-1}; -1 \rangle \end{cases}$$

most-significant bit								
0	1	1	1	1	1	1	1	= 127
0	1	1	1	1	1	1	0	= 126
0	0	0	0	0	0	1	0	= 2
0	0	0	0	0	0	0	1	= 1
0	0	0	0	0	0	0	0	= 0
1	1	1	1	1	1	1	1	= -1
1	1	1	1	1	1	1	0	= -2
1	0	0	0	0	0	0	1	= -127
1	0	0	0	0	0	0	0	= -128

8-bit two's-complement integers

- For additions and subtractions we can use the same algorithms as for the previous binary numbers representation of natural numbers.
- +/- sign can be detected from the most-significant bit.
- There is only encoding for zero.

Floating-point Data and Encodings

- representation:

- $s \times \frac{c}{b^{p-1}} \times b^e$

where

s is the sign (signum +/-)

c is the significand (fraction)

b is the base (typically 2 or 10)

p is the precision (the number of digits in the significand)

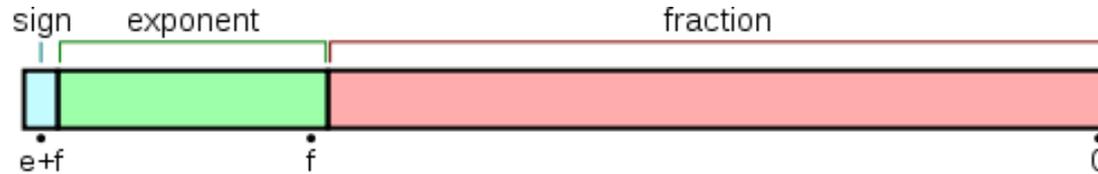
e is the integer exponent

- We want to encode also $+\infty$ a $-\infty$.

- If $b=2$ (the most common case) then there can arise some problems when inputs and outputs are converted from/to decimal base.

Floating-point Data and Encodings

- IEEE 754:



type	exponent field	significand (fraction field)
+/- zero	0	0
denormalized numbers	0	non zero
normalized numbers	$1 \text{ až } 2^e - 2$	any
+/- ∞	$2^e - 1$	0
NaN (Not a Number)	$2^e - 1$	non zero

- normalized value:

- value = $(-1)^{\text{sign}} \times 2^{\text{exponent-exponent bias}} \times (1.\text{fraction})$

- denormalized value:

- value = $(-1)^{\text{sign}} \times 2^{\text{exponent-exponent bias}+1} \times (0.\text{fraction})$

Floating-point Data and Encodings

■ IEEE 754:

- **NaN** (Not a Number) is used for encodings of numbers that were a result of arithmetical operations with nonstandard inputs:
 - operations with a NaN as at least one operand
 - the divisions: $0/0$, ∞/∞ , $\infty/-\infty$, $-\infty/\infty$, and $-\infty/-\infty$
 - the multiplications: $0 \times \infty$ and $0 \times -\infty$
 - the additions: $\infty + (-\infty)$, $(-\infty) + \infty$ and equivalent subtractions
 - calling functions with arguments out of its domain:
 - the square root of a negative number
 - the logarithm of a negative number
 - trigonometric functions ...
- NaNs have two types:
 - **Quiet** (qNaN)
 - do not raise any additional exceptions as they propagate through most operations)
 - **Signalling** (sNaN)
 - should raise an invalid exception as underflow or overflow).
- NaNs may also be explicitly assigned to variables, typically as a representation for missing values.

Differences Between Computer and Standard Arithmetic

- in both worlds (computer and standard arithmetic) holds:
 - $1 \cdot x = x$
 - $x \cdot y = y \cdot x$
 - $x + x = 2 \cdot x$
- in computer arithmetic needs not hold:
 - $x \cdot (1/x) = 1$
 - $(1 + x) - 1 = x$
 - $(x + y) + z = x + (y + z)$
- a common programmer's mistake is
 - addition of one (or another different number) in float type inside some loop with the stop condition with equality to some arbitrary number. Typically, such loop will never finish.
 - If conditions with exact equality to float constant. Such constructions need not be satisfied.

Summary of Matrix Algebra I

- An $m \times n$ **matrix** A is a rectangular array of numbers with m rows and n columns. The numbers m and n are the dimensions of A .
- *Example:* 2×3 matrix A .
- The **transpose**, A^T , of a matrix A is the matrix obtained from A by writing its rows as columns. If A is an $m \times n$ matrix and $B = A^T$, then B is the $n \times m$ matrix with $b_{ij} = a_{ji}$.
- A **vector** is a matrix with the second dimension always 1.
- The **unit vector** e_i is the vector whose i -th element is 1 and all of whose other elements are 0. Usually, the size of a unit vector is clear from the context.
- A **Square** matrix is an $n \times n$ matrix.
- A **diagonal matrix** has $a_{ij} = 0$ whenever $i \neq j$.
- The $n \times n$ **identity matrix** I_n is a diagonal matrix with 1's along the diagonal.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \\ = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

$$x = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

$$\text{diag}(a_{11}, a_{22}, \dots, a_{nn}) = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}$$

$$I_n = \text{diag}(1, 1, \dots, 1) \\ = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

Summary of Matrix Algebra II

- An **upper-triangular matrix** U is one for which $u_{ij} = 0$ if $i > j$. All entries below the diagonal are zero:
- An upper-triangular matrix is **unit upper-triangular** if it has all 1's along the diagonal.

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

- A **lower-triangular matrix** L is one for which $l_{ij} = 0$ if $i < j$. All entries above the diagonal are zero:
- A lower-triangular matrix is **unit lower-triangular** if it has all 1's along the diagonal.

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}$$

- A **permutation matrix** P has exactly one 1 in each row or column, and 0's elsewhere. An example of a permutation matrix is:

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

- An **inverse matrix** for $n \times n$ matrix A is a matrix $n \times n$, we denote it as A^{-1} (if it exists), that holds:

$$A A^{-1} = I_n = A^{-1} A$$

LUP Decomposition

■ solving systems of linear equations

- Consider systems of n linear equations $Ax = b$, letting $A = (a_{ij})$, $x = (x_j)$ and $b = (b_i)$

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

...

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

- If the rank of A is less than n -then the system is *underdetermined*. An underdetermined system typically has infinitely many solutions, although it may have no solutions at all if the equations are inconsistent.
- If the number of equations exceeds the number n of unknowns, the system is *overdetermined*, and there may not exist any solutions.
- If A is nonsingular, it possesses an inverse A^{-1} and $x = A^{-1}b$ is the solution vector, because
$$x = I_n x = A^{-1} A x = A^{-1}b.$$
- Thus we have only one solution.

LUP Decomposition

■ solving systems of linear equations

□ one possible solution:

- Compute A^{-1} and then multiply both sides by A^{-1} , yielding $A^{-1}Ax = A^{-1}b$, or $x = A^{-1}b$. This approach suffers in practice from numerical instability.

□ a solution using LUP decomposition:

- The idea behind LUP decomposition is to find three $n \times n$ matrices L , U , and P such that

$$PA = LU$$

where

- L is a unit lower-triangular matrix,
- U is an upper-triangular matrix, and
- P is a permutation matrix.

LUP Decomposition

■ solving systems of linear equations with a LUP decomposition knowledge

- Multiplying both sides of $Ax = b$ by P yields the equivalent equation:

$$P Ax = Pb, \text{ which only permutes the original linear equations.}$$

- Using our LUP decomposition equality $PA=LU$, we obtain

$$LUx = Pb.$$

- We can now solve this equation by solving two triangular linear systems.

- Let us define $y = Ux$, where x is the desired solution vector.

- First, we solve the lower-triangular system:

$$Ly = Pb \quad \text{for the unknown vector } y \text{ by a method called } \mathbf{forward substitution}.$$

- Having solved for y , we then solve the upper-triangular system

$$Ux = y \quad \text{for the unknown } x \text{ by a method called } \mathbf{back substitution}.$$

- The vector x is our solution to $Ax = b$, since the permutation matrix P is invertible:

$$Ax = P^{-1}LUx = P^{-1}Pb = b.$$

LUP Decomposition

forward substitution

- can solve the lower-triangular system in $\Theta(n^2)$ given L , P , and b .
- Let us define $c = Pb$ as permutation of a vector b (in detail: $c_i = b_{\pi(i)}$).
- Since L is unit lower-triangular, equation $Ly = Pb$ can be rewritten as

$$\begin{array}{rcccccc} y_1 & & & & & = & c_1 \\ l_{21}y_1 & + & y_2 & & & = & c_2 \\ l_{31}y_1 & + & l_{32}y_2 & + & y_3 & = & c_3 \\ \vdots & & & & \ddots & & \vdots \\ l_{n1}y_1 & + & l_{n2}y_2 & + & l_{n3}y_3 & + \cdots + & y_n = c_n \end{array}$$

- We can solve for y_1 directly (from the 1st equation). Having solved for y_1 , we can substitute it into the second equation, yielding

$$y_2 = c_2 - l_{21}y_1$$

- In general, we substitute y_1, y_2, \dots, y_{i-1} "forward" into the i -th equation to solve for y_i :

$$y_i = c_i - \sum_{j=1}^{i-1} l_{ij} y_j$$

LUP Decomposition

■ back substitution

- is similar to *forward substitution*. It solves the upper-triangular system in $\Theta(n^2)$ given U and y .
- Since U is upper-triangular, we can rewrite the system $Ux = y$ as

$$\begin{array}{ccccccccc} u_{11}x_1 & + & u_{12}x_2 & + \cdots + & u_{1,n-1}x_{n-1} & + & u_{1n}x_n & = & y_1 \\ & & u_{22}x_2 & + \cdots + & u_{2,n-1}x_{n-1} & + & u_{2n}x_n & = & y_2 \\ & & & \ddots & & & & \vdots & \\ & & & & u_{n-1,n-1}x_{n-1} & + & u_{n-1,n}x_n & = & y_{n-1} \\ & & & & & & u_{nn}x_n & = & y_n \end{array}$$

- We can solve for x_n from the last equation as $x_n = y_n / u_{nn}$. Having solved for x_n , we can substitute it into the previous equation, yielding
- In general, we substitute $x_n, x_{n-1}, \dots, x_{i+1}$ "back" into the i -th equation to solve for x_i :

$$x_i = \left(y_i - \sum_{j=i+1}^n u_{ij} x_j \right) / u_{ii}$$

LUP Decomposition

■ solving systems of linear equations with a LUP decomposition knowledge

- We represent the permutation P compactly by a permutation array $\pi[1..n]$.

For $i = 1, 2, \dots, n$, the entry $\pi[i]$ indicates that $P_{i, \pi[i]} = 1$ and $P_{ij} = 0$ for $j \neq \pi[i]$.

- We have now shown that if an LUP decomposition can be computed for a nonsingular matrix A , *forward* and *back substitution* can be used to solve the system $Ax = b$ of linear equations in $\Theta(n^2)$ time.
- It remains to show how an LUP decomposition for A can be found efficiently.
- We start with the case in which A is an $n \times n$ nonsingular matrix and P is absent (or, equivalently, $P = I_n$). We call it *LU decomposition*.

LUP Decomposition

■ computing an LU decomposition

□ the idea is based on *Gaussian elimination*:

- We start by subtracting multiples of the first equation from the other equations so that the first variable is removed from those equations.
- Then, we subtract multiples of the second equation from the third and subsequent equations so that now the first and second variables are removed from them.
- We continue this process until the system that is left has an upper-triangular form-in fact, it is the matrix U . The matrix L is made up of the row multipliers that cause variables to be eliminated.

□ the recursive algorithm:

1. Divide A into following parts according the picture:

A' is $(n - 1) \times (n - 1)$ matrix,
 v is a column vector, and
 w^T is a row vector.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix},$$

LUP Decomposition

■ computing an LU decomposition

2. Then we decompose the matrix:

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & w^T \\ v & A' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix}. \end{aligned}$$

- The submatrix $A' - vw^T/a_{11}$ with dimensions $(n-1) \times (n-1)$ is called *Schur complement* A with respect to a_{11} .
- Because the *Schur complement* is nonsingular, we can now recursively find an LU decomposition of it ($= L'U'$).
- where L' is unit lower-triangular and U' is upper-triangular. Then, using matrix algebra, we have

$$\begin{aligned} A &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & A' - vw^T/a_{11} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & L'U' \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^T \\ 0 & U' \end{pmatrix} \\ &= LU, \end{aligned}$$

LUP Decomposition

■ computing an LU decomposition (nonrecursive)

- 1) Procedure LU-DECOMPOSITION(matrix A)
- 2) $n = \text{rows}[A]$;
- 3) for $k = 1$ to n do {
- 4) $u_{kk} = a_{kk}$;
- 5) for $i = k + 1$ to n do {
- 6) $l_{ik} = a_{ik}/u_{kk}$; // l_{ik} represents v_i
- 7) $u_{ki} = a_{ki}$; // u_{ki} represents w_i^T
- 8) }
- 9) for $i = k + 1$ to n do
- 10) for $j = k + 1$ to n do
- 11) $a_{ij} = a_{ij} - l_{ik}u_{kj}$;
- 12) }
- 13) return L and U

- The asymptotic time complexity is $\Theta(n^3)$.

LUP Decomposition

■ computing an LU decomposition (Example)

$$\begin{array}{cccc} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{array}$$

(a)

$$\begin{array}{c|cccc} \mathbf{2} & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ 1 & 16 & 9 & 18 \\ 2 & 4 & 9 & 21 \end{array}$$

(b)

$$\begin{array}{c|cccc} 2 & 3 & 1 & 5 \\ \hline 3 & \mathbf{4} & 2 & 4 \\ 1 & 4 & 1 & 2 \\ 2 & 1 & 7 & 17 \end{array}$$

(c)

$$\begin{array}{c|cccc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ 1 & 4 & \mathbf{1} & 2 \\ 2 & 1 & 7 & 3 \end{array}$$

(d)

$$\begin{pmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 2 & 1 & 7 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

A

L

U

(e)

- Of course, if it holds $a'_{11} = 0$ for a currently processed sub-matrix A' , then this method doesn't work, because it attempts to divide by 0.
- Thus, if $|a'_{11}|$ is near to 0, then this algorithm can produce big errors.

LUP Decomposition

■ computing an LUP decomposition (nonrecursive)

```
1) Procedure LUP-DECOMPOSITION(matrix  $A$ )
2)  $n = \text{rows}[A]$ ;
3) for  $i = 1$  to  $n$  do  $\pi[i] = i$ ;
4) for  $k = 1$  to  $n$  do { // main cycle
5)      $p = 0$ ; // initialization of pivot
6)     for  $i = k$  to  $n$  do { // selection of pivot
7)         if  $|a_{ik}| > p$  then {
8)              $p = |a_{ik}|$ ;
9)              $k' = i$ ; // position of pivot
10)        }
11)    if  $p = 0$  then error "singular matrix";
12)    exchange  $\pi[k] \leftrightarrow \pi[k']$ ;
13)    for  $i = 1$  to  $n$  do exchange  $a_{ki} \leftrightarrow a_{k'i}$ ;
14)    for  $i = k + 1$  to  $n$  do {
15)         $a_{ik} = a_{ik}/a_{kk}$ ; //  $k$ -th column of  $L$ 
16)        for  $j = k + 1$  to  $n$  do  $a_{ij} = a_{ij} - a_{ik}a_{kj}$ ; //  $U$ 
17)    }
18) }
```

- The asymptotic time complexity is $\Theta(n^3)$.
- The resulting matrices L and U are contained in “improved” matrix A in the following way

$$a_{ij} = \begin{cases} l_{ij} & \text{if } i > j \\ u_{ij} & \text{if } i \leq j \end{cases}$$

LUP Decomposition

■ computing an LUP decomposition (Example)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 1 & 2 & 0 & 2 & 0.6 \\ 2 & 3 & 3 & 4 & -2 \\ 3 & \mathbf{5} & 5 & 4 & 2 \\ 4 & -1 & -2 & 3.4 & -1 \end{array} \right] \end{array}$$

(a)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & \mathbf{5} & 5 & 4 & 2 \\ 2 & 3 & 3 & 4 & -2 \\ 1 & 2 & 0 & 2 & 0.6 \\ 4 & -1 & -2 & 3.4 & -1 \end{array} \right] \end{array}$$

(b)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & \mathbf{5} & 5 & 4 & 2 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \\ 1 & 0.4 & -2 & 0.4 & -2 \\ 4 & -0.2 & -1 & 4.2 & -0.6 \end{array} \right] \end{array}$$

(c)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \\ 1 & 0.4 & \mathbf{-2} & 0.4 & -2 \\ 4 & -0.2 & -1 & 4.2 & -0.6 \end{array} \right] \end{array}$$

(d)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 1 & 0.4 & \mathbf{-2} & 0.4 & -2 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \\ 4 & -0.2 & -1 & 4.2 & -0.6 \end{array} \right] \end{array}$$

(e)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 1 & 0.4 & \mathbf{-2} & 0.4 & -2 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \\ 4 & -0.2 & 0.5 & 4 & -0.5 \end{array} \right] \end{array}$$

(f)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 1 & 0.4 & -2 & 0.4 & -2 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \\ 4 & -0.2 & 0.5 & \mathbf{4} & -0.5 \end{array} \right] \end{array}$$

(g)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 1 & 0.4 & -2 & 0.4 & -2 \\ 4 & -0.2 & 0.5 & \mathbf{4} & -0.5 \\ 2 & 0.6 & 0 & 1.6 & -3.2 \end{array} \right] \end{array}$$

(h)

$$\begin{array}{c} \left[\begin{array}{cccc|c} 3 & 5 & 5 & 4 & 2 \\ 1 & 0.4 & -2 & 0.4 & -2 \\ 4 & -0.2 & 0.5 & \mathbf{4} & -0.5 \\ 2 & 0.6 & 0 & 0.4 & -3 \end{array} \right] \end{array}$$

(i)

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 & 2 & 0.6 \\ 3 & 3 & 4 & -2 \\ 5 & 5 & 4 & 2 \\ -1 & -2 & 3.4 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4 & 1 & 0 & 0 \\ -0.2 & 0.5 & 1 & 0 \\ 0.6 & 0 & 0.4 & 1 \end{pmatrix} \begin{pmatrix} 5 & 5 & 4 & 2 \\ 0 & -2 & 0.4 & -0.2 \\ 0 & 0 & 4 & -0.5 \\ 0 & 0 & 0 & -3 \end{pmatrix}$$

(j)

Computing Inverse Matrix

- **computing inverse matrix using LUP decomposition**
 - Using LUP-DECOMPOSITION, we can solve an equation of the form $Ax = b$ in time $\Theta(n^2)$.
 - Since the LUP-DECOMPOSITION decomposition depends on A but not b , we can run LUP-DECOMPOSITION on a second set of equations of the form $Ax = b'$ in additional time $\Theta(n^2)$.
 - Using the same LUP-DECOMPOSITION, we can solve n equations of the form $Ax = e_i$ for i from 1 to n (dimensions of matrix A is $n \times n$) where e_i is a unit vector also in time $\Theta(n^2)$.
 - If we join all n vectors e_i for i from 1 to n together then we have I_n (unit matrix).
 - The task of finding an inverse matrix X for A is to find a solution of the following matrix equation $AX = I$.
 - If we join all n solutions x from $Ax = e_i$ from 1 to n together then we have a matrix to X (so it holds: $AX = I$).
 - Since the LUP decomposition of A can be computed in time $\Theta(n^3)$, the inverse A^{-1} of a matrix A can be determined in time $\Theta(n^3)$.

References

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms (2nd ed.)*. MIT Press and McGraw-Hill. ISBN 0-262-53196-8.
- <http://babbage.cs.qc.cuny.edu/IEEE-754/References.xhtml>