Week 6: aim to cover

- Numerical linear algebra: Gauss Elimination with Partial Pivoting (GEPP), operations count (Lecture 11)
- Newton's method, 2D arrays (matrices) in MATLAB (Lab 6)
- LU factorization, special matrices (Lecture 12)

Trefethen's Maxims

In principle, the Taylor series of a function of n variables involves an n-vector, an $n \times n$ matrix, an $n \times n \times n$ tensor, and so on. Actual use of orders higher than two, however, is so rare that the manipulation of matrices is a hundred times better supported in our brains and in our software tools than that of tensors.

The problem

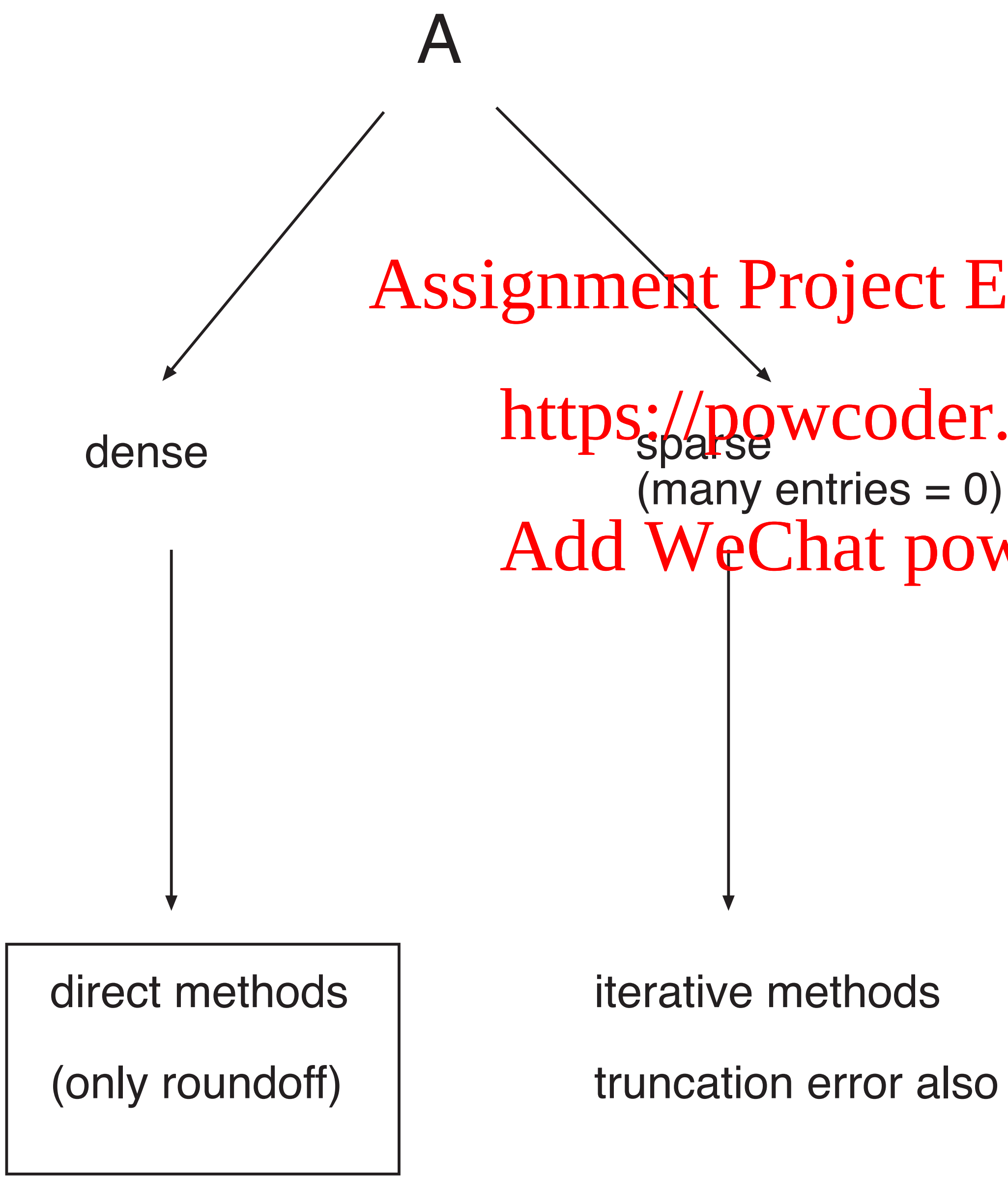
Given $\mathbf{A}$: $n \times n$ square matrix
$\mathbf{b}$: $n \times 1$ matrix (column vector)

find $\mathbf{x}$ (column vector) where

$$\mathbf{Ax} = \mathbf{b}$$

We assume $\mathbf{A}$ is nonsingular so a unique solution exists.

Direct vs. iterative methods



In MATLAB

it's very easy to solve a linear system of equations

1. define the matrix **A**
   e.g. `A = rand(100,100)`
2. define the RHS column vector
   e.g. `b = rand(100,1)`
3. then solve with backslash
   e.g. `x = A\b`
   MAGIC

did it solve correctly?

compute `b - A*x`

We now explore how this magic is performed ...

Gauss elimination

Recall from Linear Algebra:
Use row operations to reduce augmented system to **upper triangular form** then solve by back-substitution.

Note: no need for Gauss-Jordan elimination (reduction to reduced row echelon form) — it takes more work!

As an algorithm:
Denote original $\mathbf{A}$ by $\mathbf{A}^{(1)} = [A_{ij}^{(1)}]$
original $\mathbf{b}$ by $\mathbf{b}^{(1)} = [b_i^{(1)}]^T$

Step 1:

Assume $A_{11}^{(1)} \neq 0$

Define multipliers $l_{i1} = \frac{A_{i1}^{(1)}}{A_{11}^{(1)}}, \quad i = 2..n$

Put zeroes below A_{11}:

$$A_{ij}^{(2)} = A_{ij}^{(1)} - l_{i1} A_{1j}^{(1)}, \quad i, j = 2..n$$

$$b_i^{(2)} = b_i^{(1)} - l_{i1} b_1^{(1)}, \quad i = 2..n$$

We now have

$$\begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & \cdots & A_{2n}^{(2)} \\ \vdots & & & \\ 0 & A_{n2}^{(2)} & \cdots & A_{nn}^{(2)} \end{pmatrix} \mathbf{x} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

Repeat this step.

After k steps

$$\mathbf{A}^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$$

$$\mathbf{A}^{(k)} = \begin{pmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & \ddots & \cdots & A_{2n}^{(2)} \\ \vdots & 0 & \ddots & \cdots & \vdots \\ \vdots & \vdots & A_{kk}^{(k)} & \cdots & A_{kn}^{(k)} \\ 0 & 0 & A_{nk}^{(k)} & \cdots & A_{nn}^{(k)} \end{pmatrix}$$

Assume $A_{kk}^{(k)} \neq 0$

Define multipliers $l_{ik} = \frac{A_{ik}^{(k)}}{A_{kk}^{(k)}}, \quad i = k+1..n$

Put zeroes below A_{kk}:

$$A_{ij}^{(k+1)} = A_{ij}^{(k)} - l_{ik} A_{kj}^{(k)}, \quad i,j = k+1..n$$

$$b_i^{(k+1)} = b_i^{(k)} - l_{ik} b_k^{(k)}, \quad i = k+1..n$$

After n steps

we have

$$\begin{pmatrix} A_{11}^{(1)} & \cdots & A_{1n}^{(1)} \\ 0 & & \\ 0 & \cdots & A_{nn}^{(n)} \end{pmatrix} \mathbf{x} = \begin{pmatrix} b_1^{(1)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}$$

where $A^{(n)}$ is upper triangular; let's call it $\mathbf{U}$ for 'upper'

$$\mathbf{Ux} = \mathbf{b}^{(n)} = \mathbf{g}$$

Triangular system

Solve this by back-substitution

$$x_n = g_n / U_{nn}$$

$$x_k = \frac{1}{U_{kk}}[g_k - \sum_{j=k+1}^{n} U_{kj} x_j], \quad k = n-1..1$$

These formulae $\rightarrow$ algorithm for Gauss elimination (ready for coding)
BUT

Pivoting

1. what if $A_{kk}^{(k)} = 0$?
   $A_{kk}^{(k)}$ is the **pivot**
   Remedy: swap rows
2. in order to put zeroes under pivot, we subtract rows
   $\implies$ we can amplify roundoff error if we subtract 2 large numbers (most common if the multiplier is large)

Gauss elimination is potentially vulnerable to subtractive cancellation!

Example

an extreme 2×2 case

Partial pivoting

Remedy: choose pivot so that multipliers are never large
— called a **pivoting strategy**.

Simplest pivoting strategy (and usually enough)
Partial pivoting

At step k,

- look at elements $A_{kk}^{(k)}$ and below
- choose the largest of these in magnitude to be the new pivot e.g. $A_{lk}^{(k)}$
- swap rows l and k

$\Longrightarrow$ multipliers formed from new pivot satisfy $| l_{ik} | < 1$

This also handles zero pivots

Note: don't actually swap rows; swap pointers or row indices

Example

Back to our extreme case

Other pivoting strategies exist but partial pivoting is regarded as usually sufficiently stable.

Gauss Elimination with Partial Pivoting (GEPP) is the default algorithm for solving linear systems.

Operations Count for Gaussian Elimination

Measure by number of multiply/divides.

Assume no pivoting required.
1st stage: for each $i = 2..n$

form $l_{i1} = \frac{A_{i1}^{(1)}}{A_{11}^{(1)}}$ $\quad 1\div$

$A_{ij}^{(2)} = A_{ij}^{(1)} - l_{i1} A_{1j}^{(1)}, \quad i, j = 2..n \quad n - 1\times$

$b_i^{(2)} = b_i^{(1)} - l_{i1} b_1^{(1)} \quad 1\times$

$\rightarrow (n-1)(n+1) \times /\div$ for 1st stage (inc. $n-1$ for $\mathbf{b}$)

2nd stage

$\vdots$

$n \mapsto n-1$

$\to (n-2)(n) \times / \div$ for 2nd stage (inc. $n-2$ for $\mathbf{b}$)

$\vdots$

last stage: $n = 2$

$\to (1)(3) \times / \div$ for last stage (inc. 1 for $\mathbf{b}$)

Total so far:

$$\sum_{k=1}^{n}(k^2-1) = \frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{6}n$$

inc. $\sum_{k=1}^{n-1} k = \frac{1}{2}n^2 - \frac{1}{2}n$ for $\mathbf{b}$

Solve triangular system

Now back-substitute

$x_n = g_n / U_{nn}$ $\qquad$ $1 \div$

$x_{n-1} = [g_{n-1} - U_{n-1,n} x_n] / U_{n-1,n-1}$ $\qquad$ $2 \times / \div$

$\vdots$

$x_1 = \cdots$ $\qquad$ $n \times / \div$

$\rightarrow$ total for back-substitution $\sum_{k=1}^{n} k = \frac{1}{2} n^2 + \frac{1}{2} n$

Total work

$$\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{6}n + \frac{1}{2}n^2 + \frac{1}{2}n$$

$$= \frac{1}{3}n^3 + n^2 - \frac{1}{3}n \approx \frac{1}{3}n^3$$

of which

$$\frac{1}{2}n^2 - \frac{1}{2}n + \frac{1}{2}n^2 + \frac{1}{2}n = n^2$$

comes from processing RHS vector **b**

Important: Work for Gauss elimination: $\approx \frac{1}{3}n^3$ operations.
Work for each RHS vector $= n^2$ operations

Many RHS vectors

So to solve

$$\mathbf{Ax} = (\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_k)$$

requires $\approx \frac{1}{3}n^3 + kn^2$ operations.

By comparison: **suppose we use inverse of matrix**

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

To find $\mathbf{A}^{-1}$, solve

$$\mathbf{Ax} = \mathrm{I} = (\mathrm{e}_1, \mathrm{e}_2, \cdots, \mathrm{e}_n)$$

takes $\approx \frac{1}{3}n^3 + n \cdot n^2 \approx \frac{4}{3}n^3$ operations
actually can be done in $\approx n^3$ operations

Moral: DON'T FORM MATRIX INVERSE — 3 times more work than solving the system!

End of Lecture 11