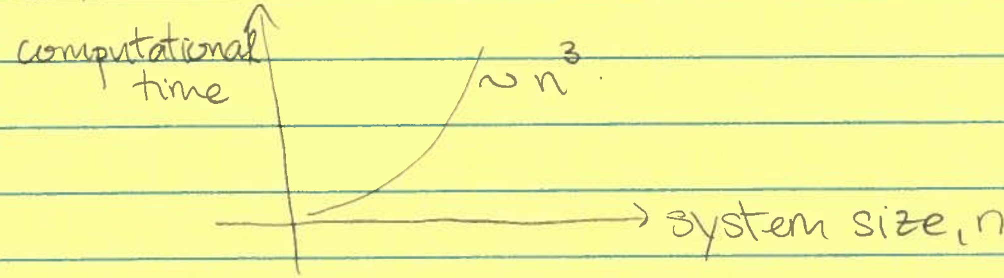
L17 Iterative techniques: Jacobi Method

MOTIVATION Direct techniques, ie variants of Gaussian elimination, take $O(n^3)$ floating point operations to solve $Ax=b$.



This is fine for a small system, but many cases of practical interest are not small.

Ex Solve

$$\frac{\partial c}{\partial t} = \frac{\partial^2 c}{\partial x^2} \qquad (*)$$

on the computer.

Sol^n Divide space and time into grids, $\{x_j\}$ and $\{t_n\}$, and let

$$c_j^n = c(x_j, t_n).$$

Then $(*)$ is approximated by:

$$\frac{c_j^{n+1} - c_j^n}{\Delta t} = \frac{c_{j+1}^{n+1} - 2c_j^{n+1} + c_{j-1}^{n+1}}{\Delta x^2}$$

Bring "n+1" terms to L.H.S., and put $r = \Delta t / \Delta x^2$:

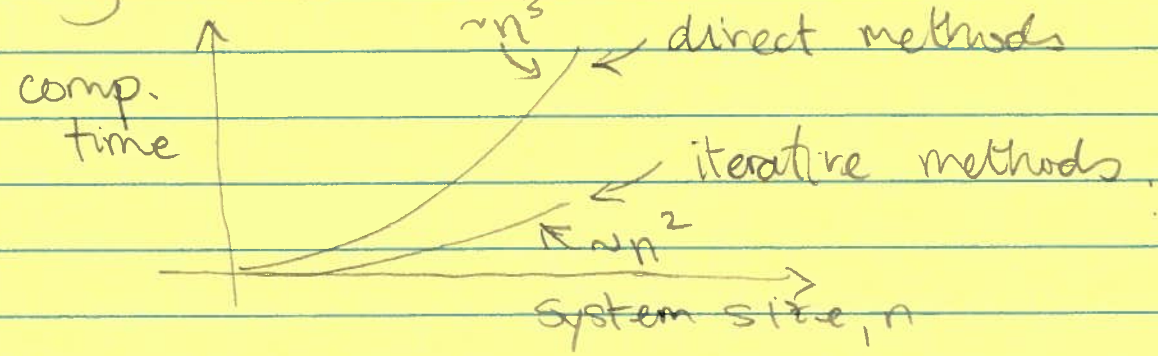
$$-r c_{j-1}^{n+1} + (1+2r) c_j^{n+1} - r c_{j+1}^{n+1} = c_j^n \quad (**)$$

This is a linear system that must be solved @ each time pt.

But if (**) is to be a good approx. of (*), we need a fine space grid, ie. the size of the system in (**) must be <u>large</u>.

G.E. could be prohibitively expensive. Is there a better way? Namely, could we get computational time to scale more favorably with system size than n^3?

Two of those three factors of n come from matrix multiplication and are unavoidable. The third factor comes from having to eliminate n columns during forward elimination. We will study methods that replace forward elimination with "iteration". Fortunately, for many systems, the # "iterations" will not scale with system size n, making "iterative techniques" much faster than "direct methods" for solving $Ax=b$;



ITERATIVE METHODS

The basic idea behind iterative methods is easy to state:

Consider : $Ax = b$. (1)

Let

$$A = S - T$$

Then (1) $\Rightarrow$

$$Sx = Tx + b$$

$$\Rightarrow \quad x = S^{-1}(Tx + b)$$

This tells us that x is the fixed point of

$$g(x) = S^{-1}(Tx + b). \qquad (2)$$

Recall from L4 that, if the sequence $\{x_k\}$, defined by

$$x_{k+1} = g(x_k) \qquad (3)$$

converges to x, then x is the fixed pt of g.

Thus we may use fixed-point iteration to solve (approximately) Eq. (1).

But, how do we compute x_{k+1} given x_k?

One approach is to compute $g(x_k)$, which involves inverting a matrix (2).

Another approach is to observe that (2) and (3) together imply that:

$$x_{k+1} = S^{-1}(Tx_k + b)$$

$$\Rightarrow \quad Sx_{k+1} = Tx_k + b.$$

Since we are given x_k, we know the RHS! Thus, to get x_{k+1}, all we have to do is solve the linear system:

$$Sx_{k+1} = \hat{b} \qquad (4)$$

where:

$$\hat{b} = Tx_k + b.$$

Recall that we can choose S to be anything we like! If we choose it to be diagonal or triangular then we can solve it w/ $O(n^2)$ ops. This is much better than $O(n^3)$ ops needed to solve $Ax=b$. The trade-off is that we must solve (4) many times. We will discuss how many in the next few lectures.

Let's agree on some notation to make things clearer. Let:

$$A = L + D + U$$

where

L = lower triangular part of A (and zero elsewhere)

D = diagonal part of A (and zero elsewhere)

U = upper triangular part of A (and zero elsewhere).

Concretely:

$$\underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 0 & \cdots & & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & & \ddots & \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}}_{L} + \underbrace{\begin{bmatrix} a_{11} & & & \\ & a_{22} & & 0 \\ & 0 & \ddots & \\ & & & a_{nn} \end{bmatrix}}_{D} + \underbrace{\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ & 0 & \cdots & a_{2n} \\ & 0 & \ddots & a_{n-1,n} \\ & & & 0 \end{bmatrix}}_{U}$$

Ex The simplest S we can think of is

$$\left.\begin{aligned} S &= I \\ \Rightarrow \quad T &= I - A \end{aligned}\right\} \text{ie. } A = \underbrace{I}_{S} - \underbrace{(I-A)}_{T}.$$

ie $$x_{k+1} = (I-A)x_k + b$$

The Notebook shows how well this simple scheme can work!

JACOBI METHOD

Another simple S that comes to mind is $S = D$, ie.

$$A = \underbrace{D}_{S} - \underbrace{(-L-U)}_{T}. \qquad (5)$$

Then (4) $\Rightarrow$

$$D x_{k+1} = -(L+U) x_k + b$$

or:

NOTE: I've moved the iteration index from subscript to superscript.

$$\boxed{a_{ii} x_i^{(k+1)} = -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j^{(k)} + b_i .}$$

This algorithm is known as the Jacobi Method.

Eq (5) represents one end of the spectrum of possible splittings $A = S - T$. At the other end is the splitting

$$A = \underbrace{A}_{S} - \underbrace{0}_{T}$$

in which case (4) reduces to

$$A x_1 = b \qquad , x_0 = \text{anything}$$

ie. within one iteration, an exact

sol^n is found! The problem, of course, is that finding x_1 is equivalent to solving $Ax=b$ outright, ie. we have not produced a simpler system to solve, as we did in the Jacobi Method.

Thus, there is a trade-off between reducing the time complexity of the system at the expense of introducing iterations.