Numerical Linear Algebra

Victor Eijkhout and Lars Koesterke

Spring 2021
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

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

Time complexity $O(n!)$
Not a good method either

$$Ax = b$$

• Compute explicitly $A^{-1}$,
• then $x \leftarrow A^{-1}b$.
• Numerical stability issues.
• Amount of work?
A close look linear system solving: direct methods
Gaussian elimination

Example

\[
\begin{pmatrix}
6 & -2 & 2 \\
12 & -8 & 6 \\
3 & -13 & 3
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}
=
\begin{pmatrix}
16 \\
26 \\
-19
\end{pmatrix}
\]

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

Solve \(x_3\), then \(x_2\), then \(x_1\)

\(6, -4, -4\) are the ‘pivots’
Gaussian elimination, step by step

\[LU\text{ factorization}\]:

for \( k = 1, n - 1 \):

\{eliminate values in column \( k \}\}

\{eliminate values in column \( k \}\}:

for \( i = k + 1 \) to \( n \):

\{compute multiplier for row \( i \}\}

\{update row \( i \}\}

\{compute multiplier for row \( i \}\}

\[
\begin{align*}
a_{ik} &\leftarrow a_{ik} / a_{kk} \\
\end{align*}
\]

\{update row \( i \}\}:

for \( j = k + 1 \) to \( n \):

\[
\begin{align*}
a_{ij} &\leftarrow a_{ij} - a_{ik} \ast a_{kj} \\
\end{align*}
\]
Gaussian elimination, all together

\langle LU \text{ factorization} \rangle:
\begin{align*}
\text{for } k = 1, n - 1: \\
\text{for } i = k + 1 \text{ to } n: \\
\quad a_{ik} &\leftarrow a_{ik} / a_{kk} \\
\text{for } j = k + 1 \text{ to } n: \\
\quad a_{ij} &\leftarrow a_{ij} - a_{ik} * a_{kj}
\end{align*}

Amount of work:
\[
\sum_{k=1}^{n-1} \sum_{i,j>k} 1 = \sum_{k=1}^{n-1} (n - k)^2 \approx \sum_{k} k^2 \approx n^3 / 3
\]
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

\[
\begin{pmatrix}
\varepsilon & 1 \\
1 & 1
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} =
\begin{pmatrix}
1 + \varepsilon \\
2
\end{pmatrix}
\]

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

Ordinary elimination:

\[
\begin{pmatrix}
\varepsilon & 1 \\
0 & 1 - \frac{1}{\varepsilon}
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} =
\begin{pmatrix}
1 + \varepsilon \\
2 - \frac{1 + \varepsilon}{\varepsilon}
\end{pmatrix} =
\begin{pmatrix}
1 + \varepsilon \\
1 - \frac{1}{\varepsilon}
\end{pmatrix}.
\]

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 $\varepsilon < \varepsilon_{\text{mach}}$, then in the rhs $1 + \varepsilon \rightarrow 1$, so the system is:

\[
\begin{pmatrix}
\varepsilon & 1 \\
1 & 1
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
=
\begin{pmatrix}
1 \\
2
\end{pmatrix}
\]

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

Eliminating:

\[
\begin{pmatrix}
\varepsilon & 1 \\
0 & 1 - \varepsilon^{-1}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
=
\begin{pmatrix}
1 \\
2 - \varepsilon^{-1}
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
\varepsilon & 1 \\
0 & -\varepsilon^{-1}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
=
\begin{pmatrix}
1 \\
-\varepsilon^{-1}
\end{pmatrix}
\]

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

\[
\begin{cases}
x_2 = \varepsilon^{-1} / \varepsilon^{-1} = 1 \\
x_1 = \varepsilon^{-1} (1 - 1 \cdot x_2) = \varepsilon^{-1} \cdot 0 = 0,
\end{cases}
\]

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

\[
L_1 Ax = L_1 b, \quad L_1 = \begin{pmatrix}
1 & 0 & 0 \\
-2 & 1 & 0 \\
-1/2 & 0 & 1
\end{pmatrix}
\]

(elementary reflector)
LU 2

Next step: $L_2L_1Ax = L_2L_1b$ with

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

Define $U = L_2L_1A$, 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:

\[ Ly = b, \text{ and } Ux = y \]

Forward sweep:

\[
\begin{pmatrix}
1 & & & & 0 \\
\ell_{21} & 1 & & & \\
\ell_{31} & \ell_{32} & 1 & & \\
& \ddots & \ddots & \ddots & \\
\ell_{n1} & \ell_{n2} & \cdots & 1 & \\
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
\vdots \\
y_n \\
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_n \\
\end{pmatrix}
\]
Solve LU system

Ax = b \rightarrow LUx = b\text{ solve in two steps:}
Ly = b, \text{ and } Ux = y

Forward sweep:

\[
\begin{pmatrix}
1 & 0 \\
\ell_{21} & 1 \\
\ell_{31} & \ell_{32} & 1 \\
\vdots & \ddots & \ddots \\
\ell_{n1} & \ell_{n2} & \cdots & 1
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
\vdots \\
y_n
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_n
\end{pmatrix}
\]

y_1 = b_1, \quad y_2 = b_2 - \ell_{21}y_1, \ldots
Solve LU 2

Backward sweep:

\[
\begin{pmatrix}
    u_{11} & u_{12} & \cdots & u_{1n} \\
    u_{22} & \cdots & u_{2n} \\
    \vdots & \ddots & \vdots \\
    0 & \cdots & u_{nn}
\end{pmatrix}
\begin{pmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{pmatrix}
=
\begin{pmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_n
\end{pmatrix}
\]
Solve LU 2

Backward sweep:

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

\[x_n = u_{nn}^{-1} y_n, \quad x_{n-1} = u_{n-1n}^{-1} (y_{n-1} - u_{n-1n} x_n), \ldots\]
Computational aspects

Compare:

Matrix-vector product:

\[
\begin{pmatrix}
y_1 \\
\vdots \\
y_n
\end{pmatrix}
\leftarrow
\begin{pmatrix}
a_{11} & \cdots & a_{1n} \\
\vdots & \ddots & \vdots \\
a_{n1} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
\vdots \\
x_n
\end{pmatrix}
\]

Solving LU system:

\[
\begin{pmatrix}
a_{11} & 0 \\
\vdots & \ddots \\
a_{n1} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
\vdots \\
x_n
\end{pmatrix}
=
\begin{pmatrix}
y_1 \\
\vdots \\
y_n
\end{pmatrix}
\]

(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:

\[
u(x + h) + u(x - h) = 2u(x) + u''(x)h^2 + u^{(4)}(x)\frac{h^4}{12} + \cdots
\]

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

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

Written as matrix equation:

\[
\begin{pmatrix}
2 & -1 & \emptyset \\
-1 & 2 & -1 \\
\emptyset & \cdots & \cdots & \cdots \\
\emptyset & \cdots & \cdots & \cdots \\
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
\vdots \\
\vdots \\
\end{pmatrix}
= 
\begin{pmatrix}
f_1 + u_0 \\
\vdots \\
\vdots \\
\end{pmatrix}
\]
Second order PDEs; 2D case

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

Now using central differences in both $x$ and $y$ directions.
The stencil view of things
Sparse matrix from 2D equation

\[
\begin{pmatrix}
4 & -1 & 0 \\
-1 & 4 & 1 \\
\vdots & \vdots & \vdots \\
0 & -1 & 4 \\
\end{pmatrix}
\begin{pmatrix}
-1 \\
-1 \\
\vdots \\
0 \\
-1 \\
\end{pmatrix}
\begin{pmatrix}
-1 & 4 \\
-1 & 4 & -1 \\
\vdots \\
-1 & 4 \\
\end{pmatrix}
\begin{pmatrix}
0 \\
-1 \\
\vdots \\
0 \\
-1 \\
\end{pmatrix}
\begin{pmatrix}
-1 & 4 \\
-1 & 4 & -1 \\
\vdots \\
-1 & 4 \\
\end{pmatrix}
\begin{pmatrix}
0 \\
-1 \\
\vdots \\
0 \\
-1 \\
\end{pmatrix}
\begin{pmatrix}
k-n \\
k-1 \\
\vdots \\
k-1 \\
\end{pmatrix}
\begin{pmatrix}
k+1 \\
k \\
\vdots \\
k \\
\end{pmatrix}
\begin{pmatrix}
-1 \\
-1 \\
\vdots \\
-1 \\
\end{pmatrix}
\begin{pmatrix}
4 \\
-1 \\
\vdots \\
4 \\
\end{pmatrix}
\end{pmatrix}

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

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

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

<table>
<thead>
<tr>
<th>val</th>
<th>10</th>
<th>-2</th>
<th>3</th>
<th>9</th>
<th>3</th>
<th>7</th>
<th>8</th>
<th>7</th>
<th>3 \cdots</th>
<th>9</th>
<th>13</th>
<th>4</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_ind</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1 \cdots</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>row_ptr</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sparse matrix-vector operations

• Simplest, and important in many contexts: matrix-vector product.
• Matrix-matrix product rare in engineering science very important in Deep Learning
• Gaussian elimination is a complicated story.
• In general: changes to sparse structure are hard!
Dense matrix-vector product

Most common operation in many cases: matrix-vector product

```c
aptr = 0;
for (row=0; row<nrows; row++) {
    s = 0;
    for (col=0; col<ncols; col++) {
        s += a[aptr] * x[col];
        aptr++;
    }
    y[row] = s;
}
```

Reuse? Locality? Cachelines?
Better implementation

Three loops: block, columns inside block, row; permute blocks to outermost

![Diagram of regular traversal and blocked traversal]
Sparse matrix-vector product

```c
aptr = 0;
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;
}
```

Again: Reuse? Locality? Cachelines?

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

Remember Gaussian elimination algorithm:

\[
\begin{align*}
\text{for } k = 1, n - 1: \\
&\quad \text{for } i = k + 1 \text{ to } n: \\
&\quad\quad \text{for } j = k + 1 \text{ to } n: \\
&\quad\quad\quad a_{ij} \leftarrow a_{ij} - a_{ik} \cdot a_{kj} / a_{kk}
\end{align*}
\]

Fill-in: index \((i, j)\) where \(a_{ij} = 0\) originally, but gets updated to non-zero. (and so \(\ell_{ij} \neq 0\) or \(u_{ij} \neq 0\).)

Change in the sparsity structure! How do you deal with that?
LU of a sparse matrix

\[
\begin{pmatrix}
2 & -1 & 0 & \ldots \\
-1 & 2 & -1 \\
0 & -1 & 2 & -1 \\
\ldots & \ldots & \ldots & \ldots
\end{pmatrix}
\Rightarrow
\begin{pmatrix}
2 & -1 & 0 & \ldots \\
0 & 2 - \frac{1}{2} & -1 \\
0 & -1 & 2 & -1 \\
\ldots & \ldots & \ldots & \ldots
\end{pmatrix}
\]

How does this continue by induction?

Observations?
LU of a sparse matrix

\[
\begin{pmatrix}
4 & -1 & 0 & \ldots & -1 \\
-1 & 4 & -1 & 0 & \ldots \\
& & & & \\
& & & & \\
-1 & 0 & \ldots & & \\
0 & -1 & 0 & \ldots & -1 \\
\end{pmatrix}
\quad \Rightarrow
\begin{pmatrix}
4 & -1 & 0 & \ldots & -1 \\
4 - \frac{1}{4} & -1 & 0 & \ldots & -1/4 \\
& & & & \\
& & & & \\
-1/4 & \ldots & & & 4 - \frac{1}{4} \\
-1 & 0 & \ldots & & -1 \\
\end{pmatrix}
\]
A little graph theory

Graph is a tuple $G = \langle V, E \rangle$ where $V = \{v_1, \ldots, v_n\}$ for some $n$, and $E \subset \{(i, j): 1 \leq i, j \leq n, i \neq j\}$.

\[
\begin{align*}
V &= \{1, 2, 3, 4, 5, 6\} \\
E &= \{(1, 2), (2, 6), (4, 3), (4, 4), (4, 5)\}
\end{align*}
\]
Graphs and matrices

For a graph $G = \langle V, E \rangle$, the adjacency matrix $M$ is defined by

$$M_{ij} = \begin{cases} 
1 & (i, j) \in E \\
0 & \text{otherwise}
\end{cases}$$

A dense and a sparse matrix, both with their adjacency graph
Fill-in

Fill-in: index \((i, j)\) where \(a_{ij} = 0\) originally, but gets updated to non-zero.

\[
a_{ij} \leftarrow a_{ij} - a_{ik} * a_{kj} / a_{kk}
\]

Eliminating a vertex introduces a new edge in the quotient graph.
LU of sparse matrix, with graph view: 1

Original matrix.
LU of sparse matrix, with graph view: 2

Eliminating (2, 1) causes fill-in at (2, 3).
LU of sparse matrix, with graph view: 3

Remaining matrix when step 1 finished.
LU of sparse matrix, with graph view: 4

Eliminating (3, 2) fills (3, 4)
LU of sparse matrix, with graph view: 5

After step 2
Fill-in is a function of ordering

After factorization the matrix is dense.
Can this be permuted?
Exercise: LU of a penta-diagonal matrix

Consider the matrix

\[
\begin{pmatrix}
2 & 0 & -1 \\
0 & 2 & 0 & -1 \\
-1 & 0 & 2 & 0 & -1 \\
-1 & 0 & 2 & 0 & -1 \\
\ddots & \ddots & \ddots & \ddots & \ddots \\
\end{pmatrix}
\]

Describe the LU factorization of this matrix:

- Convince yourself that there will be no fill-in. Give an inductive proof of this.
- What does the graph of this matrix look like? (Find a tutorial on graph theory. What is a name for such a graph?)
- Can you relate this graph to the answer on the question of the fill-in?
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?)