NUR lecture 3

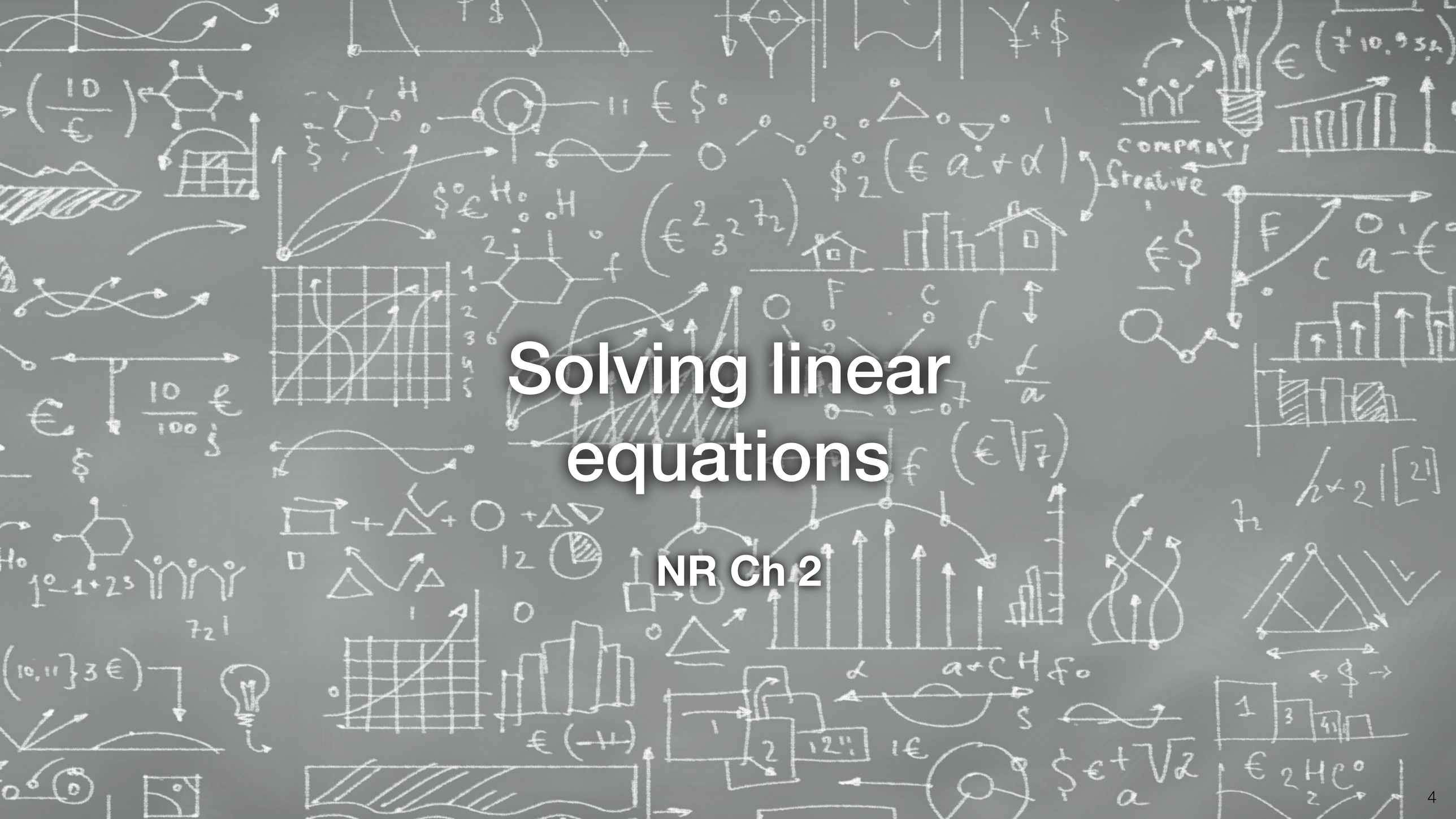
In this lecture:

Matrix inverse
Gauss-Jordan elimination
Gaussian elimination with back-substitution
LU decomposition & Crout's algorithm
Solving systems of equations
Iterative improvements
Least squares in matrix form
Singular value decomposition

Numerical Recipes for Astrophysics A

Lecture 3

Questions about last week?



Solving linear equations

NR Ch 2

Solving linear equations

- Equation to matrix form

N variables x_j, M constraining b_i

$$\rightarrow a_{i0}x_0 + a_{i1}x_1 + \dots + a_{i(N-1)}x_{N-1} = b_i \quad , \quad i = 0, 1, 2, \dots, M-1$$

$$A\underline{x} = \underline{b} : \begin{bmatrix} a_{00} & a_{01} & \cdots \\ a_{10} & a_{11} & \cdots \\ \vdots & & \ddots \\ a_{(M-1)0} & \cdots & a_{(M-1)(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_0 \\ \vdots \\ b_{M-1} \end{bmatrix}$$

If N out of M eqs. are indep. $\rightarrow$ formal solution $\rightarrow$ $\underline{x} = A^{-1}\underline{b}$

# of indep. eqs $< N$: no solution / not unique

# of indep. eqs $> N$: no solution

$\rightarrow$ A is singular $\Leftrightarrow$ A^{-1} does not exist

- Libraries: BLAS, LAPACK

Basic methods

- Compute and use the inverse?

Avoid (expensive, a lot of operations → large error)

- Gauss-Jordan elimination $(A|b|I) \rightarrow (I|x|A^{-1})$ (identity matrix)

Row operations: (1) addition/subtraction of rows; (2) scale rows; (3) row swaps

$\times \frac{1}{2}$

$$\left[\begin{array}{ccc|c|ccc} 0 & 2 & 6 & 2 & 1 & 0 & 0 \\ 3 & 3 & 4 & 7 & 0 & 1 & 0 \\ 1 & 3 & 5 & 6 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c|ccc} 0 & 1 & 3 & 1 & 1/2 & 0 & 0 \\ 0 & 0 & -11 & -11 & 0 & 1 & -3 \\ 1 & 3 & 5 & 6 & 0 & 0 & 1 \end{array}\right] \rightarrow \cdots$$

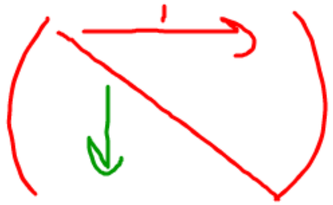
pivoting: getting particular elements on the diagonal

(first non-zero on a row for example)

Gauss-Jordan algorithm

1. Take two matrix objects (define your own class: 2D array with a certain number of rows and columns) as input, one for $\mathbf{A}$ and one for $\mathbf{b}$ (and optionally, an identity matrix $\mathbf{I}$).

2. Loop over the columns of $\mathbf{A}$ – say the index is i.



3. Loop over the rows (starting on row i) to find/choose the pivot element for that column (column i). If there is none, the matrix is singular.

4. Shuffle rows to put the row with the pivot in the right row (which will be row i) and scale the pivot row so the pivot element is 1. Don't forget to apply the same operations to $\mathbf{b}$.

5. Use the pivot row to reduce all other rows with non-zero elements in column i.
   Again, don't forget $\mathbf{b}$.

Afterwards, $\mathbf{A}$ will hold the identity matrix and $\mathbf{b}$ will hold the solution $\mathbf{x}$.

Basic methods

- Gaussian elimination with back-substitution

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a'_{00} & a'_{01} & a'_{02} & a'_{03} \\ 0 & a'_{11} & a'_{12} & a'_{13} \\ 0 & 0 & a'_{22} & a'_{23} \\ 0 & 0 & 0 & a'_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix}$$

Like GJ, but: pivots not scaled to 1, and only reduce to 0 if below the diagonal

$0 \cdot x_0 + 0 \cdot x_1 + 0 \cdot x_2 + a'_{33} x_3 = b'_3 \Rightarrow x_3 = \frac{b'_3}{a'_{33}}$

$a'_{22} x_2 + a'_{23} x_3 = b'_2 \Rightarrow x_2 = \frac{1}{a'_{22}} \left(b'_2 - a'_{23} \frac{b'_3}{a'_{33}} \right)$

Basic methods

- Gaussian elimination with back-substitution

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} a'_{00} & a'_{01} & a'_{02} & a'_{03} \\ 0 & a'_{11} & a'_{12} & a'_{13} \\ 0 & 0 & a'_{22} & a'_{23} \\ 0 & 0 & 0 & a'_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix}$$

Doesn't yield A^{-1}, but ~3x faster than GJ (and fewer operations)

$$x_i = \frac{1}{a'_{ii}} \left[b'_i - \sum_{j=i+1}^{N-1} a'_{ij} x_j \right]$$

$i = N-1, N-2, \ldots, 0$

LU decomposition: what and why

$$\begin{bmatrix} \alpha_{00} & 0 & 0 & 0 \\ \alpha_{10} & \alpha_{11} & 0 & 0 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & 0 \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ 0 & \beta_{11} & \beta_{12} & \beta_{13} \\ 0 & 0 & \beta_{22} & \beta_{23} \\ 0 & 0 & 0 & \beta_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Lower tri. Upper tria. A

N^2 eqs. in N^2+N unknown α & β $\Rightarrow$ choose $\alpha_{ii} = 1$

Always possible for A square and non-singular

$A\underline{x} = \underline{b} \Rightarrow L(U\underline{x}) = \underline{b} \Rightarrow L(\underbrace{U\underline{x}}_{=\underline{y}}) = \underline{b} \Rightarrow$ solve $L\underline{y} = \underline{b}$, then $U\underline{x} = \underline{y}$

LU decomposition: what and why

$$\begin{bmatrix} \alpha_{00} & 0 & 0 & 0 \\ \alpha_{10} & \alpha_{11} & 0 & 0 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} & 0 \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ 0 & \beta_{11} & \beta_{12} & \beta_{13} \\ 0 & 0 & \beta_{22} & \beta_{23} \\ 0 & 0 & 0 & \beta_{33} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

forward substitution $\Rightarrow \underline{y} (= U\underline{x})$

backward substitution $\rightarrow \underline{x}$

$$y_0 = \frac{b_0}{\alpha_{00}}$$

$$y_i = \frac{1}{\alpha_{ii}} \left[b_i - \sum_{j=0}^{i-1} \alpha_{ij} y_j \right]$$

$$x_{\mathrm{N}-1} = \frac{y_{\mathrm{N}-1}}{\beta_{(\mathrm{N}-1)(\mathrm{N}-1)}}$$

$$x_i = \frac{1}{\beta_{ii}} \left[y_i - \sum_{j=i+1}^{N-1} \beta_{ij} x_j \right]$$

Crout's algorithm

1. Set $\alpha_{ii} = 1$ for every i.

2. Loop over the columns j:

3. For each j, $\beta_{0j} = a_{0j}$; all β_{ij} (where $i \leq j$) can be expressed in previously calculated values:

$$\beta_{ij} = a_{ij} - \sum_{k=0}^{i-1} \alpha_{ik}\beta_{kj}$$

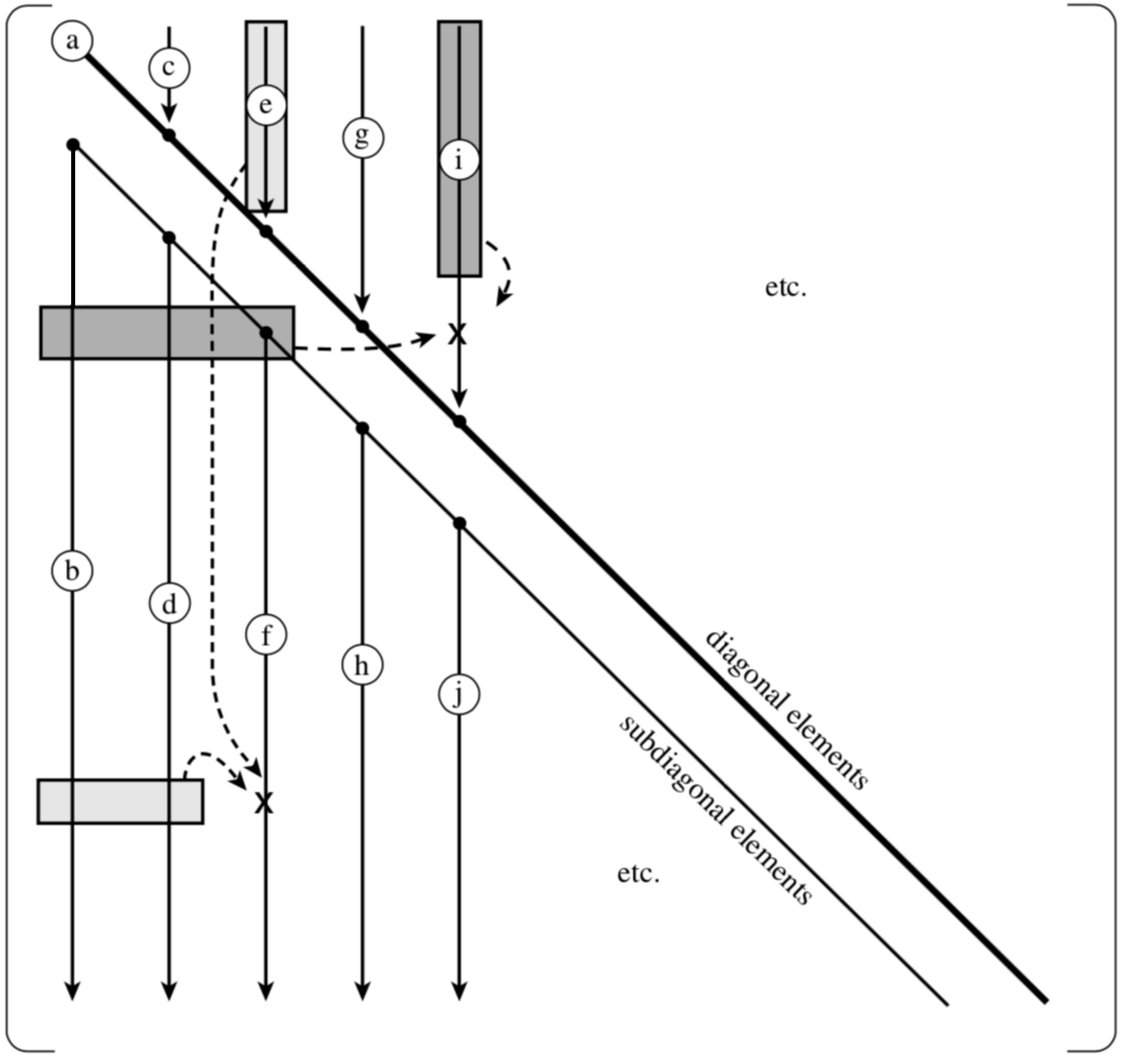
already known!

4. In the same loop, all α_{ij} (where $i > j$) can also be expressed in previously calculated values:

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=0}^{j-1} \alpha_{ik}\beta_{kj} \right)$$

5. Finish the loop over j.

Crout's algorithm



$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Overwrite! ⇓

$$\begin{bmatrix} \beta_{00} & \beta_{01} & \beta_{02} & \beta_{03} \\ \alpha_{10} & \beta_{11} & \beta_{12} & \beta_{13} \\ \alpha_{20} & \alpha_{21} & \beta_{22} & \beta_{23} \\ \alpha_{30} & \alpha_{31} & \alpha_{32} & \beta_{33} \end{bmatrix}$$

Combined "LU" matrix

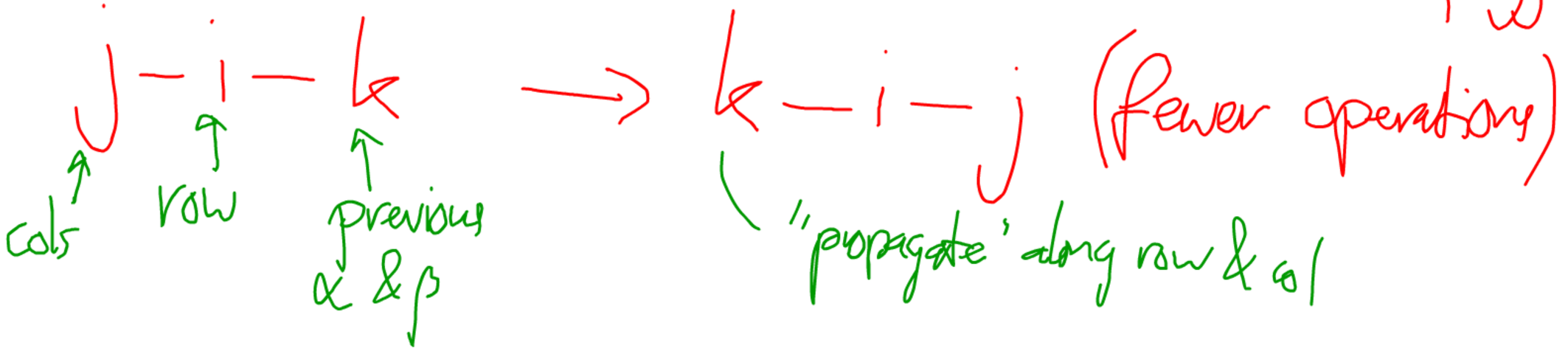
Improving Crout's algorithm

- Pivoting $\frac{1}{\beta_{jj}}$: I'd like β_{jj} to be large (more stable!)

  hold off on division by β_{jj} $\Rightarrow$ α's & β's interchangeable

  $\Rightarrow$ calculate pivot candidates in column j, then decide on pivot β_{jj}

- Changing the loop order



Improving Crout's algorithm

0. *Optional: find the largest (absolute!) coefficient on every row and store its inverse (make sure to keep track of row swaps for this variable, too). If it's zero on any row, stop, because the matrix is singular.*

1. Loop over k:

2. Using k as the column index, loop over rows $i \geq k$ to find the row with the largest (absolute!) pivot candidate, $i_{\max}$ *(if you did 0., compare the pivot candidates multiplied by the inverse max of that row)*.

3. If $i_{\max} \neq k$, then pivot (i.e. swap rows) by looping over all columns j and swapping elements $LU_{i_{\max} j}$ with elements LU_{kj}.

4. In your indexing array, keep track of what $i_{\max}$ was for each k (we need this later).

5. Now loop over $i > k$ and apply: $LU_{ik} = LU_{ik} / LU_{kk}$

6. Inside that loop, loop over $j > k$ and apply: $LU_{ij} = LU_{ij} - LU_{ik} LU_{kj}$

7. Finish the loop over i.

8. Finish the loop over k.

Solving a system

1. Set $x_i = b_i$ for all i.

2. Loop over row i in $\mathbf{b}$, from 0 through $N-1$:

3. Apply the forward substitution from before (slide 11), but don't forget that rows may have been swapped! So use the indexing array you made with the LU decomposition to swap the b_i ($= x_i$), and keep in mind that $i_{\max} \geq i$ (we never have to look back).

4. Finish the loop over i, then start a new (backwards) loop over i from $N-1$ through 0:

5. Apply the back-substitution from before (slide 11). You already swapped the x_i in the previous loop, so there's no need to swap them again.

6. Finish the (second) loop over i.

Swap x[i] and x[indx[i]] *before* applying the forward substitution to x[i]

use x[indx[i]] instead of x[i]

Further improvements

GJ: $\sim N^3$ operations, LU decomp.: $\frac{1}{3}N^3$, but solving for (new) $\underline{b}$: $\mathcal{O}(N^2)$

(1) $A\underline{x} = \underline{b} \longrightarrow AX = B$ (matrices of column vectors)

(2) $\Rightarrow$ inverse of A by $B = I$ $\left(AA^{-1} = I\right)$

(3) $\mathrm{Det}(A) = |A| = \prod_{j=0}^{N-1} \beta_{jj}$

If I did d row swaps: $|A| = (-1)^d \prod_j \beta_{jj}$

keep track of d, or parity: $1 \to -1 \to 1 \to -1 \to \ldots$

Iterative improvements

unknown errors

$A\underline{x} = \underline{b}$, found imperfect $\underline{x}' \approx \underline{x}$; $\underline{x}' = \underline{x} + \underline{\delta x}$

$A\underline{x}' = A(\underline{x} + \underline{\delta x}) = \underline{b} + \underline{\delta b}$

uses same A ⇒ same LU!

$\Rightarrow A\underline{\delta x} = \underline{\delta b} = A\underline{x}' - \underline{b} \Rightarrow$ solve for $\underline{\delta x}$

$\Rightarrow$ improved solution $\underline{x}'' = \underline{x}' - \underline{dx}$

(iterate!)

Singular value decomposition

- Least-squares problems : A is singular (no exact, unique solution)

$\Rightarrow \hat{x}$ such that $\|b - A\hat{x}\|^2$ is minimized

solution: $\hat{x} = (A^T A)^{-1} A^T b$

$A^T A$: Gram matrix

$(A^T A)^{-1}$: pseudo-inverse

- Three-matrix decomposition

$A = UWV^T$: U orthogonal $M \times N$ matrix, W non-neg. diagonalm. $(N \times N)$

V orthogonal $N \times N$.

$U^T = U^{-1} \rightarrow \hat{x} = V W^{-1} U^T b$

$$W = \begin{pmatrix} w_0 & & \emptyset \\ & w_1 & \\ \emptyset & & \ddots \end{pmatrix} \Rightarrow W^{-1} = \begin{pmatrix} 1/w_0 & & \emptyset \\ & 1/w_1 & \\ \emptyset & & \ddots \end{pmatrix}$$

A singular $\Leftrightarrow$ 1 or more $w_i = 0$

$\Rightarrow$ set $\frac{1}{w_i} = 0$ if $w_i = 0$

Singular value decomposition

e.g. Hilbert matrix

- Singularities and ill-conditioned matrices

Condition: $\text{cond}(A) = \frac{\max(w_i)}{\min(w_i)}$

$\rightarrow (w_i = 0)$: singular

$\rightarrow \gg 1 \; (w_i \approx 0)$: ill-conditioned (= "close to singular")

- Range, nullspace and their relevance

↳ collection of $A\underline{x}$ for all $\underline{x}$ (dim: rank)

if singular: rank $< \min(M, N)$, nullspace: all $\underline{x}$ such that $A\underline{x} = \underline{0}$

If I have solution to $A\underline{x} = \underline{b}$, and I know a $\underline{x}'$ in nullspace:

$A(\underline{x} + \underline{x}') = \underline{b} + \underline{0}$

The columns of U at non-zero w_i span the range of A, while the columns of V at w_i=0 span the nullspace of A!
This is why setting 1/w_i=0 works for finding the LSQ solution: zero nullspace contribution, so "closest" solution.

Special systems

If matrix is "special" (known form/pattern of sparseness / tridiagonal)
then exploit this to simplify
(e.g. special decomposition)

Bonus: Other decompositions

- Cholesky : only for square ($M=N$), symmetric ($A=A^T$), positive-definite ($\underline{v}\cdot(A\underline{v})>0$ for all $\underline{v}$) A
  → "special" LU decomp with $U=L^T \Rightarrow LL^T=A$
  (eqs. simplify greatly) ⇒ exactly fails if A is not pos.-def. (if it is square & sym.) ⇒ use this to test for pos. def. ness
- QR
  → $A=QR$, R is upper triagonal (like U), Q is orthogonal (so $Q^TQ=I$)
  ⇒ unlike LU, this works for non-square A

Next week:
Integrating functions