Lecture I MAT 228A — w/ Bob Guy 9/22

Standard Model PDEs:

1) Advection equation

prototypical ex. of a hyperbolic eqn → $u_t + c\,u_x = 0$

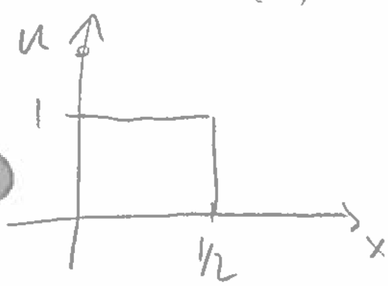
2) Diffusion eqn: (heat) $u_t = D\,u_{xx}$ ← prototypical parabolic eqn.

3) Poisson eqn: $u_{xx} = f$ ← prototypical elliptic eqn.

1) $\begin{cases} u_t + u_x = 0, & x > 0 \\ u(0,t) = 1, \quad u(x,0) = \begin{cases} 1, & x < 1/2 \\ 0, & x \geq 1/2 \end{cases} \end{cases}$

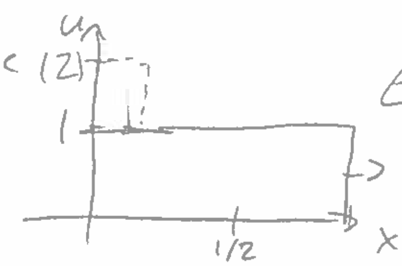
(2)

← discontinuity hard to deal w/ numerically



t=0

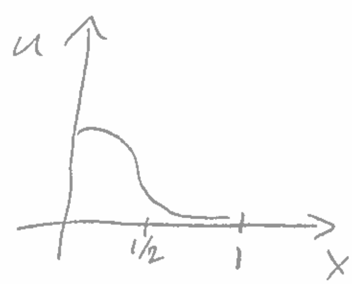
later in time →



discontinuity exists for all time!!

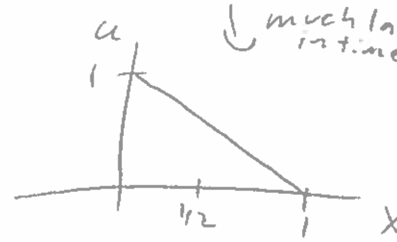
2) $\begin{cases} u_t = u_{xx}, & 0 < x \leq 1 \\ u(0,t) = 1, \quad u(1,t) = 0 \\ u(x,0) = \begin{cases} 1, & x < 1/2 \\ 0, & x \geq 1/2 \end{cases} \end{cases}$

later in time →



↓ much later in time

super easy to represent numerically →



* Need different schemes for numerically rep'ing time & space for the different problems

In 228A, we will focus on the Poisson eqn, which is time independent

Where does the Poisson eqn. come up?

1. Steady-state diffusion problems

Eg: u is a concentration: $u_t = \underbrace{D u_{xx}}_{\text{transport by diffusion}} + \underbrace{f}_{\text{input}}$

At steady-state, $u_t \to 0$ $\Rightarrow$ $-D u_{xx} = f$

Reaction-diffusion eqn: $u_t = D u_{xx} - ku + f$

Steady-state: $-D u_{xx} + ku = f$

2. Electrostatics:

Eg: $\vec{E}$ is a steady electric field, ρ is a charge distribution, ε_0 is a parameter

$\nabla \cdot \vec{E} = \rho / \varepsilon_0$ & $\nabla \times \vec{E} = 0 \to$ there is a potential φ s.t.

$E = -\nabla \varphi$

Thus $\nabla \cdot (-\nabla \varphi) = \rho/\varepsilon_0 \Rightarrow -\Delta \varphi = \rho / \varepsilon_0$

3. Potential Flow:

$\nabla \cdot \vec{u} = 0$, $\vec{u}$ is a velocity

divergence free / incompressible

& we also have $\nabla \times \vec{u} = 0$ no vorticity

$\Rightarrow \vec{u} = \nabla \varphi \Rightarrow \nabla \cdot \vec{u} = \boxed{\Delta \varphi = 0}$

4. Low Reynolds # Flow: very small length scales, e.g. bacteria swimming

drag-dominated, no inertia

$$\left.\begin{matrix} \mu \Delta \vec{u} - \nabla p = 0 \\ \nabla \cdot \vec{u} = 0 \end{matrix}\right] \text{ stoke's eqns.}$$

take div. of first eqn

$-\Delta p = 0$

Poisson Eqn:

$$u_{xx} = f \quad \text{1D}$$

$$u_{xx} + u_{yy} = f \quad \text{2D (cartesian)}$$

$$\Delta u = f$$

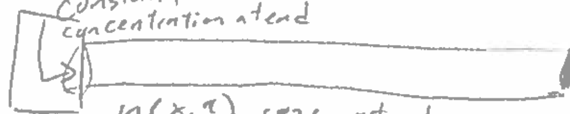
$$\text{or } \nabla^2 u = f$$

+ need a domain, + need boundary conditions

1D - Steady state diffusion eqn:

$u_t = D u_{xx} + f$

constant concentration at end



cap end $\Rightarrow u_x(l,t) = 0$

$u(x,t)$ conc. at pt. x at time t

$u = u_0 \Rightarrow u(0,t) = u_0$ (Dirichlet BC)

How does capped end translate to a BC?

Given (a,b), $q(t) = \int_a^b u(x,t)\,dx$ = total amt. of chemical in interval (a,b).

Consider $u_t = D u_{xx} \to \int_a^b u_t\,dx = \int_a^b D u_{xx}\,dx$

$$\frac{d}{dt} q(t) = D\left(u_x(b,t) - u_x(a,t)\right)$$

transport rates on boundary of (a,b)

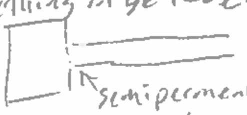
$J = -D u_x$ is the diffusive flux. capped end $\Rightarrow$ no diffusive flux!

(at end) $u_x(0,t) = 0$

Neumann B.C.

What about a given flux? $-D u_x(0,t) = g$ (non-homogeneous Neumann B.C.)

something in between?



semipermeable membrane

$J = \alpha(u_r - u_\ell)$ right & left of membrane $\Rightarrow$ flux balance:

$$-D u_x(0,t) = \alpha(u(0,t) - u_0)$$

$\Rightarrow$ Robin BC $\quad \alpha u(0,t) + D u_x(0,t) = \alpha u_0$

Solving Poisson eqn need Dirichlet, Neumann, or Robin BC's.

228A - 9/27 lecture 2

$u_{xx} = f$ on $(0,1)$
$u(0) = u(1) = 0$

Fourier series soln: What if $f = -(n\pi)^2 \sin(n\pi x)$, $n = 1, 2, \ldots$
Soln $u = \sin(n\pi x)$

Reframe higher level: $Lu = f$ w/ L operator $\frac{d^2}{dx^2}$ on space of fns. satisfying $u(0) = u(1) = 0$.

$L(\sin(n\pi x)) = -(n\pi)^2 \sin(n\pi x)$
$\Rightarrow u_n = \sin(n\pi x)$, $n = 1, \ldots$ are eigenfunctions of L w/ eigenvalues $\lambda_n = -n^2\pi^2$
These eigenfns are orthogonal in $L^2[0,1]$ ← square integrable fns.
w/ inner prod: $\langle f(x), g(x) \rangle = \int_0^1 f(x) g(x)\, dx$
The eigenfns form a complete set in $L^2[0,1]$. (w/o any BC's - FS will always converge in L^2, def not pointwise
For any $f \in L^2[0,1]$, $f(x) = \sum_{n=1}^{\infty} a_n \sin(n\pi x)$
Know soln can be written in form $u(x) = \sum_{n=1}^{\infty} \beta_n \sin(n\pi x)$
We can compute a_n by orthogonality: $a_n = 2\int_0^1 f(x) \sin(n\pi x)\, dx = \frac{\langle f, \sin n\pi x \rangle}{\langle \sin n\pi x, \sin n\pi x \rangle}$
a_n is the projection of f onto the eigenspace spanned by $\sin(n\pi x)$.

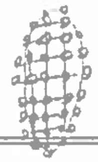
$L\left(\sum_{n=1}^{\infty} \beta_n \sin(n\pi x)\right) = \sum_{n=1}^{\infty} a_n \sin(n\pi x)$
$\sum_{n=1}^{\infty} \lambda_n \beta_n \sin(n\pi x) = \sum_{n=1}^{\infty} a_n \sin(n\pi x)$
using orthogonality $\Rightarrow \beta_n = \frac{a_n}{\lambda_n}$.

Same idea on a 2D rectangle: $\Delta u = u_{xx} + u_{yy} = f$ on $(0,1) \times (0,1)$
$u = 0$ on the boundary

Eigenfns: $u_{n,m} = \sin(n\pi x)\sin(m\pi y)$, $n, m = 1, 2, \ldots$
$\lambda_{n,m} = -(n^2 + m^2)\pi^2$

Finite Difference Methods — Discretize space

Given PDE, domain, & boundary conditions



- Discretize the domain
- Represent functions by their values at the points
- Use discrete values to approximate derivatives using algebraic formula

$\Rightarrow$ Get an algebraic eqn $A\vec{u} = \vec{b}$

Finite Element Methods — Discretize function space

Reformulate the problem as a variational problem

$$\Delta u = f \; ; \quad F(u) = \int_\Omega \tfrac{1}{2} \nabla u \cdot \nabla u + uf \, dx$$

Find $u \in S$ to minimize $F(u)$, where S is the space of admissable fns.

Assume the minimization problem gives the PDE.

Now discretize the function space S, ie. pick a finite-dim'l space $S_h \subset S$.

so that for $u_h \in S_h$, $u_h = \sum_{k=1}^{n} u_k \varphi_k(x)$

Ex:

φ's are locally supported, small overstep w/ other φ's
these are a basis for piecewise linear fns.

Solve the minimization problem "exactly" on the finite-dim. space S_h.

$$F(u_h) = \tfrac{1}{2} \sum_{i,j} A_{ij} u_i u_j + \sum_i b_i u_i \; , \quad \text{where } A_{ij} = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \, dx \leftarrow$$ mostly 0's si. φ's loc. suppo

$$\& \quad b_i = \int_\Omega f \varphi_i \, dx$$

The unknowns are the u_i's $\rightarrow$ coefficients for expansion of min. soln u_h

If there is a minimum, it must be true that $\frac{\partial F}{\partial u_i} = 0 \;\; \forall i \Rightarrow \nabla F = 0$ at min

computing this, get $A\vec{u} = \vec{b}$.

The structure of this is very similar to that of a Finite Difference method.

Spectral Methods

Use approx: $u_n(x) = \sum_{k=1}^{n} a_k \varphi_k(x)$

generally, the φ's are not locally supported (e.g. $\varphi_k = \sin(k\pi x)$)

From here, solve the variational problem

↳ changes structure of $A\vec{u} = \vec{b}$, structure of A is dense.

Or set of points, solve $(\Delta u)(x_j) = f(x_j)$.

get $\sum_{j=1}^{N} a_j \varphi_j''(x_i) = f(x_i)$ for $i = 1, \ldots, N$

$\Rightarrow$ linear system $A_{ij} = \varphi_j''(x_i)$ still dense $\rightarrow A\vec{x} = \vec{b}$.

9/29 - MAT 228A - Lecture 3

Finite-Difference Methods : idea - approximate derivatives using function values at points

Approx. derivative w/ a difference:

Forward difference operator: $D_+ u(x) = \dfrac{u(x+h) - u(x)}{h}$, h fixed

Backward difference operator: $D_- u(x) = \dfrac{u(x) - u(x-h)}{h}$

How accurately do these approximate $\frac{d}{dx}$?

Error $E = D_+ u(x) - \frac{d}{dx} u(x)$

Expand using Taylor series as $h \to 0$

$$u(x+h) = u(x) + h u'(x) + \frac{h^2}{2} u''(x) + \frac{h^3}{6} u'''(x) + \dots$$

$$D_+ u(x) = u'(x) + \frac{h}{2} u''(x) + \frac{h^2}{6} u'''(x) + \dots$$

$$E = D_+ u(x) - \frac{d}{dx} u(x) = \frac{h}{2} u''(x) + \frac{h^2}{6} u'''(x) + \dots$$

Assume u'' is bounded, then $E = O(h)$.

Same idea for D_-: $u(x-h) = u(x) - h u'(x) + \frac{h^2}{2} u''(x) - \frac{h^3}{6} u'''(x) + \dots$

$$D_- u(x) = u'(x) \underbrace{- \frac{h}{2} u''(x) + \frac{h^2}{6} u'''(x) + \dots}_{}$$

error $E = O(h)$. ← error is same but opp. sign as D_+ at leading order!

Consider $D_0 u(x) = \frac{1}{2}(D_+ + D_-) u(x) = \frac{1}{2}\left[\dfrac{u(x+h) - u(x)}{h} + \dfrac{u(x) - u(x-h)}{h}\right]$

centered difference operator

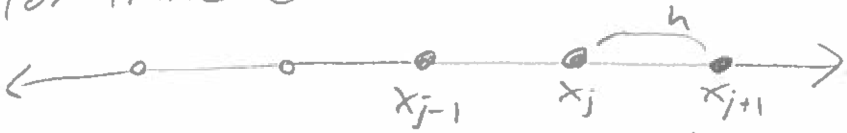
$$= \frac{u(x+h) - u(x-h)}{2h} = u'(x) + \underbrace{\frac{h^2}{6} u'''(x) + \dots}_{\text{error}}$$

$D_0 u(x)$ is $O(h^2)$ accurate (assuming $u'''(x)$ bounded)

Terminology: D_+ & D_- provide first-order accurate approximations to $\frac{d}{dx}$

D_0 provides second-order " " "

For fixed u, $D_+ u(x)$ can be evaluated everywhere.
For finite-difference methods, start on a discrete domain.



$x_j = jh$

$u_j \approx u(x_j)$

$(D_+ u)_j \approx \dfrac{u_{j+1} - u_j}{h}$

How do we get higher derivatives?
Apply difference operators multiple times (but which ones?)
↳ apply D_+^2, D_-^2, D_0^2, $D_+ D_0$, $D_- D_0$, $D_+ D_-$, etc. $\Rightarrow$ all approx. $\frac{d^2}{dx^2}$

2 of these are second-order accurate: D_0^2, $D_+ D_-$

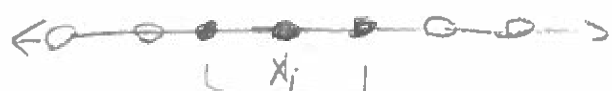
$$D_0 (D_0 u)_j = D_0\left(\frac{u_{j+1} - u_{j-1}}{2h}\right) = \frac{u_{j+2} - 2u_j + u_{j-2}}{4h^2}$$

expect wider stencil to be worse

$$D_+ (D_- u)_j = D_+\left(\frac{u_j - u_{j-1}}{h}\right) = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}$$

more local info → better approx

Different approach to approx. $u''(x_j)$:
need at least 3 pts. to det. $u''(x)$
(since 2 pts can only give you a line → 0 2nd deriv.)



May as well pick the closest pts

Take a linear combination of these 3 values:

$$(D^2 u)_j = a u_{j-1} + b u_j + c u_{j+1} \qquad a, b, c \text{ constants TBD.}$$
$$= a u(x-h) + b u(x) + c u(x+h)$$

← assume the u_j sample a cont's fn. & lift back to cont's fns to use Taylor se[ries]

TS
$$= a\left[u(x) - h u'(x) + \frac{h^2}{2} u''(x) - \frac{h^3}{6} u'''(x) + \frac{h^4}{24} u^{(4)}(x) + \dots\right]$$
$$+ b u(x) + c\left[u(x) + h u'(x) + \frac{h^2}{2} u''(x) + \frac{h^3}{6} u'''(x) + \frac{h^4}{24} u^{(4)}(x) + \dots\right]$$

← note how again the symm. saves the day and gives us more acc. for free!

$$= \underbrace{(a+b+c)}_{0} u(x) + h \underbrace{(c-a)}_{0} u'(x) + \underbrace{\frac{h^2}{2}(a+c)}_{1} u''(x) + \frac{h^3}{6}(c-a) u^{(3)}(x) + \frac{h^4}{24}(c+a) u^{(4)}(x)$$

$$\Rightarrow \left.\begin{aligned} a + b + c &= 0 \\ -ha + hc &= 0 \\ \frac{h^2}{2} a + \frac{h^2}{2} c &= 1 \end{aligned}\right] \xrightarrow{\text{solve}} \begin{aligned} a &= 1/h^2 \\ b &= -2/h^2 \\ c &= 1/h^2 \end{aligned}$$

$$\Rightarrow (D^2 u)_j = u''(x) + \frac{h^2}{12} u^{(4)}(x) + \dots$$

hence $D^2 = D_+ D_-$ is second-order accurate approx. of $\frac{d^2}{dx^2}$

note we just derived the $D_+ D_-$ operator!

$(D^2 u)_j = \frac{1}{h^2} u_{j-1} - \frac{2}{h^2} u_j + \frac{1}{h^2} u_{j+1}$

For this problem, Absolute error $= D_+ u(2) - u'(2)$ 9/29

$$\text{relative error} = \frac{D_+ u(2) - u'(2)}{u'(2)}$$

Forward differences and centered differences of $f(x) = \exp(x)$ at $x = 2$.

h	error of forward difference	ratio	error of centered difference	ratio
1.00000e+00	5.307e+00	---	1.295e+00	---
5.00000e-01	2.198e+00	2.41	3.117e-01	4.15
2.50000e-01	1.006e+00	2.19	7.721e-02	4.04
1.25000e-01	4.817e-01	2.09	1.926e-02	4.01
6.25000e-02	2.358e-01	2.04	4.812e-03	4.00
3.12500e-02	1.167e-01	2.02	1.203e-03	4.00
1.56250e-02	5.803e-02	2.01	3.007e-04	4.00
7.81250e-03	2.894e-02	2.01	7.517e-05	4.00
3.90625e-03	1.445e-02	2.00	1.879e-05	4.00

halving h results in halving errors (asymptotically)

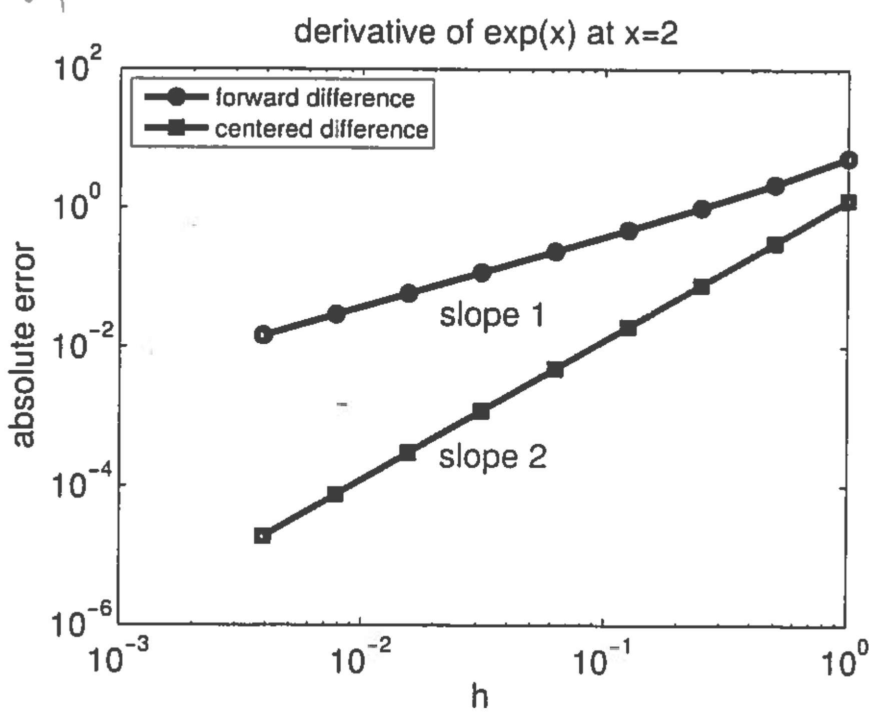
half h grid spacing should quarter the error (asymptotically)

$\frac{1}{2} \to \frac{1}{2} \Rightarrow O(h)$

$\frac{1}{2} \to \frac{1}{4} \Rightarrow O(h^2)$

Recall: D_+ is $O(h)$ accurate

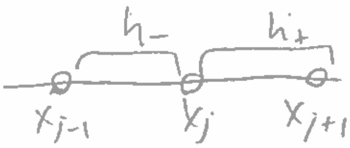
D_0 is $O(h^2)$ accurate



☆ often demonstrate the desired order of accuracy by displaying this table or this log-log plot.

$D_+ D_+$ 

this is only first-order accurate since it lacks the symmetry of our other approach

Similarly 

$h_+ \neq h_-$ also would only be 1^{st} order acc.

Expect generically that 3 point operators give only 1^{st} order approx. to the 2^{nd} derivative.

↳ can add another pt. to give another eqn/coefficient to cancel out a leading order error.

want $u^{(n)}(x_j)$

How many points m do we need to approx. n^{th} deriv. to p^{th} order accuracy,

↳ $w_1 u_1 + w_2 u_2 + \dots + w_m u_m$

Taylor series $\Rightarrow$ $= \underbrace{A_0 u(x) + A_1 u'(x) + \dots + A_n u^{(n)}(x)}_{} + A_{n+1} u^{(n+1)}(x) + \dots$

$n+1$ constraints $\Rightarrow$ at least $n+1$ points to get n^{th} derivative.

↳ solving for these constraints, we expect $w \sim \frac{1}{h^n}$

so then $A_{n+1} \sim$ size h (generically)

To get up to p^{th} order accuracy, need $m = (n+1)+(p-1)$

$\Rightarrow \underline{m = n+p}$

MAT228A - Lecture 4 - 10/4

One-D finite diff. soln. of Poisson eqn

$$\begin{cases} u_{xx} = f \\ u(0) = \alpha \\ u(1) = \beta \end{cases}$$

$j=0$ points $1-N$ $j=N+1$

$x=0$ $x=1$

$x_j = jh$

$x_{N+1} = (N+1)h = 1$

discretize interval into equally spaced points $\Rightarrow h = \frac{1}{N+1}$

$u_j \approx u(x_j) \rightarrow \dfrac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = f_j = f(x_j)$ $(*)$

discretize eqns. $\rightarrow$ for $j=1,\dots,N$

Now $\underline{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}$ eqn is of the form $Au = b$, where A is D_{+-} op & b is the f vector.

$$\left(\quad \right) \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} = \left(\underline{b} \right)$$

j^{th} eqn

$(a_{j1} \cdots a_{jN}) \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} = f_j$ should match $(*)$, hence all a_{ji} but 3 should be 0

$\Rightarrow$ For $j = 2,\dots,N-1$: $a_{jj} = -\frac{2}{h^2}$, $a_{j,j+1} = a_{j,j-1} = \frac{1}{h^2}$, $b_j = f_j = f(x_j)$

At $j=1$, $\dfrac{u_0 - 2u_1 + u_2}{h^2} = f_1$ but u_0 is known by BC $\Rightarrow \dfrac{\alpha - 2u_1 + u_2}{h^2} =$

$\Rightarrow \dfrac{-2u_1 + u_2}{h^2} = f_1 - \dfrac{\alpha}{h^2} \Rightarrow a_{11} = -\dfrac{2}{h^2}$, $a_{1,2} = \dfrac{1}{h^2}$, $b_1 = f_1 - \dfrac{\alpha}{h^2}$

hence $A = \dfrac{1}{h^2}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & \ddots & & \\ & \ddots & \ddots & 1 & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{pmatrix}$, $b = \begin{pmatrix} f_1 - \frac{\alpha}{h^2} \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N - \frac{\beta}{h^2} \end{pmatrix}$

A is tri-diagonal

What if $\vec{u} = \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix}$? Then $\dfrac{1}{h^2}\begin{pmatrix} h^2 & & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & 1 \\ & & & 1 \; -2 & 1 \\ & & & & h^2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix} = \begin{pmatrix} \alpha \\ f_1 \\ \vdots \\ f_2 \\ \beta \end{pmatrix}$ $\leftarrow$ this is weird but OK

Does A have an inverse? Yes.

How close is u_j to $u(x_j) \leftarrow$ soln to PDE?

Error at a point: $e_j^h = u_j^h - u(x_j) \rightarrow$ Error vector $\vec{e}^h = \vec{u}^h - \vec{u}_{sol}$

explicit notation of grid spacing h $\qquad$ $\vec{u}_{sol_j} = u(x_j)$

We'd like $\|e^h\| \rightarrow 0$ as $h \rightarrow 0$ for some norm.

If the error goes to zero as the mesh spacing goes to zero, the method/scheme is convergent

Vector, matrix, function norms: Consider $u(x) = 1$ on $(0,1)$

Then $\|u\|_2 = \|u\|_1 = \|u\|_\infty = 1$ efn. norms

Sample $u(x)$: $(\vec{u})_j = u(x_j)$, $\vec{u}_j = 1$ sample using N points

$$\|\vec{u}\|_2 = \left(\sum_{j=1}^{N} u_j^2\right)^{1/2} = \left(\sum^{N} 1\right)^{1/2} = \sqrt{N} \qquad \|\vec{u}\|_\infty = \max_j |u_j| = 1$$

$$\|\vec{u}\|_1 = \sum_{j=1}^{N} |u_j| = N$$

Vector norms $\Rightarrow$ values depend on length of vector

$\hookrightarrow$ not appropriate here.

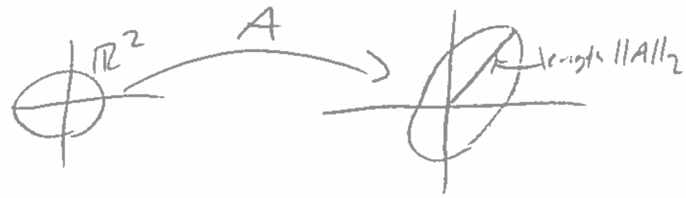
We should use discretized function norms for errors!

let $\vec{e}^h$ be a grid function: $\|\vec{e}^h\|_2 = \left(h \cdot \sum_{j=1}^{N} e_j^2\right)^{1/2}$

$\|\vec{e}^h\|_1 = h \cdot \sum_{j=1}^{N} |e_j|$, define the norms based on the mesh.

matrix norms: $A: \mathbb{R}^n \to \mathbb{R}^n$, A is an $n \times n$ matrix

$$\|A\|_\square = \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|_\square}{\|x\|_\square} = \max_{\substack{x \in \mathbb{R}^n \\ \text{s.t. } \|x\|_\square = 1}} \|Ax\|_\square :$$



$$\|A\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}| = \text{max row sum}$$

$$\|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}| = \text{max column sum}$$

$\|A\|_2 = \sqrt{\rho(A^* A)}$, where ρ is the spectral radius = modulus of largest eigenvalue

$\quad = $ largest singular value of A

Proof of $\|A\|_\infty = $ max row sum:

$\|A\|_\infty = \max_{\|x\|_\infty = 1} \|Ax\|_\infty$: $\quad \|Ax\|_\infty = \max_i \left|\sum_{j=1}^{n} a_{ij} x_j\right| \le \max_i \sum_{j=1}^{n} |a_{ij}| \cdot |x_j|$ by $\|x\|_\infty$

$$\le \max_i \sum_{j=1}^{n} |a_{ij}| \cdot 1$$

Note the max is attained at some I: $\max_i \sum_{j=1}^{n} |a_{ij}| = \sum_{j=1}^{n} |a_{Ij}|$.

So pick $x_j = \text{sign}(a_{Ij})$, then $\|Ax\|_\infty = \max_i \left|\sum_{j=1}^{n} a_{ij} x_j\right| = \sum_{j=1}^{n} |a_{Ij}|$.

Hence max row sum $\le \|A\|_\infty$, hence equality. $\square$

For any induced matrix norm,

$$\frac{\|A\vec{x}\|}{\|\vec{x}\|} \le \max_{\vec{x} \neq 0} \frac{\|A\vec{x}\|}{\|\vec{x}\|} = \|A\| \Rightarrow \|A\vec{x}\| \le \|A\| \cdot \|\vec{x}\|.$$

228A - Lecture 5 - 10/6

$u_{xx} = f$
$u(0) = \alpha$
$u(1) = \beta$

$j = 1, \dots, N$ on $[0, 1]$, $h = \frac{1}{N+1}$

$A\vec{u}^h = \vec{b}$ finite difference approx.

Know how big the error is $\vec{e}^h = \vec{u}^h - \vec{u}_{sol}$, $(\vec{u}_{sol})_j = u(x_j)$

Hope that $\|e^h\| = \mathcal{O}(h^2)$.

We get error by using our difference operator D_{+-} instead of $\frac{d^2}{dx^2}$ of $\mathcal{O}(h^2)$

↳ Error in equation → how does it translate to error in soln?

↳ Discretization error / local truncation error

In general, $\vec{u}_{sol}$ is <u>not</u> a solution to $A\vec{u} = \vec{b}$

Local truncation error $\vec{\tau}^h = A\vec{u}_{sol} - \vec{b}$

for $j = 2, \dots, N-1$, $\tau^h_j = \frac{1}{h^2}\left(u(x_{j-1}) - 2u(x_j) + u(x_{j+1})\right) - f(x_j)$

$$= \frac{1}{h^2}\left(u(x_j - h) - 2u(x_j) + u(x_j + h)\right) - f(x_j)$$

TS → $$= u_{xx}(x_j) + \frac{h^2}{12}u^{(4)}(x_j) + \text{h.o.t.} - f(x_j) = \mathcal{O}(h^2)$$

Now $A\vec{u}^h = \vec{b}$
$A\vec{u}_{sol} = \vec{b} + \vec{\tau}^h$

$A(\vec{u}^h - \vec{u}_{sol}) = -\vec{\tau}^h \Rightarrow A(\vec{e}^h) = -\vec{\tau}^h \Rightarrow \boxed{\vec{e}^h = -A^{-1}\vec{\tau}^h}$

expect error to go like $\mathcal{O}(h^2)$

A numerical scheme is <u>consistent</u> if $\vec{\tau}^h \to 0$ as $h \to 0$.

A numerical scheme is <u>convergent</u> if $\vec{e}^h \to 0$ as $h \to 0$.

For linear schemes applied to linear PDEs:

<u>Lax-equivalence Thm</u>: If a scheme is consistent & <u>stable</u>, it is convergent.

↳ errors don't separate too rapidly in time

For a time-independent BVP, stable: $A^h u^h = b^h \Rightarrow \|(A^h)^{-1}\| \le C$ for all $h \le h_0$.

$\|e^h\| = \|A^{-1}\tau^h\| \le \|A^{-1}\| \cdot \|\tau^h\| \le C \cdot \|\tau^h\| \to 0$ as $h \to 0$

if scheme is stable; if scheme is consistent

Analyze stability by bounding $\|A^{-1}\|$

Stability in 2-norm: $A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}$ is symmetric.

A symm $\Rightarrow \|A\|_2 = \rho(A) = \max_j |\lambda_j|$

A symm $\Rightarrow A^{-1}$ also symm $\Rightarrow \|A^{-1}\|_2 = \rho(A^{-1}) = \max \left|\frac{1}{\lambda_j}\right| = \frac{1}{\min|\lambda_j|}$

(Since A^{-1} has inverse eigenvalues of A)

w/ eigvals $\lambda_k = -k^2\pi^2$

Recall: $u^k(x) = \sin(k\pi x)$ is an eigenfn. of $\frac{\partial^2}{\partial x^2}$ on fns. zero at $x=0,1$.

It turns out that the eigenvectors of A are the eigenfns. of $\frac{\partial^2}{\partial x^2}$ evaluated at the grid points. (boundary mismatch doesn't matter since boundary data in $\vec{b}$)

$\Rightarrow$ Show $u_j^k = \sin(k\pi x_j)$ $(k=1,\dots,N)$ is an eigenvector, compute the eigenvalues

$$\frac{1}{h^2}\left(\sin(k\pi x_{j-1}) - 2\sin(k\pi x_j) + \sin(k\pi x_{j+1})\right)$$

$$= \frac{1}{h^2}\left(\sin(k\pi(x_j - h)) - 2\sin(k\pi x_j) + \sin(k\pi(x_j+h))\right)$$

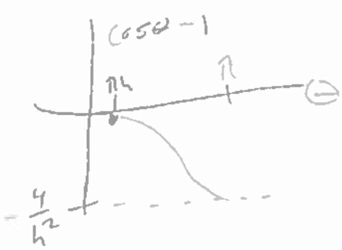
$$= \frac{1}{h^2}\left(\sin(k\pi x_j)\cos(k\pi h) - \cancel{\sin(k\pi h)\cos(k\pi x_j)} - 2\sin(k\pi x_j) + \sin(k\pi x_j)\cos(k\pi h) + \cancel{\sin(k\pi h)\cos(k\pi x_j)}\right.$$

$$= \frac{1}{h^2}\sin(k\pi x_j)\left[2\cos(k\pi h) - 2\right] = \lambda_k \sin(k\pi x_j)$$

$\Rightarrow$ eigenvalues are $\lambda_k = \frac{2}{h^2}\left[\cos(k\pi h) - 1\right]$, $k=1,\dots,N$

$$= \frac{-4}{h^2}\sin^2(k\pi h/2)$$

For k from 1 to N, $k\pi h$ goes from πh to $N\pi h = \frac{N\pi}{N+1}$



Smallest eigenvalue λ_k occurs at $k=1$

$$\lambda_1 = \frac{2}{h^2}(\cos(\pi h) - 1)$$

also for small k, $\lambda_k = -k^2\pi^2 + O(h^2)$ by $\Downarrow$

$$\Rightarrow \|A^{-1}\| = 1/|\lambda_1|$$

$\lambda_1 \to ?$ as $h \to 0$: $\lambda_1 = \frac{2}{h^2}(\cos(\pi h) - 1) = \frac{2}{h^2}\left(1 - \frac{1}{2}\pi^2 h^2 + O(h^4) - 1\right) = -\pi^2 + O(h^2)$

hence $\|A^{-1}\| = \frac{1}{\pi^2} + O(h^2) \Rightarrow \vec{e}^h = -A^{-1}\vec{\tau}^h \Rightarrow \|\vec{e}^h\| \le \|A^{-1}\|_2 \|\vec{\tau}^h\| = \left(\frac{1}{\pi^2} + O(h^2)\right)\cdot O(h^2)$.

Lecture 5 - 10/6 handout

Problem

$$u_{xx} = -\sin(3\pi x)$$
$$u(0) = 1, \quad u(1) = 0$$

Errors

$$e^h = u^h - u(x)$$

expect $e^h \approx Ch^2 \Rightarrow e^{h/2} \approx C\frac{h^2}{4}$

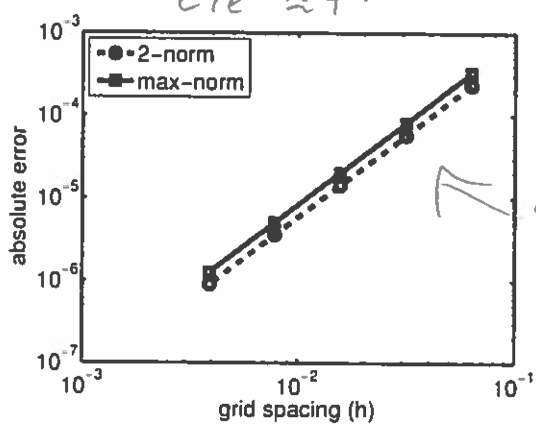
expect ratio 4

halving grid spacing

h	e2	ratio	emax	ratio
6.250000e-02	2.342e-04	---	3.312e-04	---
3.125000e-02	5.779e-05	4.053	8.173e-05	4.053
1.562500e-02	1.440e-05	4.013	2.037e-05	4.013
7.812500e-03	3.598e-06	4.003	5.088e-06	4.003
3.906250e-03	8.992e-07	4.001	1.272e-06	4.001

$e^h/e^{h/2} \approx 4$ ✓

also 2nd-order accurate in the max norm



slope 2.

need 2 spacings for 1 ratio. $e^h/e^{h/2} \approx 4$.

What if we don't know the soln?

Make your own by plugging in & computing f + B.C.

or

estimate the error by looking at successive approximations

Differences

$$d^h = u^h - u^{h/2}$$

$u^h \approx u(x) + Ch^p$

$-)\ u^{h/2} \approx u(x) + C\left(\frac{h}{2}\right)^p$

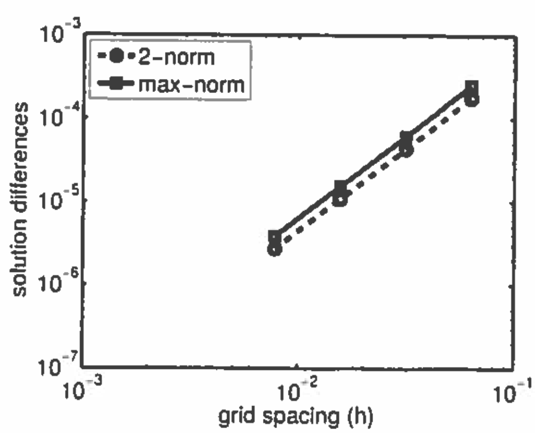
$d^h = u^h - u^{h/2} \approx C\left(h^p - \left(\frac{h}{2}\right)^p\right)$

$= Ch^p\left(1 - \frac{1}{2^p}\right)$

$\approx C_2 h^p$

$d^{h/2} \approx C_2 (h/2)^p$

h	d2	ratio	dmax	ratio
6.250000e-02	1.764e-04	---	2.495e-04	---
3.125000e-02	4.339e-05	4.066	6.137e-05	4.066
1.562500e-02	1.080e-05	4.016	1.528e-05	4.016
7.812500e-03	2.698e-06	4.004	3.816e-06	4.004



need 3 spacings for 1 ration

$d^h/d^{h/2} \approx 4$

MAT 228A - Lecture 6 — 10/11

$u_{xx} = f$, $u(0) = \alpha$, $u(1) = \beta$ $\rightarrow$ discretize to $A\vec{u} = \vec{b}$

where $A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}$. How do we solve this linear system?

Solve by Gaussian elimination $A = LU = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \text{stuff} & & 1 \end{pmatrix}\begin{pmatrix} 1 & & \text{stuff} \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$

Then $A\vec{u} = \vec{b} \Rightarrow LU\vec{u} = \vec{b} \Rightarrow L\vec{v} = \vec{b}$ & then $U\vec{u} = \vec{v}$ — How expensive is this?

Suppose A is $n \times n$, takes n^2 computations to eliminate one of the n rows

$\hookrightarrow$ Work is $\mathcal{O}(n^3)$ to factor A

Work to solve a triangular system is $\mathcal{O}(n^2)$

Our matrix has a special structure $\rightarrow$ tridiagonal & diagonally-dominant

$$\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ -1/2 & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}\begin{pmatrix} -2 & 1 & & & \\ 0 & -3/2 & 1 & 0 & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & & \\ -1/2 & 1 & & \\ & -2/3 & \ddots & \\ & & & 1 \end{pmatrix}\begin{pmatrix} -2 & 1 & & & \\ 0 & -3/2 & 1 & & \\ & 0 & -4/3 & 1 & \\ & & 1 & -2 & \ddots \\ & & & \ddots & 1 \\ & & & 1 & -2 \end{pmatrix}$$

$\swarrow$ work on each row doesn't depend on n! But n rows, so work is Cn. So work to turn A into LU is $\mathcal{O}(n)$!

For this matrix,

$\leftarrow \mathcal{O}(n)$ to factor

$\leftarrow \mathcal{O}(n)$ to forward & back-sub

This process is the Thomas Algorithm. In Matlab, $\vec{u} = A \backslash \vec{b}$ where A is sparse

We can get the inverse analytically, but $A^{-1}\vec{b}$ takes $\mathcal{O}(n^2)$ work!!! (but it is dense)

MATLAB: `n=100; e=ones(n,1); A=spdiags([e -2*e e], -1:1, n,n);`
diag entries $\uparrow$ diagonals $\uparrow$ dim'ns $\uparrow$

`Af = full(A); b=rand(n,1);`

`tic; y=A\b; toc, tic; z=Af\b; toc`
$\hookrightarrow$ 0.000994 sec $\hookrightarrow$ 0.228348 sec

`tic; B=inv(A); toc` $\rightarrow$ .001892 sec

`tic; w=B*b; toc` $\rightarrow$ .001858 sec

Last time, we showed $A\vec{e} = -\vec{\tau}$

$$\& \ \|A^{-1}\|_2 = O(1)$$

had $O(h^2)$ from discretization

$$\Rightarrow \|\vec{e}\|_2 = \|A^{-1}\vec{\tau}\|_2 \le \|A^{-1}\|_2 \|\vec{\tau}\|_2 = O(h^2)$$

~~Saw an example where $\|\vec{e}\|_\infty = O(h^2)$~~

Try norm equivalence: $\underbrace{c\|\vec{e}\|_\infty \le \|\vec{e}\|_2}_{\text{try this}} \le C\|\vec{e}\|_\infty$

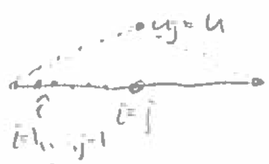
$$\|\vec{e}\|_2 = h^{1/2}\left(\sum_{j=1}^{n} e_j^2\right)^{1/2} \ge h^{1/2}|e_i| \text{ for any } i \Rightarrow \ \ge h^{1/2}\|\vec{e}\|_\infty$$

$$\Rightarrow h^{1/2}\|\vec{e}\|_\infty \le \|\vec{e}\|_2 \le Ch^2 \Rightarrow \|\vec{e}\|_\infty \le Ch^{3/2}$$ ← we can do better.

<u>Max-norm analysis:</u> $A\vec{u} = \vec{b}$ where $b_i = \begin{cases} 1 & \text{for } i=j \\ 0 & \text{for } i \ne j \end{cases}$ for some j

Then $A^{-1}b = j^{\text{th}}$ colmn. of A^{-1}

(Constructing a sort of Green's fn / inverse operator to our difference operator)



$$\begin{cases} \dfrac{u_{i-1} - 2u_i - u_{i+1}}{h^2} = 0 \\ u_0 = 0 \\ u_j = U \end{cases}$$ ← Can satisfy this w/ a linear function of x_i → for $i<j$, $u_i = \dfrac{U}{x_j} x_i$

Similar idea for $i>j$ $\quad u_i = \dfrac{U(1-x_i)}{1-x_j}$

For $i=j$, $\dfrac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = 1$

$$\frac{U x_{j-1}}{x_j} - 2U + \frac{U(1-x_{j+1})}{1-x_j} = h^2$$ ← solve this for U

$$\Rightarrow U = h(x_j - 1)x_j$$

So the soln to $\vec{u} = A^{-1}\vec{b}$ is $u_i = \begin{cases} h(x_j-1)x_i & i \le j \\ h(x_i-1)x_j & i > j \end{cases}$ ← looks like a green's fn !!

this is the i^{th} element of the j^{th} column of A^{-1}. $\quad = h(x_c - 1)x_s$

$$\Rightarrow \|A^{-1}\|_\infty = \max_i \sum_{j=1}^{n} |A^{-1}_{ij}| \le nh \le 1 \text{ since } n \text{ scales like } 1/h.$$

Hence $\|\vec{e}\|_\infty \le \|A^{-1}\|_\infty \|\vec{\tau}\|_\infty = O(h^2)$.

MAT228A - Lecture 7 - 10/13

Last time $A = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & \\ 1 & -2 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{pmatrix}$ analyzed errors in the max norm by computing the inverse $A^{-1} = B$

where $B_{ij} = h(x_{>} - 1)x_{<}$ → where $x_{>} = x_j$ for $j \geq i$

Looks like a Green's fn (kernel of inverse op to $\frac{\partial^2}{\partial x^2}$).

Recall: $A\vec{e} = -\vec{\tau} \Rightarrow \vec{e} = -A^{-1}\vec{\tau} = -B\vec{\tau} = -\begin{pmatrix} \vec{b}_1 & \vec{b}_2 & \cdots & \vec{b}_n \end{pmatrix}\begin{pmatrix} \tau_1 \\ \vdots \\ \tau_n \end{pmatrix}$

$= \sum_{j=1}^{n} \vec{b}_j \tau_j$ ← know this is $O(h^2)$

How much does truncation error at a point affect the error?

→ Consider an interior point x_j:

← graph of $\vec{b}_j$, the j^{th} column of B

↖ B_{jj} is the largest element & is $O(h)$.

Hence, each term in the sum $\sum_{j=1}^{n} \vec{b}_j \tau_j$ is $O(h^3)$. But we add up $O(\frac{1}{h})$ terms $\Rightarrow$ total error still $O(h^2)$.

What about truncation error at a point near the boundary?

$\vec{b}_1$ ← biggest element is $B_{11} = h(x_1 - 1)x_1 = h(h-1)h = O(h^2)$.

$\Rightarrow \vec{b}_1 \tau_1 = O(h^4)$ ← can get away w/ bigger truncation error near boundaries in Dirichlet problem since BC's keep you close to 0.

Neumann Boundaries

1) discretize
2) solve linear system

$x_0 = 0 \quad x_1 = h \quad x_2 = 2h \ldots$

$u_x(0) = g$

Look at discretized operator at x_1: $\frac{u_0 - 2u_1 + u_2}{h^2} = f_1$ don't know u_0 this time!

Why not discretize the BC? $u_x(0) = g \approx \frac{u_1 - u_0}{h} \Rightarrow u_0 = u_1 - gh$

$$\Rightarrow \frac{1}{h^2}\begin{pmatrix} -h & h & & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ & & & 1 \end{pmatrix}\vec{u} = \begin{pmatrix} g \\ f_1 \\ \vdots \\ \vdots \end{pmatrix} \quad \text{or} \quad \frac{1}{h^2}\begin{pmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 + \frac{g}{h} \\ f_2 \\ \vdots \\ \vdots \end{pmatrix}$$

Discretizing the BC introduces an $O(h)$ approximation
→ unobviously, this makes the total error $O(h)$.
↳ boundary errors didn't matter in Dirichlet problem b/c of its Green's fn. but this problem has a different Green's function!

This method is only first-order accurate.

Another way to discretize the Neumann BC:

Note that $\frac{u_1 - u_0}{h} = u_x(0) + O(h)$, but $\frac{u_1 - u_0}{h} = u_x(h/2) + O(h^2)$
(as a centered difference approx.)

Imagine extending the domain by adding ghost points $\to$ $x_{-1} = -h \quad x_0 \quad x_1 = h$

~~Enforce PDE at $x_0 = 0$:~~ $\frac{u_{-1} - 2u_0 + u_1}{h^2} = f_0$

& use BC $u_x(0) = g$ to define the ghost point: centered difference: $\frac{u_1 - u_{-1}}{2h} = g + O$

$\Rightarrow u_{-1} = u_1 - 2hg \Rightarrow f_0 = \frac{u_1 - 2hg - 2u_0 + u_1}{h^2} \Rightarrow \frac{-2u_0 + 2u_1}{h^2} = f_0 + \frac{2g}{h}$

$$\Rightarrow \frac{1}{h^2}\begin{pmatrix} -2 & 2 & & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_0 + \frac{2g}{h} \\ f_1 \\ \vdots \end{pmatrix}$$

$\Leftarrow$ this method is $O(h^2)$ accurate.
but our matrix is not symmetric

can divide first row by 2 to get $\frac{1}{h^2}\begin{pmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & \ddots & & \end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \end{pmatrix} = \begin{pmatrix} \frac{f_0}{2} + g/h \\ f_1 \\ \vdots \end{pmatrix}$

Difference btwn these methods lies in the size of the matrices & misplacement of the boundary

Solvability:

$$\begin{cases} u_{xx} = f, & 0 < x < 1 \\ u_x(0) = \alpha \\ u_x(1) = \beta \end{cases} \Rightarrow \int_0^1 u_{xx}\,dx = \int_0^1 f\,dx$$

$$u_x(1) - u_x(0) = \int_0^1 f\,dx$$

solvability condition $\rightarrow \boxed{\beta - \alpha = \int_0^1 f(x)\,dx}$

Suppose u is a soln to the problem. Then $u + c$ for any constant is also a soln
This doesn't matter physically since this comes from potential problems
(who cares what actual voltage/pressure is, only care about it relative to norm)
This will cause issues in our numerical method, though:

Discretize the problem: $\frac{1}{h^2}\begin{pmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix}\begin{pmatrix} u_0 \\ \vdots \\ \vdots \\ u_{n+1} \end{pmatrix} = \begin{pmatrix} f_0/2 + \alpha/h \\ f_1 \\ \vdots \\ f_n \\ f_{n+1}/2 - \beta/h \end{pmatrix}$

Matrix is singular!

Supp. $\vec{u}$ a soln, then $\vec{u} + c\vec{1}$ is also a soln (all row sums of our matrix are 0)

$\Rightarrow \vec{1}$ spans the nullspace of A. $A\vec{1} = \vec{0}$.

$A\vec{u} = \vec{b}$ will have a soln only for $\vec{b} \in \operatorname{ran} A \Rightarrow \vec{b} \perp \ker A^* \Rightarrow \vec{b} \perp \vec{1}$.

$\Rightarrow A\vec{u} = b - \langle b, \vec{1}\rangle \vec{1}$ has a soln.

$\Rightarrow A\vec{u} = f - \int_0^1 f(x)\,dx = f - \beta + \alpha$

MAT228A - Lecture 8 - 10/18

Finish up 1-D stuff: Poisson eqn w/ Neumann BC's: $\begin{cases} u_{xx} = f \\ u_x(0) = \alpha \\ u_x(1) = \beta \end{cases}$ soln is not unique & condition for solvability $\int_0^1 f(x)dx = \beta - \alpha$

boundary values unknown

2^{nd} order discretization: $\underbrace{\frac{1}{h^2}\begin{pmatrix} -2 & 2 & & \\ 1 & -2 & 1 & \\ & 1 & \ddots & 1 \\ & & 2 & -2 \end{pmatrix}}_{A} \vec{u} = \begin{pmatrix} f_0 + 2\alpha/h \\ f_1 \\ \vdots \\ f_n \\ f_{n+1} - 2\beta/h \end{pmatrix}$

Soln is not unique & matrix is singular.

Note: $A \cdot \vec{1} = \vec{0} \Rightarrow \vec{1} \in \text{null } A$ & w/ more work, $\vec{1}$ spans null A.

How to solve $A\vec{u} = \vec{b}$ w/ singular A:

↳ Need $\vec{b} \in \text{ran } A = (\text{null } A^*)^{\perp}$. Luckily, $\dim(\text{null } A^*) = 1$, so only need to check orthogonality against 1 spanning vec

$A^* = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & \\ 2 & -2 & 1 & \\ & 1 & -2 & \ddots & \\ & & \ddots & \ddots & 2 \\ & & & 1 & -2 \end{pmatrix}$ → nullspace spanned by $\vec{v} = \begin{pmatrix} 1/2 \\ 1 \\ \vdots \\ 1 \\ 1/2 \end{pmatrix}$ so check $\vec{b} \cdot \vec{v} = 0 \Rightarrow \vec{b} \in \text{ran } A$.

$$v^T b = \frac{1}{2} f_0 + \frac{\alpha}{h} + f_1 + \dots + f_n + \frac{1}{2} f_{n+1} - \frac{\beta}{h} = 0$$

$$\Rightarrow \frac{h}{2} f_0 + h \sum_{j=1}^{n} f_j + \frac{h}{2} f_{n+1} = \beta - \alpha$$

This is a 2^{nd} order accurate approximation to $\int_0^1 f(x)dx$ using Trapezoidal rule.

$$\Rightarrow \int_0^1 f(x)dx + O(h^2) = \beta - \alpha.$$

Satisfying the cont's condition doesn't guarantee discrete condition holds!

One way to solve $A\vec{u} = \vec{b}$ is w/ an iteration scheme

E.g. $\vec{u}^{(k+1)} = Tu^{(k)} + \vec{c}$ To get a soln, need $\vec{b} \in \text{ran } A$

Project $\vec{b}$ onto ran A if it's not exactly there (but $O(h^2)$ away!)

$P\vec{b} = \vec{b} - \underbrace{\frac{\vec{v}^T\vec{b}}{\vec{v}^T\vec{v}} \cdot \vec{v}}$ is guaranteed to be in ran A

this is $O(h^2)$, so changing discretization but not accuracy.

Direct Solve: perturbed system: $A\vec{u} = \vec{b} - \lambda\vec{v}$ ← assume don't know λ.

$\Rightarrow A\vec{u} + \lambda\vec{v} = \vec{b}$. For an appropriate choice of λ, have soln. $\vec{u}$.

choose: $\vec{1}^T\vec{u} = 0$ ← want soln of above to have mean zero.

Needed an extra eqn. for λ, so just picked one arbitrarily since solns. are nonunique.

Just pins down a "unique" ...

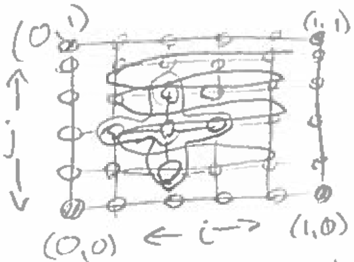
Augmented system:

This system has a unique soln.

$$\begin{pmatrix} A & \vec{1} \\ \vec{1}^T & 0 \end{pmatrix}\begin{pmatrix} \vec{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \vec{b} \\ 0 \end{pmatrix}$$

2D Eqns.

2D Poisson w/ Dirichlet BC's
$$\begin{cases} \Delta u = f & \text{on } (0,1)\times(0,1) \\ u(0,y) = u(1,y) = u(x,0) = u(x,1) = 0 \end{cases}$$



$x_i = ih,\ i=0,\dots,n+1$
$y_j = jh,\ j=0,\dots,n+1$] all points including bdry
$h = \frac{1}{n+1}$

$i=0,n+1 \to$ all j
$j=0,n+1 \to$ all i

Discrete fn. on lattice: $u_{ij} \approx u(x_i, y_j)$ 2D grid fn.

Discretize $\Delta u = u_{xx} + u_{yy} = f$

$$(u_{xx})_{ij} \approx \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h^2}, \quad (u_{yy})_{ij} \approx \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h^2}$$

$$\Rightarrow (\Delta u)_{ij} \approx \underbrace{\frac{u_{i-1,j} + u_{i,j-1} - 4u_{ij} + u_{i+1,j} + u_{i,j+1}}{h^2}}_{\hookrightarrow\ A\vec{u} = \vec{b}} = f_{ij} \quad \text{for } \begin{matrix} i=1,\dots,n \\ j=1,\dots,n \end{matrix}$$

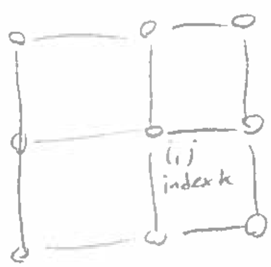
$\Rightarrow n^2$ linear eqns.

Stencil of the difference op.

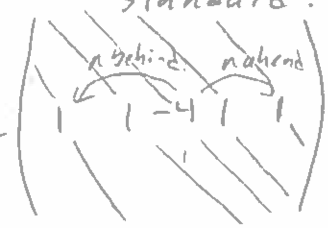
$\hookrightarrow \frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$

need to pack grid fn. into vector
$u_{ij} \to$ single indexed vector
Standard: order by x, then y

$$\vec{u} = \begin{pmatrix} u_{11} \\ u_{21} \\ u_{31} \\ \vdots \\ u_{12} \\ u_{22} \\ \vdots \end{pmatrix} \Rightarrow$$



$\Rightarrow \frac{1}{h^2}$

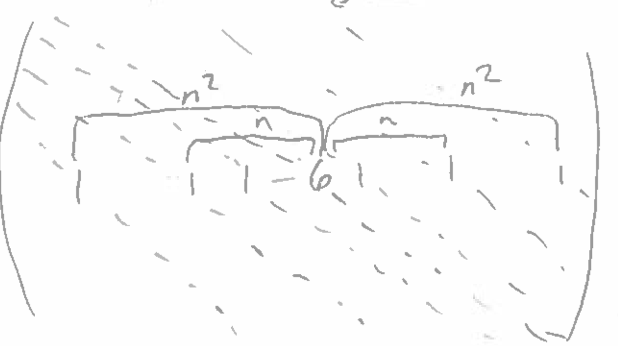


$\begin{pmatrix} \vec{u} \end{pmatrix} = \begin{pmatrix} \vec{b} \end{pmatrix}$

pentadiagonal structure

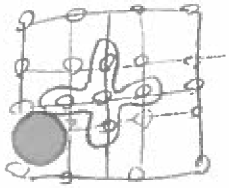
3D

$\frac{1}{h^2}$



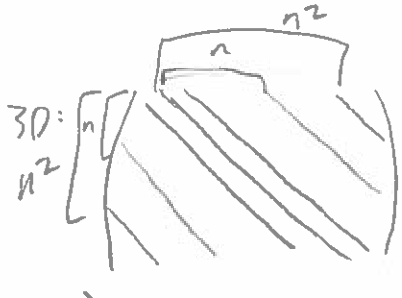
Gaussian elimination on banded matrices doesn't preserve sparcity of matrix.
$\hookrightarrow$ factoring will fill in between bands

MAT 228A – Lecture #9 – 10/20



$A\vec{u} = \vec{b}$

2D: 

3D: 

← Sparsity patterns not good enough for Gaussian elimination
↳ fills in between bands

Work to factor this banded matrix? (LU decomp.)

$\mathcal{O}(abN)$

So in 2D, $N = n^2$, $a = b = n \Rightarrow \mathcal{O}(n^4) = \mathcal{O}(N^2)$

better than full matrix $\to \mathcal{O}(N^3)$

not as good as possible $\to \mathcal{O}(N)$.

backward/forward solve the factored system

work $= \mathcal{O}(Nn) = \mathcal{O}(n^3) = \mathcal{O}(N^{3/2})$.

In 3D, work to factor is $\mathcal{O}(N^3 \cdot n^2 \cdot n^2) = \mathcal{O}(n^7) = \mathcal{O}(N^{7/3})$.

$(N = n^3,\ a = b = n^2)$

Work to back/forward solve factored system is $\mathcal{O}(n^2 N) = \mathcal{O}(n^5) = \mathcal{O}(N^{5/3})$.

<u>Memory usage in 3D?</u> $\mathcal{O}(N)$ nonzero points in original matrix b/c of banded sparse structure

Filled in factored matrix:

(Cholesky factored)

↳ symmetric matrices both factors similar

$n^3 = N$ 

storage scales like $\mathcal{O}(n^5) = \mathcal{O}(N^{5/3})$.

how big for $n = 100$? $100^3 = 10^6 = 1$ million gridpoints $= N$

$\Rightarrow n^5 = 10^{10} = 10 \cdot 10^9 = 10$ billion

In double-precision, 2 bytes/16 bits per number. Have ~10 gigabytes in RAM, need ~20 gigabytes for this!

For systems like this, we know the eigenvalues & eigenvectors, so we can diagonalize

$A\vec{u} = \vec{b}$, Q is the matrix of eigenvectors of A, Λ diag. matrix of eigvalues

In our problem, eigenvectors are orthogonal $\Rightarrow Q^{-1} = Q^T$

$\Rightarrow AQ = Q\Lambda \Rightarrow A = Q\Lambda Q^T \Rightarrow Q\Lambda Q^T\vec{u} = \vec{b}$

$\Rightarrow \Lambda Q^T\vec{u} = Q^T\vec{b}$

$\Rightarrow Q^T\vec{u} = \Lambda^{-1}Q^T\vec{b}$

$\Rightarrow \vec{u} = Q\Lambda^{-1}Q^T\vec{b}$

Storing Q is expensive as a dense matrix

Dealing w/ Λ is trivial

Fast Fourier Transform: Does multiplication by Q^T in $\mathcal{O}(N \ln N)$ time.

Then inverse FFT for Q for another $\mathcal{O}(N\ln N)$ time.

(Poisson) special solver $\cancel{A}$.

In 2D  or $\frac{1}{h^2}\begin{pmatrix} T & I & & \\ I & \ddots & \ddots & \\ & \ddots & \ddots & I \\ & & I & T \end{pmatrix}$ ← n×n identity w/ $T = \begin{pmatrix} -4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -4 \end{pmatrix}$

Exploit this structure to get $\mathcal{O}(n^{3/2})$.

w/ "nested dissection"

Convergence in 2D: $A\underline{e}^h = -\underline{\tau}^h$ w/ $\underline{e}^h$ error & $\underline{\tau}^h$ discretization error.

For the 2-norm, same idea as in 1D.
↳ Know eigenvalues, eigenfunctions: $u_{ij}^{km} = \sin(k\pi x_i)\sin(m\pi y_j)$
↳ $\lambda^{km} = \frac{2}{h^2}[\cos(k\pi h) + \cos(m\pi h) - 2]$

& do similar proof as in 1D.

For the ∞-norm: Show in 2 steps: ① Discrete Maximum Principle:

① $Lu = f$. If $Lu \geq 0$ on some region, then max. value of u is attained on the boundary of this region.
(L: discrete operator)

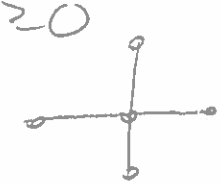
Similarly, if $Lu \leq 0$ on region, min u attained on the boundary.

② If u is a discrete fn. zero on the boundary of the discrete unit square, then $\|u\|_\infty \leq \frac{1}{8}\|Lu\|_\infty$

How do we use this? $Le = -\tau$ => using ②, $\|e\|_\infty \leq \frac{1}{8}\|Le\|_\infty = \frac{1}{8}\|\tau\|_\infty = O(h^2)$.

Hence convergence!

Pf. of 1: Supp. $Lu \geq 0$ => $\frac{1}{h^2}(u_{i-1,j} + u_{i,j-1} - 4u_{ij} + u_{i,j+1} + u_{i+1,j}) \geq 0$



=> $u_{ij} \leq \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i,j+1} + u_{i+1,j})$.

Center point of stencil is $\leq$ average of neighbors.

If u_{ij} is a maximum, then all neighbors must be equal => u constant.
Hence any max. must be on the boundary.

Pf. of 2: $Lu = f$, u is zero on the boundary

Let $w_{ij} = \frac{1}{4}((x_i - \frac{1}{2})^2 + (y_j - 1/2)^2)$. Then $Lw = 1$.

Then $L(u + w\|f\|_\infty) = f + \|f\|_\infty \geq 0$.

So max value of $u + w\|f\|_\infty$ is on the boundary by ① (and $u=0$ on bdry)

=> $u \leq u + w \cdot \|f\|_\infty \leq \max(w)\|f\|_\infty = \frac{1}{8}\|f\|_\infty$

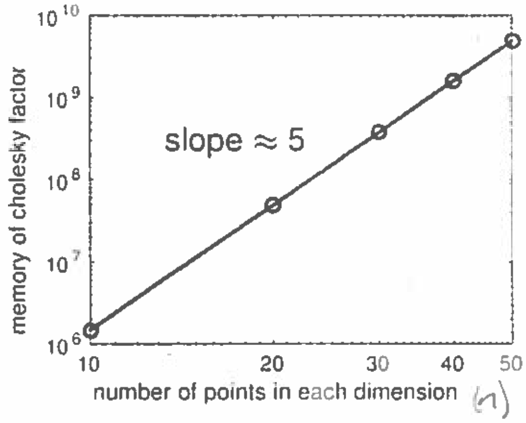
Hence $u \leq \frac{1}{8}\|Lu\|_\infty$ => $\|u\|_\infty \leq \frac{1}{8}\|Lu\|_\infty$

=> $\|e\|_\infty \leq \frac{1}{8}\|Le\|_\infty = \frac{1}{8}\|\tau\|_\infty = O(h^2) \to 0$. $\blacksquare$

Lecture 9 Handout - 10/20

3D w/ gridsize n in each dim.

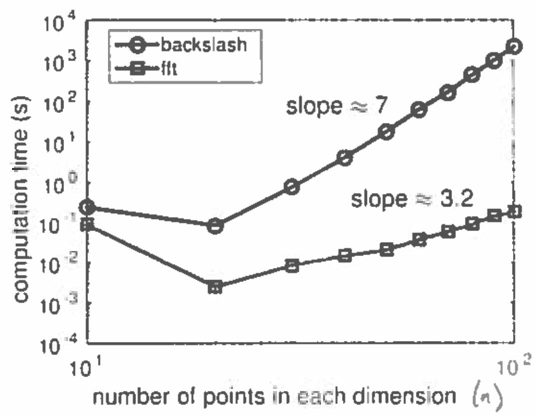
Memory Usage



n	mem (B) ← bytes
10	1.48e+06
20	4.90e+07
30	3.77e+08
40	1.60e+09
50	4.90e+09 ~5 Gb.

Hence memory scales like $O(n^5)$

Solution Time



n	t (GE)	t (FFT)
10	0.2415	0.0893
20	0.0834	0.0025
30	0.7502	0.0085
40	4.0408	0.0145
50	17.3654	0.0202
60	61.7943	0.0365
70	163.9611	0.0569
80	443.7283	0.0891
90	1016.0200	0.1412
100	2185.5417	0.1790

Comp. time grows like $O(n^7)$

FFT is near-optimal, time $O(n^{3.2}) \approx O(N \ln N)$

Lecture 10 10/25

$Au = b$ or $Lu = f \to u = \underbrace{u_{sol}}_{\text{soln. to PDE}} + \mathcal{O}(h^2)$ → can approx. u algebraically within $\mathcal{O}(h^2)$ & remain within $\mathcal{O}(h^2)$ to u_{sol}.

Suppose u^k is an approximate soln to $Lu = f$, where u is the exact soln. of the alg. eqn. (discrete PDE).

Algebraic error: $e^k = u - u^k$.

$Le^k = Lu - Lu^k = f - Lu^k = r^k$ ← residual (defect)

correction eqn: $u = u^k + e^k = u^k + \underbrace{L^{-1}}_{\text{approximate this operator to get iterative scheme}} \underbrace{r^k}_{\text{easy to compute}}$

Let $B \approx L^{-1}$, then example iterative scheme:

$$u^{k+1} = u^k + Br^k$$

If $B \approx L^{-1}$ is good approx, hope u^{k+1} is better approx to u than u^k.

A simple method: Let $B = D^{-1}$ where D is the diagonal part of L.

$\Rightarrow u^{k+1} = u^k + D^{-1}r^k = u^k + D^{-1}(f - Lu^k) = (I - D^{-1}L)u^k + D^{-1}f$

For our problem, $D = -\frac{4}{h^2} I \Rightarrow D^{-1} = -\frac{h^2}{4} I$ (our problem is Dirichlet Laplacian on rectangle)

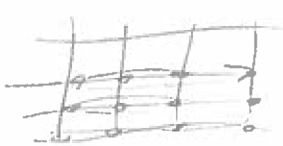
$$\Rightarrow u^{k+1} = \left(I + \frac{h^2}{4}L\right)u^k - \frac{h^2}{4} f$$

At a point: $u_{ij}^{k+1} = u_{ij}^k + \frac{h^2}{4}\left(\frac{u_{i-1,j}^k + u_{i,j-1}^k - 4u_{ij}^k + u_{i+1,j}^k + u_{i,j+1}^k}{h^2}\right) - \frac{h^2}{4} f_{ij}$

$$u_{ij}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^k + u_{i,j-1}^k + u_{i+1,j}^k + u_{i,j+1}^k\right) - \frac{h^2}{4} f_{ij}$$

This is the Jacobi iterative scheme, and for our problem it converges (very slowly).

As we do this, we go through the grid & compute better values u_{ij} & then update at the end all at once — why not take advantage of better values "behind you" as we go?



loop some # of times
  loop i
    loop j $v_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}) - \frac{h^2}{4} f_{ij}$

} can parallelize

Gauss-Seidel (lexicographic order)

loop some max #
  loop i
    loop j
      $u_{ij} = \frac{1}{4}(u_{i,j-1} + u_{i-1,j} + u_{i+1,j} + u_{i,j+1}) - \frac{h^2}{4} f_{ij}$
    end
  end
end

converges faster than Jacobi, but still slow

but not parallelizable

Think of both methods as splitting methods:

$Au = b \to A = M - N$ where M is easy to invert

$\Rightarrow Mu - Nu = b \Rightarrow u = M^{-1}Nu + M^{-1}b$

Iterative scheme: $u^{k+1} = M^{-1}Nu^k + M^{-1}b$

For our problem,

$A = D - L - U$ where D is diagonal, $-L$ is lower triangular part, $-U$ is upper triangular part

Jacobi: $M = D$, $N = L + U \Rightarrow u^{k+1} = D^{-1}(L+U)u^k + D^{-1}b$

Gauss-Seidel: $M = D - L$, $N = U \Rightarrow u^{k+1} = (D-L)^{-1}Uu^k + (D-L)^{-1}b$

Both methods are of the form: $u^{k+1} = Tu^k + c$ ← a fixed pt. iteration

fixed point soln: $u = Tu + c$

When does this fixed pt. iteration converge?

$e^k = u - u^k$,

$$
\begin{aligned}
-(u^{k+1} &= Tu^k + c) \\
+)\ u &= Tu + c \\
\hline
e^{k+1} = u - u^{k+1} &= T(u - u^k)
\end{aligned}
$$

$\Rightarrow e^k = T^k e^0$

when does $e^k \to 0$? only for $\rho(T) < 1$. (spectral radius (i.e. magnitude of largest λ))

Suppose we can diagonalize T: $TX = X\Lambda$ ← diagonal λ matrix

$T = X\Lambda X^{-1}$

$\Rightarrow T^2 = (X\Lambda X^{-1})(X\Lambda X^{-1}) = X\Lambda^2 X^{-1} \Rightarrow T^k = X\Lambda^k X^{-1}$

$\Lambda^k = \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_n^k \end{pmatrix} \to 0$ if $|\lambda_\ell| < 1 \ \forall \ell$ $\Rightarrow T^k e^0 \to 0$ for any e^0 if $|\lambda_\ell| < 1 \ \forall \ell$.

MAT228A – Lecture 11   11/1

Iterative Methods for the Poisson Eqn:

$\underline{u}^{k+1} = T\underline{u}^k + \underline{c}$ When do we stop the iteration?

Two standard ways to stop iteration:

1) Based on the size of the residual $r^k = f - Au^k$
2) Based on the size of difference between iterates $u^{k+1} - u^k$

} How do these control error?

Use the residual absolute tolerance: $\|r^k\| < tol = \varepsilon$ ← typically on same order as numerical scheme expected accuracy.

relative tolerance: $\|r^k\| < (tol)\cdot\|f\|$

If $u_0 = 0$, $r^0 = f$, so relative tol. measures reduction of residual from this initial point

Bound error based on residual bound: $Ae^k = r^k$

Want to control e^k: $\|e^k\| = \|A^{-1}r^k\| \le \|A^{-1}\|\,\|r^k\| \le \|A^{-1}\|\varepsilon$

Calculating residuals takes extra work, but already have successive iterates!

Use $u^{k+1} - u^k$: absolute tolerance $\|u^{k+1} - u^k\| < tol$

relative tolerance $\|u^{k+1} - u^k\| < (tol)\,\|u^k\|$

Jacobi, GS, SOR, multigrid all of the form $u^{k+1} = u^k + Br^k$, $B \approx A^{-1}$
↑ depends on grid spacing

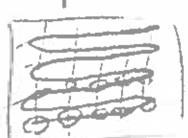
$\Rightarrow u^{k+1} - u^k = Br^k$.

Note $\|e^k\| = \|A^{-1}r^k\| = \|A^{-1}B^{-1}(u^{k+1} - u^k)\| \le \|A^{-1}B^{-1}\|\,\|u^{k+1} - u^k\|$.

Jacobi: $B = -\frac{h^2}{4}I \Rightarrow \|B^{-1}\| = \frac{4}{h^2} \Rightarrow \|e^k\| \le \frac{4}{h^2}\|A^{-1}\|\,\|u^{k+1} - u^k\|$

Jacobi does <u>not</u> depend on the <u>ordering</u> of the unknowns

GS <u>does</u> depend on the ordering of the unknowns

GS-lex: 

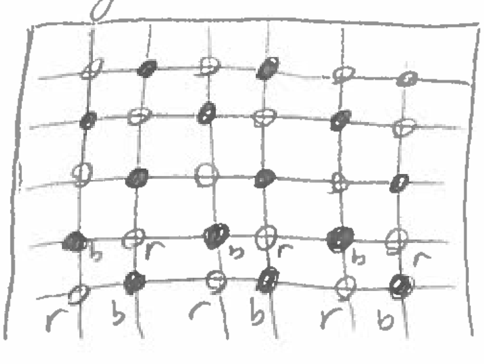
update one point after another

loop k
loop i,j

$u_{i,j} = \frac{1}{4}(\underbrace{u_{i-1,j} + u_{i,j-1}}_{new} + \underbrace{u_{i+1,j} + u_{i,j+1}}_{old} - h^2 f_{ij})$ ← hard to // -ize

Another way to order:

update all red, then update all black



Take colored points
$i+j = \begin{cases} \text{even} - \text{red} \\ \text{odd} - \text{black} \end{cases}$

Order reds, then order blacks & then apply GS-RB
update all red, then all,

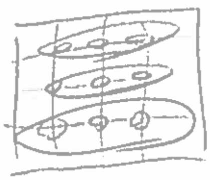
GS-RB:

loop k
  loop red
    $u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - h^2 f_{i,j})$ ← note these are all black points
  loop black
    $u_{ij} =$ "

GS-RB makes sense for our Poisson solver since a red point is updated via all black points & vice-versa.

↳ Easier to ∥-ize than GS lex.

Other variations → block or line relaxation methods



← could update each row at same time (like 1D Poisson eqn.)
↳ tri-diagonal solve on each row of pts.

Not especially better except very useful for problems such as $u_{xx} + \varepsilon u_{yy} = f$

weak coupling in y direction, so just do 1D solve on the x direction & loop over y-rows. "anisotropy"

could also do blocks

SOR (successive over-relaxation)

GS: $u_{ij} = \frac{1}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - h^2 f_{ij})$

SOR: $u_{ij} = \frac{\omega}{4}(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1} - h^2 f_{ij}) + (1-\omega) u_{ij}$

Linear combination of GS update & current iterate.
$\omega < 1$ under-relaxation, $\omega > 1$ over-relaxation

*SOR requires that $0 < \omega < 2$ for convergence. *

Recall: All these methods are "matrix-splitting" $A = M - N$
$M u^{k+1} = N u^k + f$
$u^{k+1} = M^{-1} N u^k + M^{-1} f$

SOR: $M = \frac{1}{\omega} D - L$ (where $A = D - L - U$)
$N = \frac{1-\omega}{\omega} D + U$

$$T_{SOR} = M^{-1} N = \left(\tfrac{1}{\omega} D - L\right)^{-1}\left(\tfrac{1-\omega}{\omega} D + U\right)$$
$$= \omega (D - \omega L)^{-1}\left(\tfrac{1-\omega}{\omega} D + U\right) = (D - \omega L)^{-1}((1-\omega) D + \omega U)$$

$$\det(T_{SOR}) = \det(\underbrace{(D - \omega L)}_{\text{triangular}}{}^{-1}) \cdot \det(\underbrace{(1-\omega) D + \omega U}_{\text{triangular}}) = \frac{\det((1-\omega)D)}{\det(D)} = (1-\omega)^{N}$$

of grid points

Require $|\det(T_{SOR})| < 1 \Rightarrow |1-\omega| < 1 \Rightarrow 0 < \omega < 2$.

□

Convergence analysis for SOR:

(use same trick as in GS convergence analysis to compute eigenvalues of the update matrix in terms of Jacobi update)

Let μ be an eigenvalue of Jacobi update T_J.

Then $\mu = \dfrac{\lambda + \omega - 1}{\omega \lambda^{1/2}}$ for λ eigenvalue of T_{SOR}

Rearranged: $\lambda - \omega\mu\lambda^{1/2} + (\omega - 1) = 0$

$$\Rightarrow 2\lambda^{1/2} = \omega\mu \pm \sqrt{\omega^2\mu^2 - 4(\omega-1)}$$

As $\omega \to 0$, real eigenvalues $|\lambda^{1/2}| \to 1$

As $\omega \to 2$, complex eigenvalues

if λ complex, know $|\lambda^{1/2}| = |\omega - 1|$, hence $|\lambda^{1/2}| \to 1$.

Want to minimize $|\lambda^{1/2}|$ so decrease ω from 2 $\to$ better convergence.

If λ real, to minimize $|\lambda^{1/2}|$, $\dfrac{\partial \lambda^{1/2}}{\partial \omega} < 0$ so increase ω from 0 for better convergence.

$\Rightarrow$ Pick on boundary of real & complex λ:

Optimal ω^* satisfies $\omega^{*2}\mu^2 = 4(\omega^* - 1)$

Work $\Rightarrow$ $\omega^* = 2 / 1 + (1 - \rho_J^2)^{1/2}$, where $\rho_J = \rho(T_J)$ spectral radius of Jacobi matrix

$\Rightarrow \rho_{SOR} = \omega^* - 1$.

Recall: $\rho_J = \cos(\pi h) \Rightarrow \omega^* = 2 / 1 + (1 - \cos^2 \pi h)^{1/2} = \dfrac{2}{1 + \sin \pi h}$

As $h \to 0$, $\omega^* = \dfrac{2}{1 + \sin \pi h} \approx \dfrac{2}{1 + \pi h} \approx 2(1 - \pi h) + O(h^2)$

$\Rightarrow$ As $h \to 0$, $\rho_{SOR} \approx 1 - 2\pi h + O(h^2)$.

iterations to reduce error by 10^{-1}:

N×N	GS	SOR
32×32	254	12
64×64	985	24
128×128	3882	47
256×256	15404	94

SOR has better iteration count and scaling!!

MAT228A - Lecture #12 11/3

Last time SOR: $w^* = \frac{2}{1+(1-\rho_J^2)^{1/2}}$ optimal w & $\rho_{SOR} = w^* - 1$

Asymptotically as $h \to 0$, $w^* = \frac{2}{1+\sin\pi h} = 2(1-\pi h) + O(h^2) \Rightarrow \rho_{SOR} = 1 - 2\pi h + O(h^2)$

compared to GS on 256^2 mesh, # iterations per digit of accuracy: GS → 15409, SOR → 94

Scaling on the work of SOR to converge to a tol $= Ch^2$.

$$\rho^k = Ch^2 \Rightarrow k = \frac{\ln Ch^2}{\ln \rho} . \quad \ln \rho \approx \ln(1-2\pi h) \approx -2\pi h$$

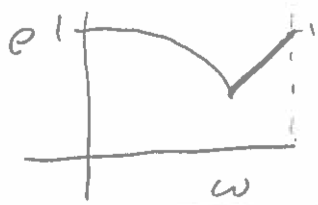
$$\approx \frac{\ln C + \ln h^2}{-2\pi h} \Rightarrow k = O(h^{-1}\ln h)$$

So in 2D, $h \sim \frac{1}{n}$, n = # of pts. in each direction, $N = n^2$ = total # of points

$\Rightarrow k = O(\sqrt{N} \ln N)$ iteration count

Work per iteration is $O(N) \Rightarrow$ work to solve is $O(N^{3/2}\ln N)$

Drawback: Need to know ω^*. Generically, expect $\omega^* = \frac{2}{1+Ch}$



every sensitive

applies for large k, what happens for small k?

All methods so far, $\frac{\|e^{k+1}\|}{\|e^k\|} \approx \rho \to 1$ as $h \to 0$

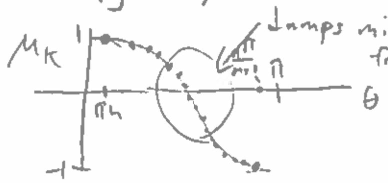
Can we find a method with $\rho < C < 1$ as $h \to 0$?

Then we can get iteration count to fixed tolerance indep. of mesh!

Analyze Jacobi in 1D: $T_J = I + \frac{h^2}{2}A$

eigvals of A are $\lambda_k = \frac{2}{h^2}(\cos(k\pi h) - 1)$, eigvalues of T_J are $\mu_k = 1 + \frac{h^2}{2}\lambda_k = \cos k\pi h$

$k = 1, \dots, n$, $h = \frac{1}{n+1} \Rightarrow k\pi h = \pi h, \dots, \frac{n\pi}{n+1}$



high k & low k have same order magnitude λ_k.

high & low spatial frequencies are damped the least

Jacobi iterations result in low freq. average w/ high freq. oscillations on top.

Can we try to improve convergence/smoothing [illegible] for Jacobi?

$\hookrightarrow$ ω-Jacobi method: $u_j^{k+1} = \frac{\omega}{2}\left(u_{j-1}^k + u_{j+1}^k - h^2 f_j\right) + (1-\omega)\, u_j^k$

update matrix $T_{\omega J} = \omega T_J + (1-\omega) I = I + \omega \frac{h^2}{2} A = \omega\left(I + \frac{h^2}{2} A\right) + (1-\omega) I$

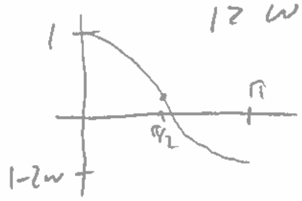
eigvalues $\mu_k^\omega = \omega \cos(k\pi h) + (1-\omega)$

---

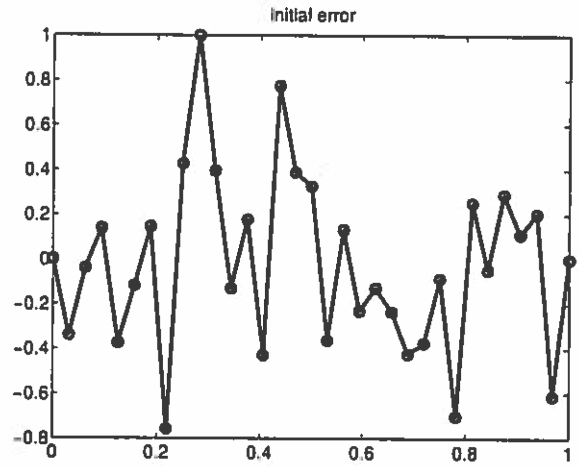
for k small, $\mu_k^\omega = \omega\left(1 - \frac{k^2\pi^2 h^2}{2} + \dots\right) + (1-\omega) = 1 - \frac{\omega k^2 \pi^2 h^2}{2}$ $\leftarrow$ not much less than 1

for k big, $\mu_k^\omega = 1 - 2\omega + \dots$ $\leftarrow$ much less than 1, will damp high frequencies

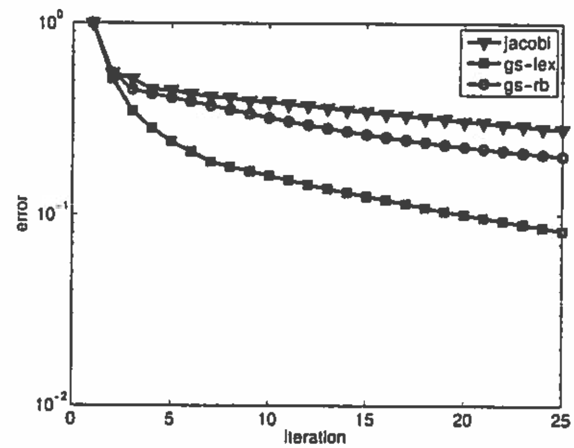
if $\omega \geq 1/2$

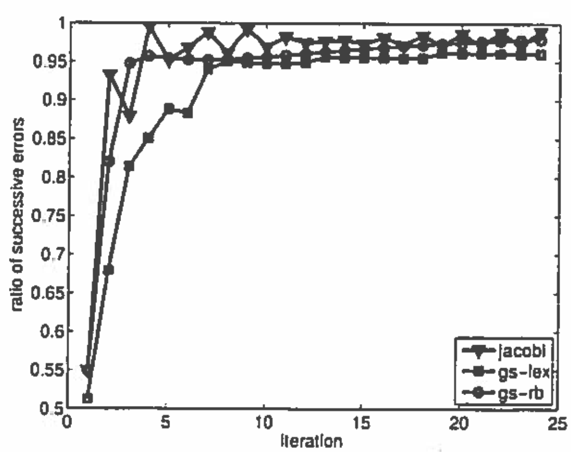


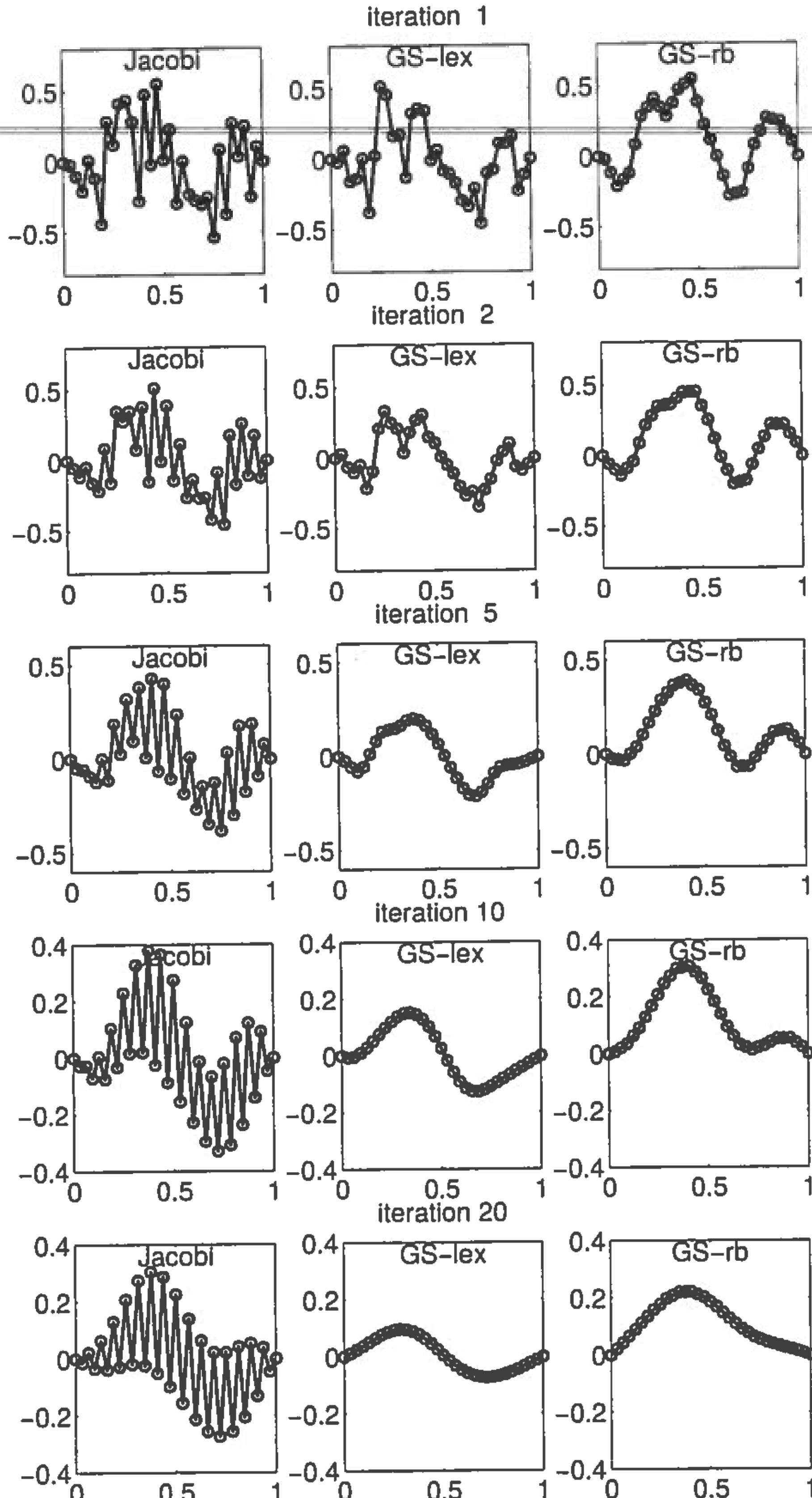
Lecture 12 Handout 11/3



← "equal contribution" of all frequencies







low frequencies dominate
average dynamics

high frequencies make grid scale oscillations

MAT228A – Lecture 13 – 11/8

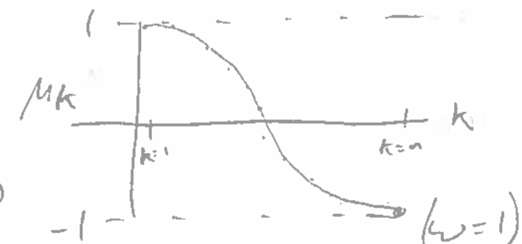
Multi~~pass~~Grid: Couple simple iterative methods w/ more complicated

ω-Jacobi in 1D: $u_j^{k+1} = \frac{\omega}{2}\left(u_{j-1}^k + u_{j+1}^k - \underset{RHS}{?}\right) + (1-\omega)u_j^k$

update matrix: $\omega T_J + (1-\omega)I$

eigenvectors: $u_{j,k} = \sin(k\pi x_j)$

eigvalues $\mu_k = \omega\cos(k\pi h) + (1-\omega)$



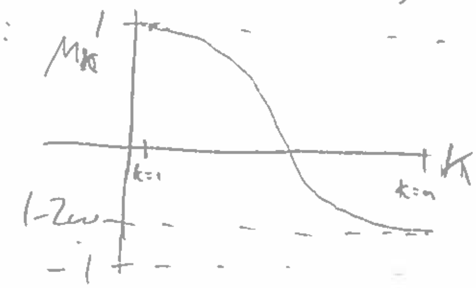
When $\omega = 1$, superposition of low & high frequencies, rapid damping of mid freqs.

for $\omega \neq 1$, shift & scale this graph:

$\mu_1 = \omega\left(1 - \frac{\pi^2 h^2}{2} + \dots\right) + (1-\omega)$

$= 1 - \omega\frac{\pi^2 h^2}{2}$ ← looks like should make $\omega > 1$ to get better convergence

$\mu_n = 1 - 2\omega + \dots$ ← should make $0 < \omega < 1$ or **no** convergence

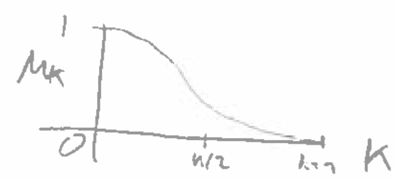


↑ for $\frac{1}{2} < \omega < 1$

↳ So only low freq. are damped slowly, higher freqs. damp rapidly but convergence for the low freqs. is now slowed for $\omega \neq 1$.

Using GS/SOR yields smooth errors → can represent on a coarse grid!
Smooth using GS/SOR iterative methods, then use coarse-correction!

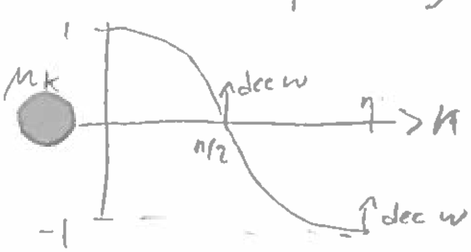
Pick ω to eliminate highest frequencies:
$\omega = 1/2$ most effective on highest freqs.



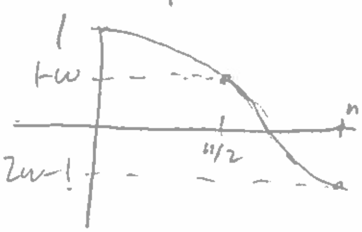
($\omega = 1/2$)

Want to reduce frequencies w/ wavenumbers $k \geq n/2$
(since coarsing mesh by throwing out every other point makes representing freqs. of $k \geq n/2$ impossible!)

reduces these by a factor of 3 every iteration!



optimizing problem
n/2 gets worse, n gets better as w



} same in abs. value = min. of both

optimal ω^* : $1-\omega^* = 2\omega^* - 1 \Rightarrow \omega^* = 2/3$

More formally. $\mu = \max_{\frac{n}{2} \le k \le n} |\mu_k|$ is the smoothing factor, i.e. the largest factor by which high freqs ($k \ge n/2$) are reduced w/ one application of the smoother ω-Jac. w/ ω = 2/3

Smoothing factors

ω-Jacobi:	ω^*	μ	GS lex μ	GSRB μ	ω-GSRB
1D	2/3	1/3	0.45	0.125	
2D	4/5	3/5	0.5	0.25	
3D	6/7	5/7	0.567	0.445	0.23
				winner	$\omega \approx 1$

GS has nonlinear eigenfns, not important how these are eliminated
↳ How does GS eliminate high freq. sine waves?

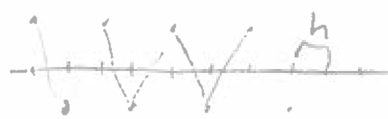
GS lex analysis of smoothing: Local Fourier analysis:

ignore boundaries & analyze the problem on the infinite domain ($x_j = jh$, $j \in \mathbb{Z}$)

Now the eigenfns are $u^k_\ell = e^{ikx_\ell} = e^{ikh\ell}$ ← Fourier modes

What values does k take?

Highest wavenumber k corresponds to



period 2h

$\Rightarrow k(2h) = 2\pi \Rightarrow k = \pi/h$

$\Rightarrow$ All k must be in $-\pi/h \le k \le \pi/h \Rightarrow -\pi \le kh \le \pi$
$-\pi \le \Theta \le \pi$

Θ is "continuous" wave number

eigenfns: $u_\ell(\Theta) = \exp(i\Theta\ell)$ high freq. $|\Theta| \ge \pi/2$

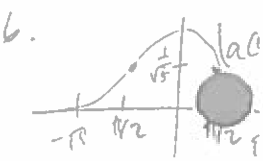
GS-lex:

$\underbrace{e^{k+1}}_{\text{error at } k+1^{\text{st}} \text{ iterate}} = Te^k \xrightarrow{\text{on infinite lattice}} e^{k+1}_\ell = \frac{1}{2}\left(e^{k+1}_{\ell-1} + e^k_{\ell+1}\right)$ Assume $e^k_\ell = \exp(i\Theta\ell)$

error in one eigenspace, fix Θ

$\Rightarrow e^{k+1}_\ell = Te^k_\ell = a e^k_\ell$ where a is "amplification factor"

$\Rightarrow a e^k_\ell = \frac{1}{2}\left(a e^k_{\ell-1} + e^k_{\ell+1}\right)$ ← $\begin{cases} e^k_{\ell-1} = e_\ell \exp(-i\Theta) \\ e^k_{\ell+1} = e_\ell \exp(i\Theta) \end{cases}$ use sub.



$\Rightarrow a = \frac{1}{2}\left(a\exp(-i\Theta) + \exp(i\Theta)\right)$

$\Rightarrow a = \frac{\exp(i\Theta)}{2 - \exp(-i\Theta)}$ $\xrightarrow{\text{smoothing factor}}$ $\mu = \max_{|\Theta| \ge \pi/2} |a(\Theta)| = \frac{1}{\sqrt{5}} \approx 0.45$

MAT 228A - Lecture 14 - 11/10

In 3D: GE $\sim N^{2/3}$ scaling, FFT $\sim N^{3.2/3}$ scaling, SOR $\sim N^{4.1/3}$ MATLAB
$\sim N^{4.4/3}$ Fortran
$\approx N^{1.5}$

- Vectorize code for speedup!

Recall: Big Idea of Multigrid is to use smoothers (Jacobi, GS) to get rid of high spatial freq. error & then use a coarser mesh to eliminate low freq. errors

Coarse Grid Correction:

Let u_h be the algebraic soln to $L_h u_h = f_h$

u_h^k is the approximate soln after k-iterations (i.e. through Jacobi or GS)

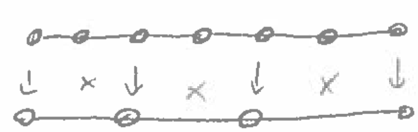
$e_h^k = u_h - u_h^k$ algebraic error $\quad r_h^k = f_h - L_h u^k$ residual

$$\hookrightarrow u_h = u_h^k + e_h^k = u_h^k + \underbrace{L_h^{-1}}_{\text{approximate } L_h^{-1} \text{ to gen. iterative scheme}} r_h^k$$

- a coarse mesh to "solve" $L_h e_h^k = r_h^k$ for e_h^k (since we wiped out high freq. error!)

Notation:

Let Ω_h = mesh w/spacing h (original fine grid)



Then Ω_{2h} is a coarser grid $\to$ half as many pts. in 1D

$G(\Omega_h)$ set of grid functions on Ω_h (i.e. $f, u, L, \dots$)

Transfer operators

Restriction operator: $I_h^{2h} : G(\Omega_h) \to G(\Omega_{2h})$

Interpolation/prolongation: $I_{2h}^{h} : G(\Omega_{2h}) \to G(\Omega_h)$

Have u_h^k. Compute fine grid residual $r_h^k = f_h - L_h u_h^k$

- Restrict residual $r_{2h}^k = I_h^{2h} r_h^k$

Solve for error $e_{2h}^k = L_{2h}^{-1} r_{2h}^k$ ← doesn't matter how we solve since way cheaper

Interpolate to fine grid: $\tilde{e}_h^k = I_{2h}^h e_{2h}^k$

Correct the approx soln:

$$u_h^{k+1} = u_h^k + \tilde{e}_h^k$$

~~Iterative Scheme:~~

$$u_h^{k+1} = u_h^k + \tilde{e}_h^k$$
$$= u_h^k + I_{2h}^h e_{2h}^k$$
$$= u_h^k + I_{2h}^h L_{2h}^{-1} r_{2h}^k$$
$$= u_h^k + I_{2h}^h L_{2h}^{-1} I_h^{2h} r_h^k$$
$$= u_h^k + I_{2h}^h L_{2h}^{-1} I_h^{2h} (f_h - L_h u_h^k)$$
$$= (I - \underbrace{I_{2h}^h L_{2h}^{-1} I_h^{2h}}_{} L_h) u_h^k + I_{2h}^h L_{2h}^{-1} I_h^{2h} f_h$$

hope this is close to L_h^{-1} => product close to I

Still a f.p. scheme!

$u_h^{k+1} = K u_h^k + c$

will converge quickly when $\rho(K) \approx 0$.

Perform smoothing to converge!

Let S = smoothing operator

Two-Grid Iteration:

1. Presmoothing => apply S ν_1 times
2. Apply Coarse Grid correction
   - → compute residual
   - → restrict residual
   - → solve for coarse grid error ✱ add more grids here for multigrid
   - → interpolate error
   - → correct (add error back in)
3. Postsmooth => apply S ν_2 times

↳ multigrid iteration Matrix $M = S^{\nu_2}(I - I_{2h}^h L_{2h}^{-1} I_h^{2h} L_h) S^{\nu_1}$

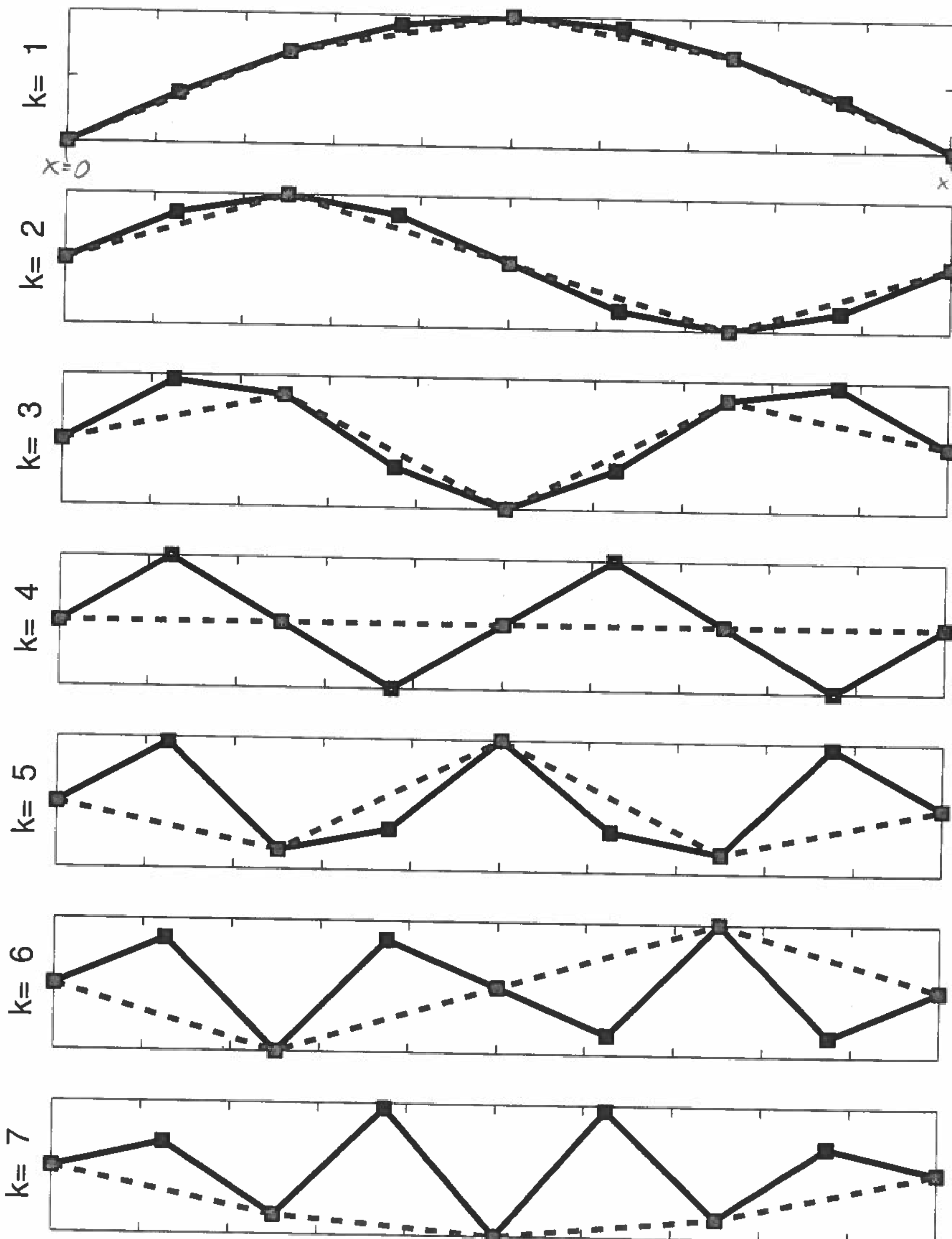
Questions:
1. How to pick ν_1 & ν_2?
2. What are the transfer operators
3. How to solve for L_{2h}^{-1}? What is this coarse grid operator?
4. Which smoothing operator?
5. How efficient is this? => $O(N \ln N)$ work

What happens to sine waves as we coarsen mesh?

Lecture 17 Handy 11/10

— $\Rightarrow h = 1/$

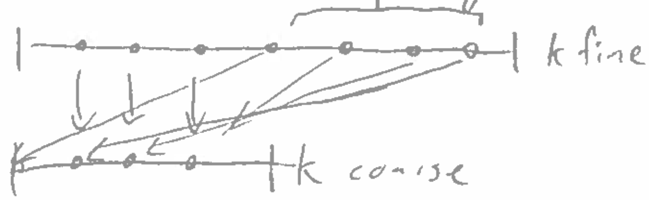
- - - $\Rightarrow h = \frac{1}{4}$



these look like k=3,2,1 on coarse mesh

creates extra low freq. error upon coarsening

use smoothing to erase these first!



(Aliasing error)

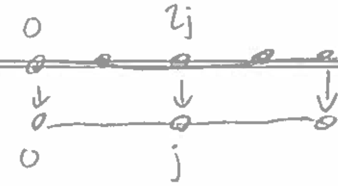
coarse grid correction alone is terrible!

Questions Answered   Lecture 14 - 11/10 cont'd

5. Multigrid will be a $\mathcal{O}(N \ln N)$ algorithm

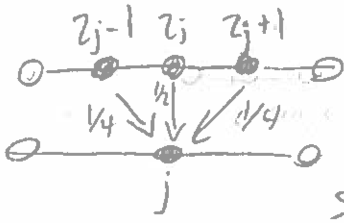
2. Transfer operators:

Restriction



Injection $(u_{2h})_j = (u_h)_{2j}$ } turns out to be not very good

Full-weighting



$$(u_{2h})_j = \frac{1}{4}(u_h)_{2j-1} + \frac{1}{2}(u_h)_{2j} + \frac{1}{4}(u_h)_{2j+1}$$

* wipes out high freqs. very well

* Adjoint of the linear interpolation op.

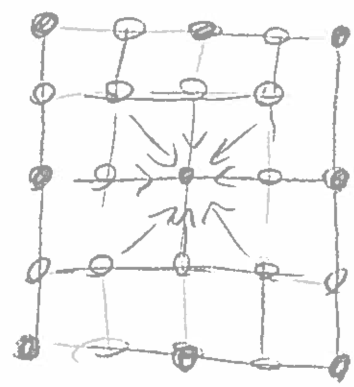
Stencil for Full-Weighting:

1D: $$I_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

2D: $$I_h^{2h} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$= \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

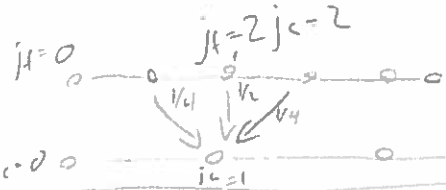
$$= \left(I_h^{2h}\right)^T I_h^{2h}$$ ← outer/tensor product



MAT228A - Lecture 16 - 11/15

Transfer operators

- Restriction Operator: I_h^{2h}



full-weighting

1D: $I_h^{2h} = \frac{1}{4}[1\ 2\ 1]$

2D: $I_h^{2h} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

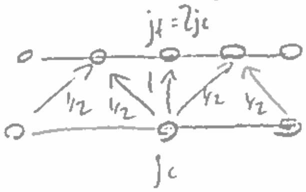
pseudocode for 1D:

loop over coarse mesh jc
  compute jf
    $u_c(j_c) = \frac{1}{4}\left(u_f(j_f-1) + 2u_f(j_f) + u_f(j_f-1)\right)$
(to vectorize, add 3 shifted arrays together)

Half-weighting

$$I_h^{2h} = \frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ ← less information, so need better smoothing. Not as good as full-weighting

Interpolation/Prolongation Operator:



$(u_h)_{2j_c} = (u_{2h})_{j_c}$

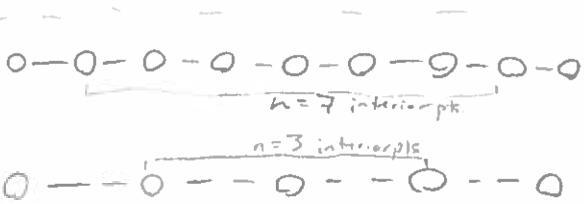
$(u_h)_{2j_c-1} = \frac{1}{2}\left((u_{2h})_{j_c-1} + (u_{2h})_{j_c}\right)$

Bilinear interp.

pseudocode for 1D: $u_f = \vec{0}$
- loop over coarse mesh jc
  $u_f(2j_c) = u_c(j_c)$
  $u_f(2j_c-1) \mathrel{+}= \frac{1}{2}u_c(j_c)$
  $u_f(2j_c+1) \mathrel{+}= \frac{1}{2}u_c(j_c)$

In 1D: $I_{2h}^{h} = \frac{1}{2}\,]1\ 2\ 1[$

In 2D: $I_{2h}^{h} = \frac{1}{4}\,\left]\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}\right[$



Matrix for Full-weighting restriction: I_h^{2h} is a 3×7 matrix

$$I_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \end{bmatrix}$$

Matrix for Interpolation: I_{2h}^{h} is 7×3 matrix

$$I_{2h}^{h} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

Clearly, $I_h^{2h} = \frac{1}{2}\left(I_{2h}^{h}\right)^T$

With the inner product $\langle u, v\rangle_h = h u^T v$,

$$\langle u_h, I_{2h}^h v_{2h}\rangle_h = h u_h^T I_{2h}^h v_{2h}$$
$$= h\left((I_{2h}^h)^T u_h\right)^T v_{2h} = h\left(2 I_h^{2h} u_h\right)^T v_{2h}$$
$$= \langle I_h^{2h} u_h, v_{2h} \rangle$$

? Full weighting is the adjoint of interpolation ☆

Multigrid/2-grid Overview

$\circ \xrightarrow{\text{smooth}} \circ \xrightarrow{\text{residual}} \circ \xrightarrow{\text{restrict}} \circ \xrightarrow{\text{solve}} \circ \xrightarrow{\text{interpolate}} \circ \xrightarrow{\text{correct}} \circ \xrightarrow{\text{smooth}}$

$L_h u_h = f_h$

$\underbrace{\quad}_{\text{solve}}$

? — $L_{2h} e_{2h} = r_{2h}$

One way to define L_{2h}: Galerkin coarse grid op. $L_{2h} = I_h^{2h} L_h I_{2h}^h$

In 2D, stencil $L_h = \frac{1}{h^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix} \Rightarrow L_{2h} = \frac{1}{(2h)^2}\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}$, $L_{4h} =$ ~~smoothing~~ else ...

This came from a PDE though! So just rediscretize the PDE!

Simpler approach: $L_{2h} = \frac{1}{(2h)^2}\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$ ← rediscritization

(Galerkin used in Algebraic multigrid → more general, but slower!)

How well does this work?

Let ν_1 = # of presmooth steps, ν_2 = # of postsmooth steps. $\nu = \nu_1 + \nu_2$

← Away from start, ν only # that matters in 2grid.

Spectral radii for 2-grid, 2D, fullweighting, bilinear interpol., rediscretization

ν	ω-Jac	GS-RB
1	0.6	0.25
2	0.360	0.074
3	0.216	0.053
4	0.137	0.041

☆ these spectral radii are independent of meshspacing!

(before: $\rho = f(h)$ & $\rho \to 1$ as $h \to 0$)

☆ $\rho < c < 1$.

Recall: To det. iteration counts to reduce error by ε: $\rho^k = \varepsilon$

$\Rightarrow k = \ln \varepsilon / \ln \rho$

Q: How much smoothing should we do?

$k \propto -1/\log_{10}(\rho)$

A: Balance smoothing work w/ iteration work

$\text{Work} = \underbrace{\underset{\text{smoothing}}{\nu}}_{O(N)} + \underset{\text{everything else}}{\omega}$. Work per digit of accuracy: $\frac{(\nu+\omega)}{-\log_{10}(\rho)} = (\text{work})(\#\text{its. per digit})$

total work

ν	$\omega=0$	$\omega=1$	$\omega=5$ (comparable work to 5 smooths)	$\omega=6$
1	1.66	3.32	9.97	11.63
2	1.77	2.65	6.19	7.07
3	2.35	3.14	6.27	7.05
4	2.88	3.60	6.49	7.21

$\omega=1$ is comparable work to 1 smooth

total work $\propto \left(-\frac{(\nu+\omega)}{\log\ldots}\right)$

Suggests $\nu=2$ gives lowest work!

MATH 269A - Lecture 17 - 11/17

Last time, we saw that $\boxed{\nu = 2 \text{ smoothing steps}}$ gives the lowest work for GS-RB, full-weighting, bilinear interpolation, rediscretization 2-grid approach

How to pick ν_1 & ν_2 (The pre-smooth step # & post-smooth step #)?

Let M be the multigrid iteration operator. Then error at iteration k is

$$e^k = M e^{k-1} \Rightarrow \|e^k\|_2 \le \|M\|_2 \|e^{k-1}\|_2 \Rightarrow \frac{\|e^k\|_2}{\|e^{k-1}\|_2} \le \|M\|_2$$

* for small k, $\|M\|_2$ is a better error approx. than ρ^k.

ν_1	ν_2	$\|M\|_2$	
1	0	0.559	] even tho same amt. of work!
0	1	1.414	
2	0	0.200	] better to mix!
1	1	0.141	
0	2	1.414	← bad!

=>

ν_1	ν_2	$\|M\|_2$	
3	0	0.137	
2	1	0.081	☆
1	2	0.081	
0	3	1.414	

In general, common choices: $\underbrace{\nu_1 = 1, \nu_2 = 1}_{\text{since } \nu=2 \text{ optimal}}$ (or $\nu_1 = 2$, $\nu_2 = 1$ if $\nu = 3$ optimal)

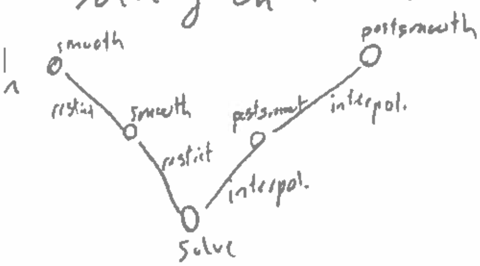
Moving from 2-grid to Multigrid:

Why use direct solve on the coarse grid?

Use another 2grid as the solver!

Multigrid iteration:

Solving $L_h u_h = f_h$



smooth ν_1 times & compute r_h

↳ restrict to r_{2h}, solve $L_{2h} e_{2h} = r_{2h}$

↳ smooth $L_{2h} u_{2h} = f_{2h}$ ν_1 times (initial guess $e_{2h}^0 = 0$)

↖ actually error & residual

↳ compute residual to error equation "r_{2h}"

↳ restrict to r_{4h} → solve $L_{2h} e_{4h} = r_{4h}$

rename & solve $L_{4h} u_{4h} = f_{4h}$ (for u_{4h})

interpolate to correct $u_{2h} = u_{2h} + I_{4h}^{2h} u_{4h}$

smooth ν_2 times

interpolate & correct $u_h = u_h + I_{2h}^{h} u_{2h}$

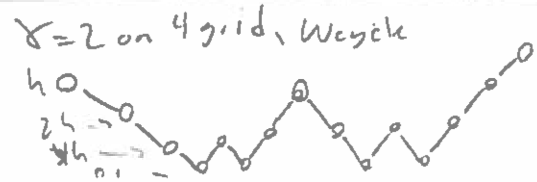
smooth ν_2 times

This is one iteration of a 3-grid algorithm using a "V-cycle"

→ other forms of cycles:

Do γ iterations before returning to fine grid. → e.g., $\gamma = 2$ on 3-grid

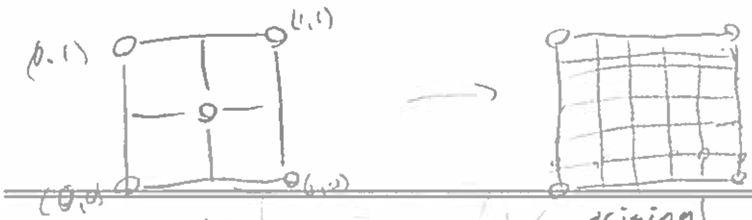
$\gamma = 2$ on 4 grid, W-cycle



V-cycle ⇒ $\gamma = 1$.

In practice, $\nu=1$ V-cycles & $\nu=2$ W-cycles are sufficient.

How deep should we go? As far as we can! Until direct solve is trivial.



Pick powers of 2 as mesh spacing

Go down until mesh $h=1/2$.

$h = 1/2$

coarsest

one-unknown $\Rightarrow$ trivial solve

How much work in a V-cycle?

On the fine mesh, have to smooth ν times

$\hookrightarrow O(N)$ work

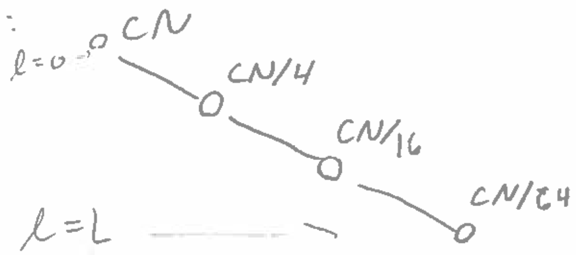
$\hookrightarrow$ compute residual $\rightarrow O(N)$ work

$\hookrightarrow$ restrict to coarse mesh $\rightarrow O(N)$ work

$\hookrightarrow$ correct/interp $\rightarrow O(N)$ work

All fine mesh ops. are same order of work $\Rightarrow$ total work is $C \cdot N$ for fine mesh

2D:



$\Rightarrow$ total work:

$$\sum_{l=0}^{L} CN\left(\tfrac{1}{4}\right)^{l} \underset{\text{large } L}{\overset{\text{limit in}}{\approx}} CN\left(\frac{1}{1-\frac{1}{4}}\right) = \frac{4}{3}CN$$

Hence V-cycle work is not very different from work on fine level

GS-RB, FW, bilin. int.

2D: $\nu = 2$

two-grid $\rho \approx 0.074$

V-cycle $\rho \approx 0.10$

64×64 mesh

tol $= 10^{-6}$

	Iterations	work/it.	total work
SOR	144	1	144
MG	6	$\frac{4}{3}(2+4)=8$	48

(work/it.: cost of 1 smooth step; total work: cost of 1 smooth step)

] 3x faster whew

256×256 mesh

tol $= 10^{-7}$

	# its	work/it	total work
SOR	658	1	658
MG	7	8	56

] ~13x faster!

Order of magnitude faster!

$$L\vec{u} = \vec{f}$$

$$-\frac{4}{h^2}u = f$$

$$u = -f \cdot h^2/4$$

Conjugate Gradient Method: $Au = f$

can solve using matrix-vector product: given u, compute Au; don't need explicit matrix.

where A is symmetric positive-definite
$A^T = A$ ↳ $y^TAy > 0 \quad \forall y \neq 0$
$\Rightarrow$ all eigenvalues of A are positive & real & complete set of orthogonal eigenvectors

CG is an example of a Krylov method → others: minres - symmetric indefinite A; gmres - not sym, indefinite

CG is related to minimization
↳ Suppose A is sym. pos. def., $N \times N$
Define functional $\phi: \mathbb{R}^N \to \mathbb{R}$: $\phi(\underline{u}) = \frac{1}{2}\underline{u}^TA\underline{u} - \underline{u}^Tf$

Soln to $A\underline{u} = \underline{f}$ is the minimizer of ϕ

Pf: $\nabla\phi = \frac{1}{2}A\underline{u} + \frac{1}{2}A^T\underline{u} - \underline{f} = A\underline{u} - f = 0$ for $A\underline{u} = \underline{f}$

$\nabla\nabla\phi = A$ ← A pos. def., so ϕ is convex. $\Rightarrow A\underline{u} = \underline{f}$ soln minimizes ϕ ∎

Method of Steepest Descents.

Have $\underline{u}_k$, k^{th} iterate, want method for generating $\underline{u}_{k+1}$.

$\Rightarrow \underline{u}_{k+1} = \underline{u}_k + \alpha \underline{r}_k$

Pick α to minimize $\phi(\underline{u}_{k+1}) = \min_\alpha \phi(\underline{u}_k + \alpha \underline{r}_k)$

$\nabla\phi(u_k) = A\underline{u}_k - f = -\underline{r}_k$

$\Rightarrow \alpha = \arg\min_\alpha \phi(\underline{u}_k + \alpha\underline{r}_k)$

compute: $\frac{d}{d\alpha}\phi(\underline{u} + \alpha\underline{r}) = 0$.

$\phi(\underline{u} + \alpha\underline{r}) = \frac{1}{2}(\underline{u} + \alpha\underline{r})^TA(\underline{u} + \alpha\underline{r}) - (\underline{u} + \alpha\underline{r})^Tf$
$= \frac{1}{2}\underline{u}^TA\underline{u} - \underline{u}^Tf + \alpha(\frac{1}{2}\underline{r}^TA\underline{u} + \frac{1}{2}\underline{u}^TA\underline{r} - \underline{r}^Tf)$
$+ \alpha^2 \frac{1}{2}\underline{r}^TA\underline{r}$

same since scalar & A symm.

$\Rightarrow \frac{d}{d\alpha}\phi(\underline{u} + \alpha\underline{r}) = (\underline{r}^TA\underline{u} - \underline{r}^Tf) + \alpha\underline{r}^TA\underline{r}$
$0 = \underline{r}^T(A\underline{u} - \underline{f}) + \alpha\underline{r}^TA\underline{r} = -\underline{r}^T\underline{r} + \alpha\underline{r}^TA\underline{r}$

note $\frac{d^2}{d\alpha^2}\phi(\underline{u} + \alpha\underline{r}) = \underline{r}^TA\underline{r} > 0$ so convex!

$$\Rightarrow \alpha = \frac{\underline{r}^T\underline{r}}{\underline{r}^TA\underline{r}}$$

Steepest Descents Algorithm

initialize u_0
loop in k
$\quad r_k = f - A u_k$
$\quad$ check $\|r_k\|$ for stopping
$$\alpha = \frac{r_k^T r_k}{r_k^T A r_k}$$
$$u_{k+1} = u_k + \alpha r_k$$

work:
mainly 2 matrix-vector products

faster: $\underline{r}_{k+1} = \underline{f} - A\underline{u}_{k+1}$
$$= \underline{f} - A(\underline{u}_k + \alpha r_k)$$
$$= \underline{r}_k - \alpha \underbrace{A\underline{r}_k}$$
which we already compute in finding A.

More efficient method

initialize u_0 & $r_0 = f - Au_0$
loop k
$\quad$ check $\|r_k\|$
$\quad$ compute $\quad w = Ar_k$
$$\alpha = \frac{r_k^T r_k}{r_k^T w}$$
$$u_{k+1} = u_k + \alpha r_k$$
$$r_{k+1} = r_k - \alpha w$$

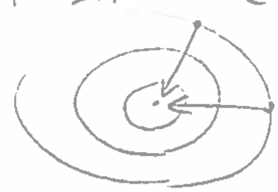
only one matrix-vector-product in $w = Ar_k$ step!

☆ residual used as stopping criterion b/c it is available & generally more robust → doesn't dep. on it. metho

Level curves of ϕ are ellipses
the shape is related to the eigenvalues

$$\underline{r} = \lambda(\underline{u} - \underline{v})$$
$$\underline{f} - A\underline{v} = \lambda(\underline{u} - \underline{v})$$
$$A\underline{u} - A\underline{v} = \lambda(\underline{u} - \underline{v})$$
$$A(\underline{u} - \underline{v}) = \lambda(\underline{u} - \underline{v})$$
Hence $\underline{u} - \underline{v}$ is an eigenvector of A
so residual at these points (in pic) are

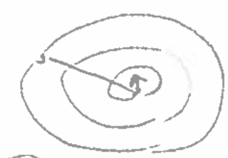
If $\lambda_1 = \lambda_2$



any starti pt. will ta 1 it. to find sol.

If $\frac{\lambda_2}{\lambda_1} \approx 1$

takes like 2 its.



metho conve quickly

If $\frac{\lambda_2}{\lambda_1} >> 1$

☆ lots of little steps slow convergence!

The convergence rate of steepest descents depends on $\kappa = \|A\|_2 \|A^{-1}\|_2$. the condition #.

For symm. matrix, $\kappa = \dfrac{\max_k |\lambda_k|}{\min_k |\lambda_k|}$

MAT228A - Lecture 19 - 11/29

Recall: Have problem $A\vec{u}=\vec{f}$ where A is symm, pos. definite

- Only need to be able to compute matrix-vector products!

Steepest Descents guaranteed to converge!

$\hookrightarrow \phi(u) = u^T A u - u^T f$ ← minimize this functional to solve $A\vec{u}=\vec{f}$.

Speed depends on condition #, $K = \|A\|_2 \cdot \|A^{-1}\|_2$

For symm. matrix, $K = \max_k |\lambda_k| / \min_k |\lambda_k|$

Conjugate Gradient Descent - search direction is different than residual

Let p_k be the vector to follow/search: $u_{k+1} = u_k + \alpha p_k$.

Follow this search path until ϕ increases, i.e. the minimizer of ϕ on a 1D space

Find $\alpha = \dfrac{p_k^T r_k}{p_k^T A p_k}$

Start w/ Conj. grad. in 2D:

- initial guess u_0
  - $\hookrightarrow$ compute initial residual r_0
  - $\hookrightarrow$ initialize $p_0 = r_0$

$\rightarrow$ one step of steepest descent to get u_1: $u_1 = u_0 + \alpha p_0$, $\alpha = \dfrac{p_0^T r_0}{p_0^T A p_0}$

- pick p_1 so that $p_0^T A p_1 = 0$.

$\hookrightarrow$ new direction not orthogonal to old, but A-conjugate, i.e. orthogonal in the A-inner product $\langle p_0, p_1 \rangle_A = p_0^T A p_1$. since A is symm. pos. def.

Why?

$\hookrightarrow p_0$ is tangent to level set of ϕ at $\phi(u_1)$, i.e. $p_0^T r_1 = 0$

$\Rightarrow p_0^T(f - Au_1) = 0 \Rightarrow p_0^T(Au - Au_1) = 0 \Rightarrow p_0^T A(u - u_1) = 0$

So p_1 will be in the direction of $u - u_1$.

$\Rightarrow u_2$ is the solution (in 2D).

Conj. Grad Desc. in 3D:

- Initialize u_0, compute r_0, set $p_0 = r_0$.
- get $u_1 = u_0 + \alpha p_0$, $\alpha = \frac{p_0^T r_0}{p_0^T A p_0}$.
- ~~Pick p_1 to be A-conjugate to p_0 (2D space to pick from)~~
  - ↳ pick some direction in that space (projection of residual into A-conjugacy sp
- p_1 & r_1 define new α, $u_2 = u_1 + \alpha p_1$.
- p_0 & p_1 span a plane $u_2 + c_0 p_0 + c_1 p_1$ which is tangent to level surface of φ at $\varphi(u_2)$.
- Now pick p_2 A-conjugate to both p_0 & p_1.
  - ↳ Minimize φ along p_2 : $u_3 = u_2 + \alpha p_2$ (solved).

[Done b/c $(c_0 p_0 + c_1 p_1)^T r_2 = 0$

$$(c_0 p_0 + c_1 p_1)^T (f - A u_2) = 0$$
$$(c_0 p_0 + c_1 p_1)^T (A u - A u_2) = 0$$
$$(c_0 p_0 + c_1 p_1)^T A \underbrace{(u - u_2)}_{\text{pick } p_2 \text{ in this direction!}} = 0$$
]

☆ CGD guaranteed to converge (w/ infinite arithmetic precision, no round-off) to the exact solution in N steps (N = # of unknowns).
Usually get "close" in far fewer steps.

Pseudo code for CGD algorithm: • initialize: u_0, $r_0 = f - A u_0$, $p_0 = r_0$

- loop k
  - ↳ $w = A p_k$
  - ↳ $\alpha = \frac{r_k^T r_k}{w^T p_k}$
  - ↳ $u_{k+1} = u_k + \alpha p_k$
  - ↳ $r_{k+1} = r_k - \alpha w$
  - ↳ check $\|r_{k+1}\|$ for stopping

(still in loop)

- ↳ $\beta = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
- ↳ $p_{k+1} = r_{k+1} + \beta p_k$ ← this is r_{k+1} with stuff in the $(p_0, p_1, \ldots, p_k)$-space projected off in A-inner pro

end loop

☆ save $r_k^T r_k$ since we use it several times.

Sketch of analysis of convergence of CGD:

B/C A is sym. pos. def., can define inner product $\langle u_0, u_1 \rangle_A = u_0^T A u_1$
& a norm $\|u\|_A^2 = u^T A u$.

Can show that error after n iterations of CGD is

$$\|e_n\|_A^2 \le 2 \left(\frac{\sqrt{K} - 1}{\sqrt{K} + 1} \right)^n \|e_0\|_A$$

If K is close to 1, error converges rapidly!
If K large, error converges very slowly.

Preconditioning - idea is to change condition number & solve a related problem.

Thm (from some book) The vectors in the CG algorithm have the following properties provided $r_k \neq 0$:

1) p_k is A-conjugate to all previous search directions $p_0, p_1, \ldots, p_{k-1}$
2) r_k is orthogonal to all previous residuals $r_0, r_1, \ldots, r_{k-1}$.
3) The following subspaces of $\mathbb{R}^N$ are identical:
   $\text{span}(p_0, p_1, \ldots, p_{k-1})$, $\text{span}(r_0, Ar_0, A^2 r_0, \ldots, A^{k-1} r_0)$,
   $\text{span}(Ae_0, A^2 e_0, \ldots, A^k e_0)$

Define $K_n = \text{span}(r_0, Ar_0, \ldots, A^{n-1} r_0)$, n-dim'l Krylov space associated w/ r_0.

Krylov methods perform minimizations on successive/nested Krylov spaces.

Note $u_n \in u_0 + K_n$ (affine space) & u_n minimizes ϕ in this space.

$\rightarrow$ Minimizing ϕ is equivalent to minimizing $\|e_j\|_A^2$.

$\hookrightarrow \|e_j\|_A^2 = e_j^T A e_j = (u_j - u)^T A (u_j - u) = u_j^T A u_j - 2 u_j^T A u + u^T A u$

$= u_j^T A u_j - 2 u_j^T f + u^T A u$

$= 2\phi(u_j) + C$

MAT228A - Last Lecture - 12/1

Recall: Minimizing $\varphi(u)$ is equivalent to $\min \|e_k\|_A^2$ for CG.

Key thm: $\operatorname{span}(p_0, p_1, \dots, p_{k-1}) = \operatorname{span}(Ae_0, A^2 e_0, \dots, A^k e_0)$

$$u_k = u_0 + \alpha_0 p_0 + \dots + \alpha_{k-1} p_{k-1}$$

$$u_k = u_0 + c_1 A e_0 + c_2 A^2 e_0 + \dots + c_k A^k e_0$$

subtract solution!

$$e_k = e_0 + c_1 A e_0 + c_2 A^2 e_0 + \dots + c_k A^k e_0 = q(A) \cdot e_0$$

matrix polynomial of A.

CG picks $q \in \Pi_k$ = space of polynomials of deg. at most k with $q(0)=1$.
to minimize $\|e_k\|_A = \|q(A) e_0\|_A$.

Note: A is diagonalizable: $A = Q \Lambda Q^T$ and $Q^T = Q^{-1}$.

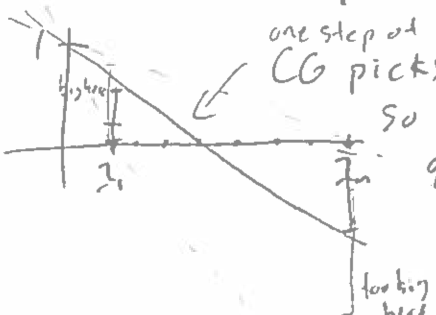
so $A^2 = Q \Lambda Q^{-1} Q \Lambda Q^{-1} = Q \Lambda^2 Q^{-1}$

& thus $A^k = Q \Lambda^k Q^{-1}$

Fact: $q(A) = Q\, q(\Lambda)\, Q^{-1} = Q \begin{pmatrix} q(\lambda_1) & & \\ & \ddots & \\ & & q(\lambda_n) \end{pmatrix} Q^{-1}$.

Can show $\|e_k\|_A^2 = \|q(A)e_0\|_A^2 \le \underbrace{\max_j (q(\lambda_j))^2}\, \|e_0\|_A^2$

our error bound comes from minimizing this (q on the spectrum of A).

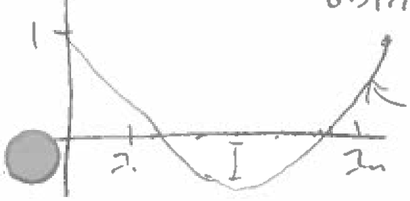


one step of CG picks $q_1(x)$ so that

$q_1(\lambda_1) = -q_1(\lambda_n)$

$\Rightarrow q_1(x) = 1 - \dfrac{2x}{\lambda_1 + \lambda_n}$

2 steps of CG: assuming eigenvalues uniformly distributed in $[\lambda_1, \lambda_n]$ (o/w no analytic soln)



satisfies $\min\limits_{q \in \Pi_2} \max\limits_{x \in [\lambda_1, \lambda_n]} |q(x)|$

In general, can use scaled & shifted Chebyshev polys. to get the bound

$$\|e_n\|_A = \|q_n(A) e_0\|_A \le 2\left(\frac{\sqrt{K}-1}{\sqrt{K}+1}\right)^n \|e_0\|_A$$

condition #

How well does this work? For Discrete Laplacian, $K = \Theta(h^{-2})$ since smallest $\lambda = \Theta(1)$, largest $\lambda = \Theta(\frac{1}{h^2})$

For large K, #iterations to converge to a rel. tol $\dfrac{\|e_n\|_A}{\|e_0\|_A} < \varepsilon$ is $\sqrt{K} \log \varepsilon$. So for ↑ its $\Theta(h^{-1})$ it.

For discrete Laplacian, #its. to converge to rel. tol. of ε $\left(\frac{\|e_k\|_A}{\|e_0\|_A} < \varepsilon\right)$ is $\sqrt{K}\log\varepsilon = \mathcal{O}(h^{-1}) = \mathcal{O}(n) = \mathcal{O}(\sqrt{N})$.

each iteration costs $\mathcal{O}(N)$, so in 2D, total work $= \mathcal{O}(N^{3/2})$ (same scaling as SOR).

What if we cluster the eigenvalues into groups? ~~Then the polynomials~~ we're minimizing over the spectrum can be lower order & CG converges way faster! This is Preconditioning

Preconditioning CG: $Au = f$ ← slow convergence if K is big for A:

Change the problem ⟹ $M^{-1}Au = M^{-1}f$ ← same solution u, but spectrum of $M^{-1}A$ is diff. than.

Pick M^{-1} s.t. $M^{-1}A$ has a smaller K & clustered eigenvalues.

For CG, the pre-conditioner M should be: 1) symmetric, positive-definite

2) $M^{-1}A$ better conditioned

3) M^{-1} easy to apply, i.e., $Mx = b$ easy to solve

☆ Don't need M, need to be able to apply M^{-1} ☆

Naively, M^{-1} should approx. A^{-1} so that $K \gtrsim 1$.

Problem: $M^{-1}A$ is not symmetric!

$Au = f \Rightarrow B^{-1}Au = B^{-1}f \Rightarrow (B^{-1}AB^{-T})(B^{T}u) = (B^{-1}f)$

left & right preconditioning

Define $\tilde{A} = B^{-1}AB^{-T}$, $\tilde{u} = B^{T}u$, $\tilde{f} = B^{-1}f \Rightarrow \tilde{A}\tilde{u} = \tilde{f}$.

↳ symmetric (obviously), pos. def. since $y^T B^{-1}AB^{-T}y = (B^{-T}y)^T AB^{-T}y = z^T A z > 0$ ✓ for $y \neq 0$.

Note: $B^{-T}B^{-1}AB^{-T}B^{T} = B^{-T}B^{-1}A = (BB^{T})^{-1}A = M^{-1}A \Rightarrow M = BB^{T}$

$\tilde{A}$ has the same eigenvalues as $M^{-1}A$ since $B^{-T}\tilde{A}B^{T}$ is just a change of basis.

Write CG in ~ variables & then transform back to original variables

$u_k = B^{-T}\tilde{u}_k$, $p_k = B^{-T}\tilde{p}_k$, $r_k = B\tilde{r}_k$, then B, B^T drop out of algorithm.

One step of CG: $\tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k \tilde{p}_k$

$B^T p_{k+1} = B^{-1}r_{k+1} + \beta_k B^T p_k \Rightarrow p_{k+1} = B^{-T}B^{-1}r_{k+1} + \beta_k p_k = M^{-1}r_{k+1} + \beta_k p_k$

Pre-conditioned Conjugate Gradient Descent Algorithm

initialize $r_0 = f - Au_0$

solve $Mz_0 = r_0$ or compute $z_0 = M^{-1}r_0$ (if have action apply M^{-1})

initialize $p_0 = z_0$

loop k

$w_k = Ap_k$

$$\alpha = \frac{z_k^T r_k}{p_k^T w_k}$$

$u_{k+1} = u_k + \alpha p_k$

$r_{k+1} = r_k - \alpha w_k$

check $\|r_{k+1}\|$ for stopping criterion

compute $z_{k+1} = M^{-1} r_{k+1}$

$$\beta = \frac{z_{k+1}^T r_{k+1}}{z_k^T r_k}$$

$p_{k+1} = z_{k+1} + \beta p_k$

end

How do we pick the pre-conditioner M^{-1}?

- Easy/Naive implementation: $M^{-1} = D^{-1}$, D is diag. of A ← does nothing for constant coefficient problem (ie. Poisson eqn) since D^{-1} is a scalar.

- Use other iteration schemes, but symmetized:
  - SSOR ← symmetric SOR, loop down mesh once, loop back up (2x total)
  - MG ← using a symmetric smoother (RB-BR, etc.)

- Approximate factorizations - incomplete LU/cholesky factorization
  - ↳ know nothing, just algebraic.