Numerical Linear Algebra

Victor Eijkhout

2018

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

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

Time complexity $O(n!)$

Gaussian elimination

Example

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

$$\begin{bmatrix} 6 & -2 & 2 & | & 16 \\ 12 & -8 & 6 & | & 26 \\ 3 & -13 & 3 & | & -19 \end{bmatrix} \longrightarrow \begin{bmatrix} 6 & -2 & 2 & | & 16 \\ 0 & -4 & 2 & | & -6 \\ 0 & -12 & 2 & | & -27 \end{bmatrix} \longrightarrow \begin{bmatrix} 6 & -2 & 2 & | & 16 \\ 0 & -4 & 2 & | & -6 \\ 0 & 0 & -4 & | & -9 \end{bmatrix}$$

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

Roundoff control

Consider

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

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

Ordinary elimination:

$$\begin{pmatrix} \epsilon & 1 \\ 0 & (1-\frac{1}{\epsilon}) \end{pmatrix} x = \begin{pmatrix} 1 \\ 2-\frac{1}{\epsilon} \end{pmatrix}$$

$$\Rightarrow x_2 = \frac{2-\frac{1}{\epsilon}}{1-\frac{1}{\epsilon}} \Rightarrow x_1 = \frac{1-x_2}{\epsilon}$$

Roundoff 2

If $\varepsilon < \varepsilon_{\mathrm{mach}}$, then $2 - 1/\varepsilon = -1/\varepsilon$ and $1 - 1/\varepsilon = -1/\varepsilon$, so

$$x_2 = \frac{2 - \frac{1}{\varepsilon}}{1 - \frac{1}{\varepsilon}} = 1, \Rightarrow x_1 = \frac{1 - x_2}{\varepsilon} = 0$$

Roundoff 3

Pivot first:

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

If ε very small:

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

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

$$L_1 Ax = L_1 b, \qquad L_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}$$

(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 $A = L_1^{-1} L_2^{-1} U$
'LU factorization'

LU 3

Observe:

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

Likewise

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

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:

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

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

Solve LU 2

Backward sweep:

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

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

Computational aspects

- Factoring and solving are recursive: parallellism is not trivial
- (compare matrix-matrix and matrix-vector product)
- Complexity: $O(n^3)$ operations for factorization,
  $O(n^2)$ for solution
- Much more stable than inversion, not quite as stable as QR

Matrix from 1D PDE

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

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

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

Written as matrix equation:

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

Sparse matrix from 2D equation

$$
\left(\begin{array}{ccccc|ccccc|cc}
4 & -1 & & & \emptyset & -1 & & & & \emptyset & & \\
-1 & 4 & 1 & & & & -1 & & & & & \\
 & \ddots & \ddots & \ddots & & & & \ddots & & & & \\
 & & \ddots & \ddots & -1 & & & & \ddots & & & \\
\emptyset & & & -1 & 4 & \emptyset & & & & -1 & & \\
\hline
-1 & & & & \emptyset & 4 & -1 & & & & -1 & \\
 & -1 & & & & -1 & 4 & -1 & & & & -1 \\
 & \uparrow & \ddots & & & \uparrow & \uparrow & \uparrow & & & & \uparrow \\
 & k-n & & & & k-1 & k & k+1 & & -1 & & k+n \\
 & & & & -1 & & & & -1 & 4 & & \\
\hline
 & & & & & \ddots & & & & & \ddots &
\end{array}\right)
$$

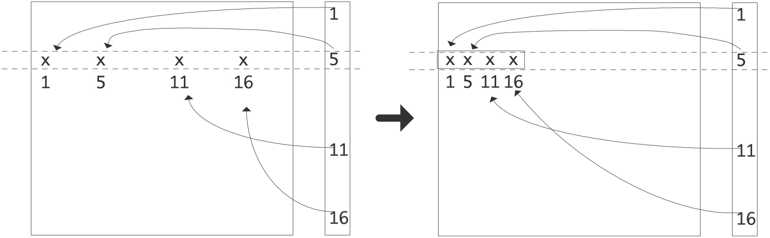
Matrix properties

- 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 matrix storage

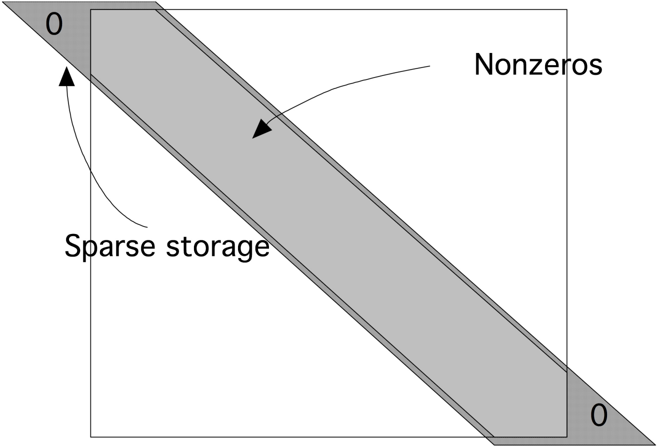
Matrix above has many zeros: n^2 elements but only $O(n)$ nonzeros. Big waste of space to store this as square array.

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



Storage by diagonals

Use the banded format:



Diagonal matrix-vector product

$$y_i \leftarrow y_i + A_{ii} x_i,$$

$$y_i \leftarrow y_i + A_{ii+1} x_{i+1} \quad \text{for } i < n,$$

$$y_i \leftarrow y_i + A_{ii-1} x_{i-1} \quad \text{for } i > 1.$$

```
for diag = -diag_left, diag_right
    for loc = max(1,1-diag), min(n,n-diag)
        y(loc) = y(loc) + val(loc,diag) * x(loc+diag)
    end
end
```

Pro/con

Pro: long vectors (D’Azevedo: $20\times$ speedup on Cray X-1)

Con: limited, little cache reuse

Variants: jagged diagonal

Compressed Row Storage

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

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	$3 \cdots 9$	13	4	2	-1
`col_ind`	1	5	1	2	6	2	3	4	$1 \cdots 5$	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

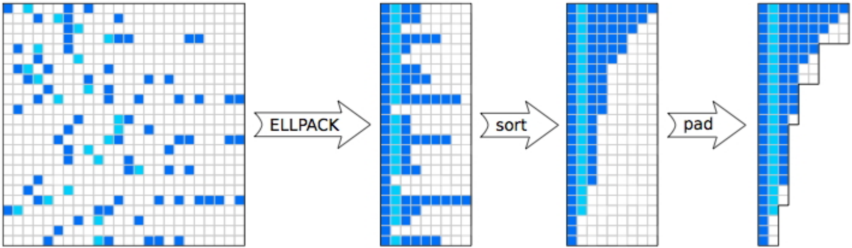
What if you need access to both rows and columns at the same time? Implement an algorithm that tests whether a matrix stored in CRS format is symmetric. Hint: keep an array of pointers, one for each row, that keeps track of how far you have progressed in that row.

Jagged diagonal storage

Align irregular sparse matrices along 'jagged' diagonals

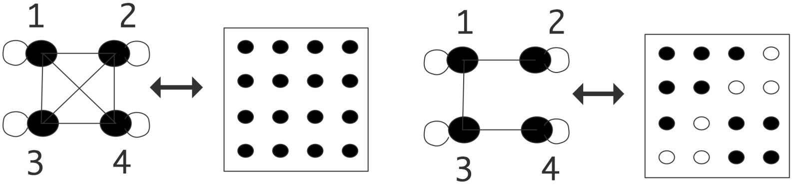
$$\begin{pmatrix} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{pmatrix} \rightarrow \begin{pmatrix} 10 & -3 & 1 & \\ 9 & 6 & -2 & \\ 3 & 8 & 7 & \\ 6 & 7 & 5 & 4 \\ 9 & 13 & & \\ 5 & -1 & & \end{pmatrix}$$

Long vectors make it suitable for GPU

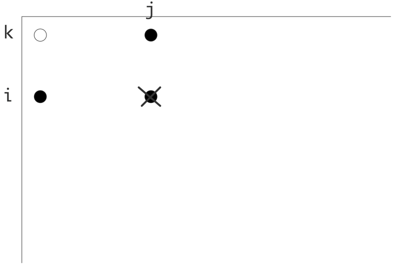


Graph theory of sparse matrices

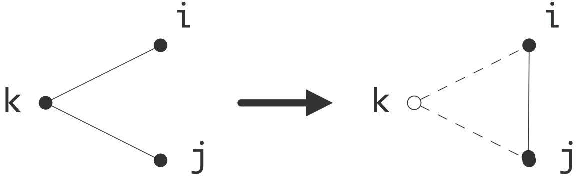
Some things (reducibility) are easiest seen in a graph



Graph theory of sparse elimination



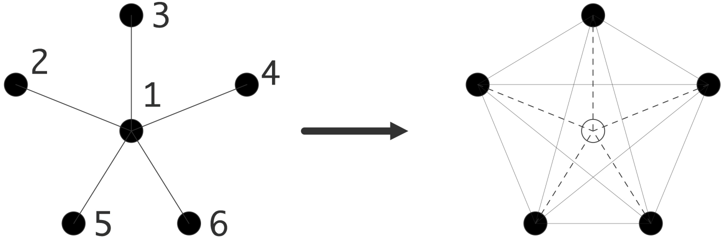
$$a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}$$



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

$$\left(\begin{array}{cccccc:ccc}
4 & -1 & 0 & \ldots & & & -1 & & \\
-1 & 4 & -1 & 0 & \ldots & & 0 & -1 & \\
 & \ddots & \ddots & \ddots & & & & \ddots & \\
\hdashline
-1 & 0 & \ldots & & & & 4 & -1 & \\
0 & -1 & 0 & \ldots & & & -1 & 4 & -1
\end{array}\right)$$

$$\Rightarrow \quad \left(\begin{array}{c|ccccc:ccc}
4 & -1 & 0 & \ldots & & & -1 & & \\
\hline
 & 4-\frac{1}{4} & -1 & 0 & \ldots & & -1/4 & -1 & \\
 & \ddots & \ddots & \ddots & & & & \ddots & \\
\hdashline
 & -1/4 & \ldots & & & & 4-\frac{1}{4} & -1 & \\
 & -1 & 0 & \ldots & & & -1 & 4 & -1
\end{array}\right)$$

Exercise: LU of a band matrix

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

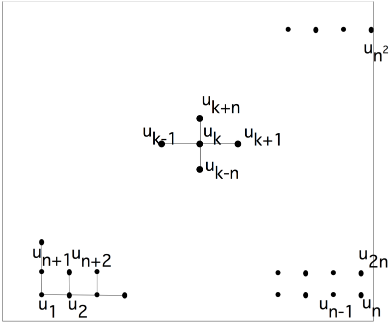
$$a_{ij} = 0 \quad \text{if } |i-j| > p$$

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

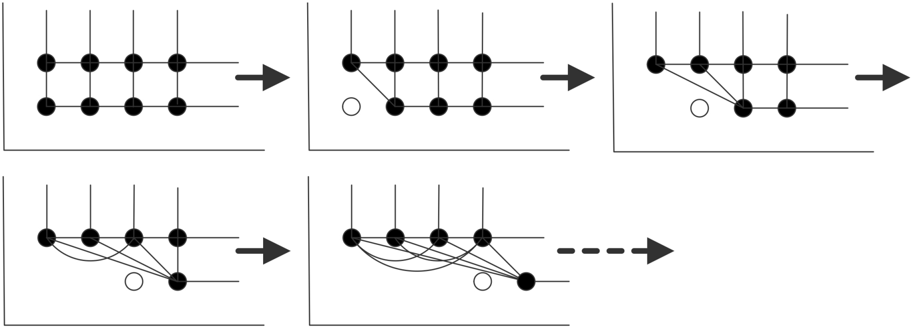
Can you also derive how much space the inverse will take? (Hint: if $A = LU$, does that give you an easy formula for the inverse?)

Domain view

Graph of connectivity of variables:



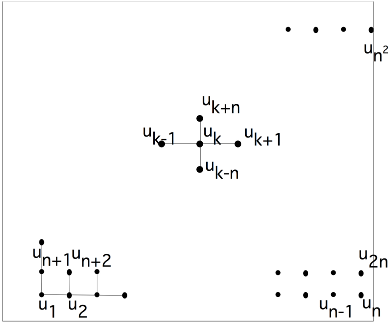
now start eliminating in sequence



Remaining matrix has a dense leading block

Fill-in during LU

Recall:



Fill-in during LU

2D BVP: Ω is $n \times n$, gives matrix of size $N = n^2$, with bandwidth n.

Matrix storage $O(N)$

LU storage $O(N^{3/2})$ (limited to band)

LU factorization work $O(N^2)$

Cute fact: storage can be computed linear in #nonzeros