

# Numerical Linear Algebra

Victor Eijkhout

Fall 2019

# Justification

Many algorithms are based in linear algebra, including some non-obvious ones such as graph algorithms. This session will mostly discuss aspects of solving linear systems, focusing on those that have computational ramifications.

# Linear algebra

- Mathematical aspects: mostly linear system solving
- Practical aspects: even simple operations are hard
  - Dense matrix-vector product: scalability aspects
  - Sparse matrix-vector: implementation

Let's start with the math. . .

# Two approaches to linear system solving

Solve  $Ax = b$

Direct methods:

- Deterministic
- Exact up to machine precision
- Expensive (in time and space)

Iterative methods:

- Only approximate
- Cheaper in space and (possibly) time
- Convergence not guaranteed

# Really bad example of direct method

Cramer's rule

write  $|A|$  for determinant, then

$$x_i = \frac{\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1i-1} & b_1 & a_{1i+1} & \dots & a_{1n} \\ a_{21} & & & & b_2 & & & a_{2n} \\ \vdots & & & & \vdots & & & \vdots \\ a_{n1} & & & & b_n & & & a_{nn} \end{vmatrix}}{|A|}$$

Time complexity  $O(n!)$

# **A close look linear system solving: direct methods**

# Gaussian elimination

Example

$$\begin{pmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 3 \end{pmatrix} x = \begin{pmatrix} 16 \\ 26 \\ -19 \end{pmatrix}$$

$$\left[ \begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 12 & -8 & 6 & 26 \\ 3 & -13 & 3 & -19 \end{array} \right] \longrightarrow \left[ \begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 0 & -4 & 2 & -6 \\ 0 & -12 & 2 & -27 \end{array} \right] \longrightarrow \left[ \begin{array}{ccc|c} 6 & -2 & 2 & 16 \\ 0 & -4 & 2 & -6 \\ 0 & 0 & -4 & -9 \end{array} \right]$$

Solve  $x_3$ , then  $x_2$ , then  $x_1$

6, -4, -4 are the 'pivots'

# Pivoting

If a pivot is zero, exchange that row and another.

(there is always a row with a nonzero pivot if the matrix is nonsingular)

best choice is the largest possible pivot

in fact, that's a good choice even if the pivot is not zero:

**partial pivoting**

(full pivoting would be row *and* column exchanges)



# Roundoff control

Consider

$$\begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 1 + \varepsilon \\ 2 \end{pmatrix}$$

with solution  $x = (1, 1)^t$

Ordinary elimination:

$$\begin{pmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{pmatrix} x = \begin{pmatrix} 1 + \varepsilon \\ 2 - \frac{1 + \varepsilon}{\varepsilon} \end{pmatrix} = \begin{pmatrix} 1 + \varepsilon \\ 1 - \frac{1}{\varepsilon} \end{pmatrix}.$$

We can now solve  $x_2$  and from it  $x_1$ :

$$\begin{cases} x_2 &= (1 - \varepsilon^{-1}) / (1 - \varepsilon^{-1}) = 1 \\ x_1 &= \varepsilon^{-1} (1 + \varepsilon - x_2) = 1. \end{cases}$$

## Roundoff 2

If  $\epsilon < \epsilon_{\text{mach}}$ , then in the rhs  $1 + \epsilon \rightarrow 1$ , so the system is:

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

The solution  $(1, 1)$  is still correct!

Eliminating:

$$\begin{pmatrix} \epsilon & 1 \\ 0 & 1 - \epsilon^{-1} \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 - \epsilon^{-1} \end{pmatrix} \Rightarrow \begin{pmatrix} \epsilon & 1 \\ 0 & -\epsilon^{-1} \end{pmatrix} x = \begin{pmatrix} 1 \\ -\epsilon^{-1} \end{pmatrix}$$

Solving first  $x_2$ , then  $x_1$ , we get:

$$\begin{cases} x_2 &= \epsilon^{-1} / \epsilon^{-1} = 1 \\ x_1 &= \epsilon^{-1} (1 - 1 \cdot x_2) = \epsilon^{-1} \cdot 0 = 0, \end{cases}$$

so  $x_2$  is correct, but  $x_1$  is completely wrong.

## Roundoff 3

Pivot first:

$$\begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix} x = \begin{pmatrix} 2 \\ 1 + \varepsilon \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 1 \\ 0 & 1 - \varepsilon \end{pmatrix} x = \begin{pmatrix} 2 \\ 1 - \varepsilon \end{pmatrix}$$

Now we get, regardless the size of epsilon:

$$x_2 = \frac{1 - \varepsilon}{1 - \varepsilon} = 1, \quad x_1 = 2 - x_2 = 1$$

# LU factorization

Same example again:

$$A = \begin{pmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 3 \end{pmatrix}$$

2nd row minus  $2 \times$  first; 3rd row minus  $1/2 \times$  first;  
equivalent to

$$L_1 A x = L_1 b, \quad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}$$

(elementary reflector)

## LU 2

Next step:  $L_2 L_1 A x = L_2 L_1 b$  with

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix}$$

Define  $U = L_2 L_1 A$ , then  $A = LU$  with  $L = L_1^{-1} L_2^{-1}$   
'LU factorization'

## LU 3

Observe:

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \quad L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1/2 & 0 & 1 \end{pmatrix}$$

Likewise

$$L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \quad L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{pmatrix}$$

Even more remarkable:

$$L_1^{-1}L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1/2 & 3 & 1 \end{pmatrix}$$

Can be computed in place! (pivoting?)

# Solve LU system

$Ax = b \longrightarrow LUX = b$  solve in two steps:

$Ly = b$ , and  $Ux = y$

Forward sweep:

$$\begin{pmatrix} 1 & & & & 0 \\ \ell_{21} & 1 & & & \\ \ell_{31} & \ell_{32} & 1 & & \\ \vdots & & \ddots & & \\ \ell_{n1} & \ell_{n2} & & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

# Solve LU system

$Ax = b \longrightarrow LUX = b$  solve in two steps:

$Ly = b$ , and  $Ux = y$

Forward sweep:

$$\begin{pmatrix} 1 & & & 0 \\ \ell_{21} & 1 & & \\ \ell_{31} & \ell_{32} & 1 & \\ \vdots & & \ddots & \\ \ell_{n1} & \ell_{n2} & & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$y_1 = b_1, \quad y_2 = b_2 - \ell_{21}y_1, \dots$$



## Solve LU 2

Backward sweep:

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

## Solve LU 2

Backward sweep:

$$\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ 0 & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$x_n = u_{nn}^{-1} y_n, \quad x_{n-1} = u_{n-1,n-1}^{-1} (y_{n-1} - u_{n-1,n} x_n), \dots$$

# Computational aspects

Compare:

Matrix-vector product:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \leftarrow \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Solving LU system:

$$\begin{pmatrix} a_{11} & & 0 \\ \vdots & \ddots & \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

(and similarly the  $U$  matrix)

Compare operation counts. Can you think of other points of comparison? (Think modern computers.)

# Short detour: Partial Differential Equations

## Second order PDEs; 1D case

$$\begin{cases} -u''(x) = f(x) & x \in [a, b] \\ u(a) = u_a, u(b) = u_b \end{cases}$$

## Second order PDEs; 1D case

$$\begin{cases} -u''(x) = f(x) & x \in [a, b] \\ u(a) = u_a, u(b) = u_b \end{cases}$$

Using Taylor series:

$$u(x+h) + u(x-h) = 2u(x) + u''(x)h^2 + u^{(4)}(x)\frac{h^4}{12} + \dots$$

so

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + O(h^2)$$

Numerical scheme:

$$-\frac{u(x+h) - 2u(x) + u(x-h)}{h^2} = f(x, u(x), u'(x))$$

## This leads to linear algebra

$$-u_{xx} = f \rightarrow \frac{2u(x) - u(x+h) - u(x-h)}{h^2} = f(x, u(x), u'(x))$$

Equally spaced points on  $[0, 1]$ :  $x_k = kh$  where  $h = 1/(n+1)$ , then

$$-u_{k+1} + 2u_k - u_{k-1} = -h^2 f(x_k, u_k, u'_k) \quad \text{for } k = 1, \dots, n$$

Written as matrix equation:

$$\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \ddots & \ddots & \ddots \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_1 + u_0 \\ f_2 \\ \vdots \end{pmatrix}$$

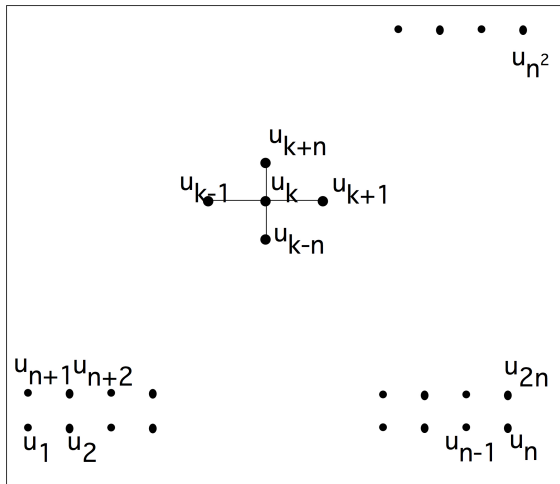
## Second order PDEs; 2D case

$$\begin{cases} -u_{xx}(\bar{x}) - u_{yy}(\bar{x}) = f(\bar{x}) & x \in \Omega = [0, 1]^2 \\ u(\bar{x}) = u_0 & \bar{x} \in \delta\Omega \end{cases}$$

Now using central differences in both  $x$  and  $y$  directions.



# The stencil view of things



# Sparse matrix from 2D equation

$$\left( \begin{array}{cccc|cccc|c} 4 & -1 & & & 0 & -1 & & & 0 \\ -1 & 4 & 1 & & & & & & \\ & \ddots & \ddots & \ddots & & & & & \\ & & \ddots & \ddots & -1 & & & & \\ 0 & & & -1 & 4 & 0 & & & -1 \\ \hline -1 & & & & 0 & 4 & -1 & & -1 \\ & -1 & & & & -1 & 4 & -1 & \\ & \uparrow & \ddots & & & \uparrow & \uparrow & \uparrow & \\ & k-n & & & & k-1 & k & k+1 & \\ & & & & -1 & & & -1 & 4 \\ \hline & & & & & \ddots & & & \ddots \end{array} \right)$$

The stencil view is often more insightful.

# Matrix properties

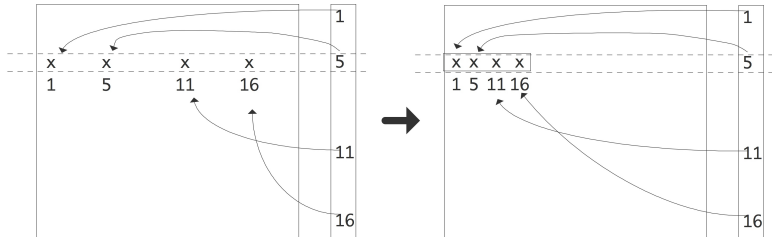
- Very sparse, banded
- Factorization takes less than  $n^2$  space,  $n^3$  work
- Symmetric (only because 2nd order problem)
- Sign pattern: positive diagonal, nonpositive off-diagonal (true for many second order methods)
- Positive definite (just like the continuous problem)
- Constant diagonals: only because of the constant coefficient differential equation
- Factorization: lower complexity than dense, recursion length less than  $N$ .

# Sparse matrices

# Sparse matrix storage

Matrix above has many zeros:  $n^2$  elements but only  $O(n)$  nonzeros.  
Big waste of space to store this as square array.

Matrix is called 'sparse' if there are enough zeros to make specialized storage feasible.



# Compressed Row Storage

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}. \quad (1)$$

Compressed Row Storage (CRS): store all nonzeros by row, their column indices, pointers to where the columns start (1-based indexing):

val	10	-2	3	9	3	7	8	7	3 ... 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ... 5	6	2	5	6
row_ptr	1	3	6	9	13	17	20	.					

# Sparse matrix operations

Most common operation: matrix-vector product

```
for (row=0; row<nrows; row++) {  
    s = 0;  
    for (icol=ptr[row]; icol<ptr[row+1]; icol++) {  
        int col = ind[icol];  
        s += a[aptr] * x[col];  
        aptr++;  
    }  
    y[row] = s;  
}
```

Operations with changes to the nonzero structure are much harder!

Indirect addressing of  $x$  gives low spatial and temporal locality.

## Exercise: sparse coding

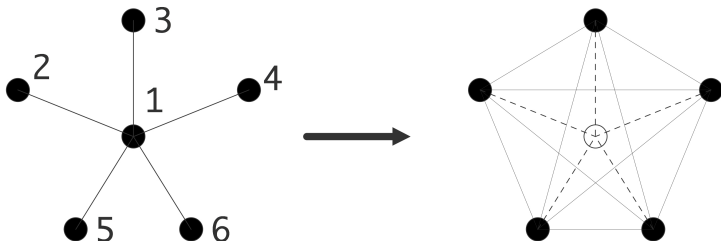
What if you need access to both rows and columns at the same time? Implement an algorithm that tests whether a matrix stored in CRS format is symmetric. Hint: keep an array of pointers, one for each row, that keeps track of how far you have progressed in that row.



# Fill-in

Fill-in: index  $(i, j)$  where  $a_{ij} = 0$  but  $\ell_{ij} \neq 0$  or  $u_{ij} \neq 0$ .

## Fill-in is a function of ordering



$$\begin{pmatrix} * & * & \cdots & * \\ * & * & & \emptyset \\ \vdots & & \ddots & \\ * & \emptyset & & * \end{pmatrix}$$

After factorization the matrix is dense.  
Can this be permuted?

## LU of a sparse matrix

$$\Rightarrow \left( \begin{array}{cccc|cc} 4 & -1 & 0 & \dots & -1 & \\ -1 & 4 & -1 & 0 & \dots & 0 & -1 \\ & \ddots & \ddots & \ddots & & \ddots & \\ -1 & 0 & \dots & & 4 & -1 & \\ 0 & -1 & 0 & \dots & -1 & 4 & -1 \end{array} \right)$$

$$\Rightarrow \left( \begin{array}{c|cccc|cc} 4 & -1 & 0 & \dots & -1 & \\ \hline & 4 - \frac{1}{4} & -1 & 0 & \dots & -1/4 & -1 \\ & \ddots & \ddots & \ddots & & \ddots & \\ -1/4 & \dots & & & 4 - \frac{1}{4} & -1 & \\ \hline & -1 & 0 & \dots & -1 & 4 & -1 \end{array} \right)$$

## Exercise: LU of a band matrix

Suppose a matrix  $A$  is banded with *halfbandwidth*  $p$ :

$$a_{ij} = 0 \quad \text{if } |i - j| > p$$

Derive how much space an LU factorization of  $A$  will take if no pivoting is used. (For bonus points: consider partial pivoting.)

Can you also derive how much space the inverse will take? (Hint: if  $A = LU$ , does that give you an easy formula for the inverse?)