

FWAM Session B: Function Approximation and Differential Equations

Alex Barnett¹ and Keaton Burns²

Wednesday afternoon, 10/30/19

¹Center for Computational Mathematics, Flatiron Institute

²Center for Computational Astrophysics, Flatiron Institute, and Department of Mathematics, MIT

LECTURE 1

interpolation, integration, differentiation, spectral methods

Goals and plan

Overall: graph of $f(x)$ needs ∞ number of points to describe, so how handle f to user-specified accuracy in computer w/ least cost? (bytes/flops)

Goals and plan

Overall: graph of $f(x)$ needs ∞ number of points to describe, so how handle f to user-specified accuracy in computer w/ least cost? (bytes/flops)

- Interpolation: also key to numerical ODE/PDEs...

task: given exact $f(x_j)$ at some x_j , model $f(x)$ at other points x ?

App: cheap but accurate “look-up table” for possibly expensive func.

Contrast: fit noisy data = learning (pdf for) params in model, via likelihood/prior

- Numerical integration:

App: computing expectation values, given a pdf or quantum wavefunc.

App: integral equation methods for PDEs (Jun Wang's talk)

- Numerical differentiation:

App: build a matrix (linear system) to approximate an ODE/PDE (Lecture II)

App: get gradient ∇f , eg for optimization (cf adjoint methods)

Goals and plan

Overall: graph of $f(x)$ needs ∞ number of points to describe, so how handle f to user-specified accuracy in computer w/ least cost? (bytes/flops)

- Interpolation: also key to numerical ODE/PDEs...

task: given exact $f(x_j)$ at some x_j , model $f(x)$ at other points x ?

App: cheap but accurate “look-up table” for possibly expensive func.

Contrast: fit noisy data = learning (pdf for) params in model, via likelihood/prior

- Numerical integration:

App: computing expectation values, given a pdf or quantum wavefunc.

App: integral equation methods for PDEs (Jun Wang's talk)

- Numerical differentiation:

App: build a matrix (linear system) to approximate an ODE/PDE (Lecture II)

App: get gradient ∇f , eg for optimization (cf adjoint methods)

Key concepts:

convergence rate, degree of smoothness of f , global vs local,
spectral methods, adaptivity, rounding error & catastrophic cancellation

Goals and plan

Overall: graph of $f(x)$ needs ∞ number of points to describe, so how handle f to user-specified accuracy in computer w/ least cost? (bytes/flops)

- Interpolation: also key to numerical ODE/PDEs...

task: given exact $f(x_j)$ at some x_j , model $f(x)$ at other points x ?

App: cheap but accurate “look-up table” for possibly expensive func.

Contrast: fit noisy data = learning (pdf for) params in model, via likelihood/prior

- Numerical integration:

App: computing expectation values, given a pdf or quantum wavefunc.

App: integral equation methods for PDEs (Jun Wang's talk)

- Numerical differentiation:

App: build a matrix (linear system) to approximate an ODE/PDE (Lecture II)

App: get gradient ∇f , eg for optimization (cf adjoint methods)

Key concepts:

convergence rate, degree of smoothness of f , global vs local,
spectral methods, adaptivity, rounding error & catastrophic cancellation

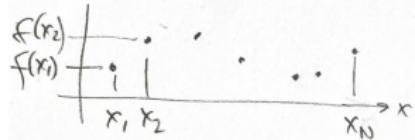
Plus: good 1D tools, pointers to codes, higher dim methods, opinions!

Interpolation in 1D ($d = 1$)

Say $y_j = f(x_j)$ known at nodes $\{x_j\}$ N -pt "grid"

note: exact data, not noisy

want interpolant $\tilde{f}(x)$, s.t. $\tilde{f}(x_j) = y_j$

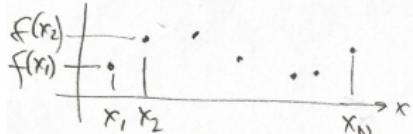


Interpolation in 1D ($d = 1$)

Say $y_j = f(x_j)$ known at nodes $\{x_j\}$ N -pt "grid"

note: exact data, not noisy

want interpolant $\tilde{f}(x)$, s.t. $\tilde{f}(x_j) = y_j$



hopeless w/o assumptions on f , eg smoothness, otherwise...

- extra info helps, eg f periodic, or $f(x) = \text{smooth} \cdot |x|^{-1/2}$

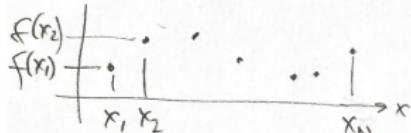


Interpolation in 1D ($d = 1$)

Say $y_j = f(x_j)$ known at nodes $\{x_j\}$ N -pt "grid"

note: exact data, not noisy

want interpolant $\tilde{f}(x)$, s.t. $\tilde{f}(x_j) = y_j$



hopeless w/o assumptions on f , eg smoothness, otherwise...

- extra info helps, eg f periodic, or $f(x) = \text{smooth} \cdot |x|^{-1/2}$



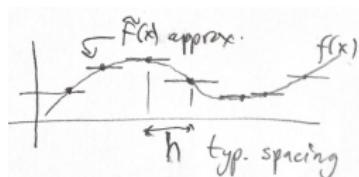
Simplest: use value at x_j nearest to x

"snap to grid"

Error $\max_x |\tilde{f}(x) - f(x)| = \mathcal{O}(h)$ as $h \rightarrow 0$

holds if f' bounded; ie f can be nonsmooth but not crazy

Recap notation " $\mathcal{O}(h)$ ": exists C, h_0 s.t. error $\leq Ch$ for all $0 < h < h_0$

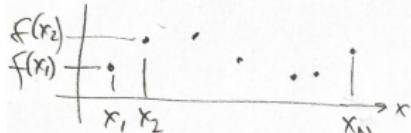


Interpolation in 1D ($d = 1$)

Say $y_j = f(x_j)$ known at nodes $\{x_j\}$ N -pt "grid"

note: exact data, not noisy

want interpolant $\tilde{f}(x)$, s.t. $\tilde{f}(x_j) = y_j$



hopeless w/o assumptions on f , eg smoothness, otherwise...

- extra info helps, eg f periodic, or $f(x) = \text{smooth} \cdot |x|^{-1/2}$



Simplest: use value at x_j nearest to x

"snap to grid"

Error $\max_x |\tilde{f}(x) - f(x)| = \mathcal{O}(h)$ as $h \rightarrow 0$

holds if f' bounded; ie f can be nonsmooth but not crazy

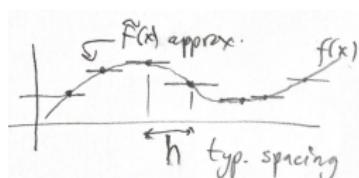
Recap notation " $\mathcal{O}(h)$ ": exists C, h_0 s.t. error $\leq Ch$ for all $0 < h < h_0$

Piecewise linear:

"connect the dots"

max error = $\mathcal{O}(h^2)$ as $h \rightarrow 0$

needs f'' bounded, ie smoother than before



Message: a higher order method is *only* higher order if f smooth enough



Interlude: convergence rates

Should know or measure convergence rate of any method you use

- “effort” parameter N eg # grid-points = $1/h^d$ where h = grid spacing, d = dim
- We just saw algebraic conv. error = $\mathcal{O}(N^{-p})$, for order $p = 1, 2$

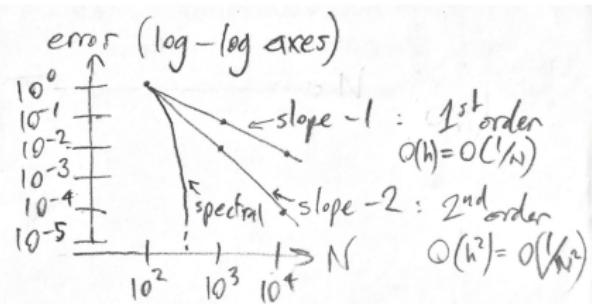
Interlude: convergence rates

Should know or measure convergence rate of any method you use

- “effort” parameter N eg # grid-points = $1/h^d$ where h = grid spacing, d = dim

We just saw algebraic conv. error = $\mathcal{O}(N^{-p})$, for order $p = 1, 2$

There's only one graph in numerical analysis: “relative error vs effort”



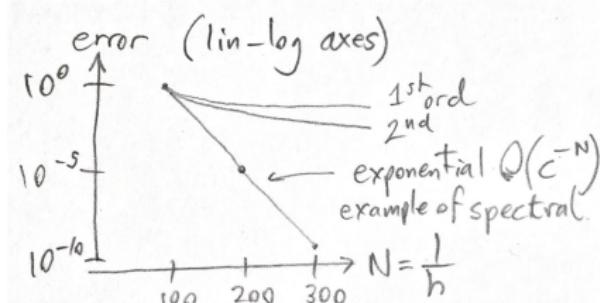
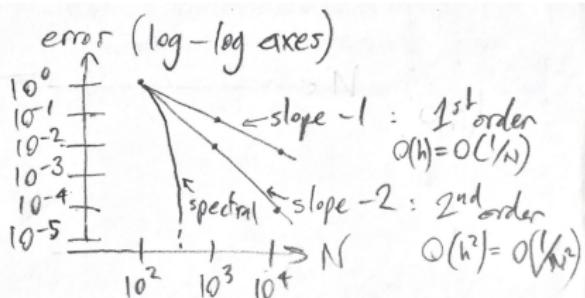
Interlude: convergence rates

Should know or measure convergence rate of any method you use

- “effort” parameter N eg # grid-points = $1/h^d$ where h = grid spacing, d = dim

We just saw algebraic conv. error = $\mathcal{O}(N^{-p})$, for order $p = 1, 2$

There's only one graph in numerical analysis: “relative error vs effort”



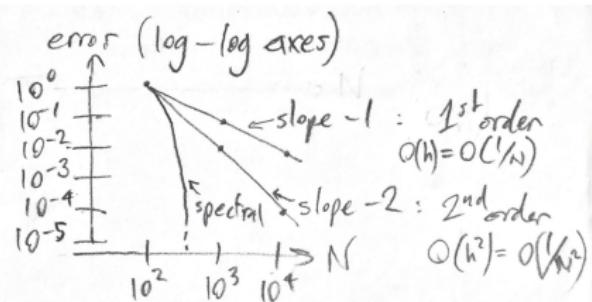
Interlude: convergence rates

Should know or measure convergence rate of any method you use

- “effort” parameter N eg # grid-points = $1/h^d$ where h = grid spacing, d = dim

We just saw algebraic conv. error = $\mathcal{O}(N^{-p})$, for order $p = 1, 2$

There's only one graph in numerical analysis: “relative error vs effort”



Note how spectral gets many digits for small N

crucial for eg 3D prob.

“spectral” = “superalgebraic”, beats $\mathcal{O}(N^{-p})$ for any p

- how many digits to you want? for 1-digit (10% error), low order ok, easier to code

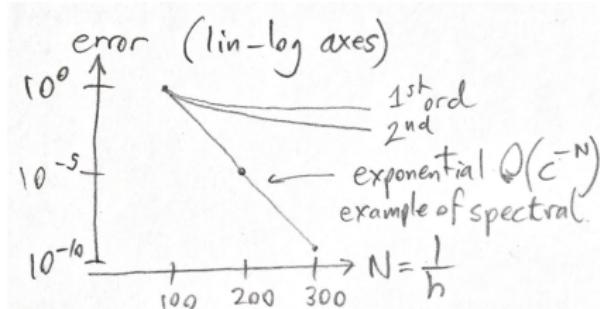
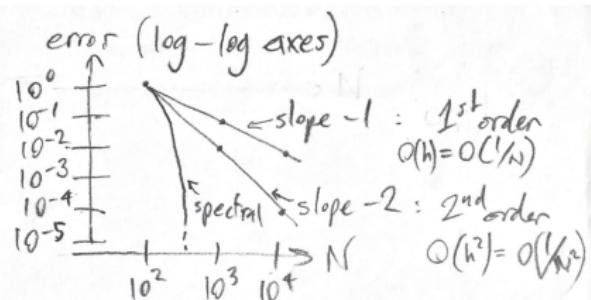
Interlude: convergence rates

Should know or measure convergence rate of any method you use

- “effort” parameter N eg # grid-points = $1/h^d$ where h = grid spacing, d = dim

We just saw algebraic conv. error = $\mathcal{O}(N^{-p})$, for order $p = 1, 2$

There's only one graph in numerical analysis: “relative error vs effort”



Note how spectral gets many digits for small N

crucial for eg 3D prob.

“spectral” = “superalgebraic”, beats $\mathcal{O}(N^{-p})$ for any p

- how many digits to you want? for 1-digit (10% error), low order ok, easier to code

<rant> test your code w/ known exact soln to check error conv. <\rant>

How big is prefactor C in error $\leq Ch^p$? Has asympt. rate even kicked in yet? :)

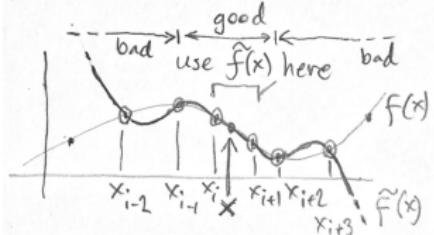
Higher-order interpolation for smooth f : the local idea

Pick a p , eg 6. For any target x , use only the nearest p nodes:

Exists unique degree- $(p-1)$ poly, $\sum_{k=0}^{p-1} c_k x^k$
which matches local data $(x_j, y_j)_{j=1}^p$

generalizes piecewise lin. idea

do not eval poly outside its central region!



Higher-order interpolation for smooth f : the local idea

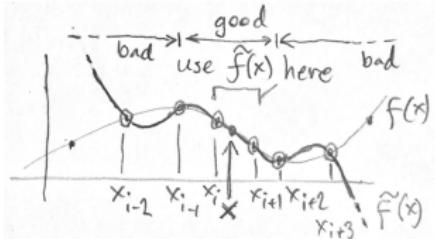
Pick a p , eg 6. For any target x , use only the nearest p nodes:

Exists unique degree- $(p-1)$ poly, $\sum_{k=0}^{p-1} c_k x^k$
which matches local data $(x_j, y_j)_{j=1}^p$

generalizes piecewise lin. idea

do not eval poly outside its central region!

- error $\mathcal{O}(h^p)$, ie high order, but \tilde{f} not continuous ($\tilde{f} \notin C$) has small jumps
if must have cont, recommend splines, eg cubic $p = 3$: $\tilde{f} \in C^2$, meaning \tilde{f}'' is cont.



Higher-order interpolation for smooth f : the local idea

Pick a p , eg 6. For any target x , use only the nearest p nodes:

Exists unique degree- $(p-1)$ poly, $\sum_{k=0}^{p-1} c_k x^k$
which matches local data $(x_j, y_j)_{j=1}^p$

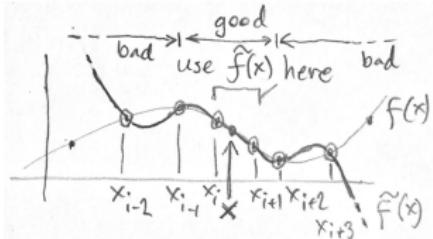
generalizes piecewise lin. idea

do not eval poly outside its central region!

- error $\mathcal{O}(h^p)$, ie high order, but \tilde{f} not continuous ($\tilde{f} \notin C$) has small jumps
if must have cont, recommend splines, eg cubic $p = 3$: $\tilde{f} \in C^2$, meaning \tilde{f}'' is cont.

How to find this degree- $(p-1)$ poly?

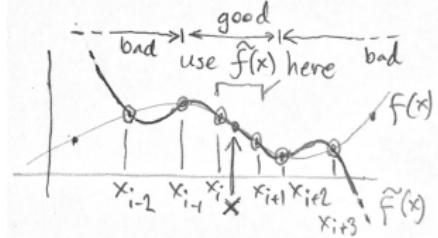
- 1) crafty: solve square lin sys for coeffs $\sum_{k < p} x_j^k c_k = y_j \quad j = 1, \dots, p$
ie, $V\mathbf{c} = \mathbf{y}$ V ="Vandermonde" matrix, is ill-cond. but works



Higher-order interpolation for smooth f : the local idea

Pick a p , eg 6. For any target x , use only the nearest p nodes:

Exists unique degree- $(p-1)$ poly, $\sum_{k=0}^{p-1} c_k x^k$
which matches local data $(x_j, y_j)_{j=1}^p$



generalizes piecewise lin. idea

do not eval poly outside its central region!

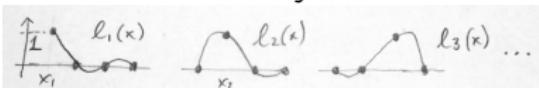
- error $\mathcal{O}(h^p)$, ie high order, but \tilde{f} not continuous ($\tilde{f} \notin C$) has small jumps
if must have cont, recommend splines, eg cubic $p = 3$: $\tilde{f} \in C^2$, meaning \tilde{f}'' is cont.

How to find this degree- $(p-1)$ poly?

1) crafty: solve square lin sys for coeffs $\sum_{k < p} x_j^k c_k = y_j \quad j = 1, \dots, p$
ie, $V\mathbf{c} = \mathbf{y}$ V = "Vandermonde" matrix, is ill-cond. but works

2) traditional: barycentric formula $\tilde{f}(x) = \frac{\sum_{j=1}^p \frac{y_j}{x-x_j} w_j}{\sum_{j=1}^p \frac{1}{x-x_j} w_j} \quad w_j = \frac{1}{\prod_{i \neq j} (x_j - x_i)}$ [Tre13, Ch. 5]

Either way, $\tilde{f}(x) = \sum_{j=1}^p y_j \ell_j(x)$ where $\ell_j(x)$ is j th Lagrange basis func:

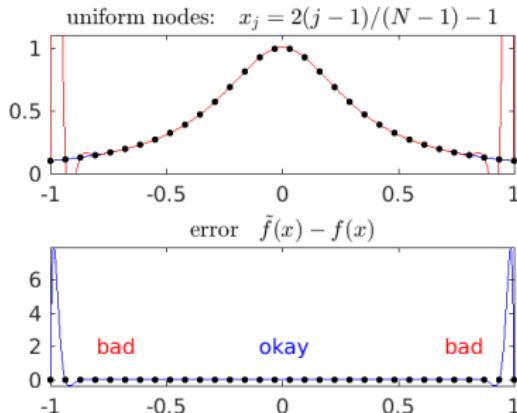


Global polynomial (Lagrange) interpolation?

Want increase order p . Use *all* data, get single $\tilde{f}(x)$, so $p = N$? "global"

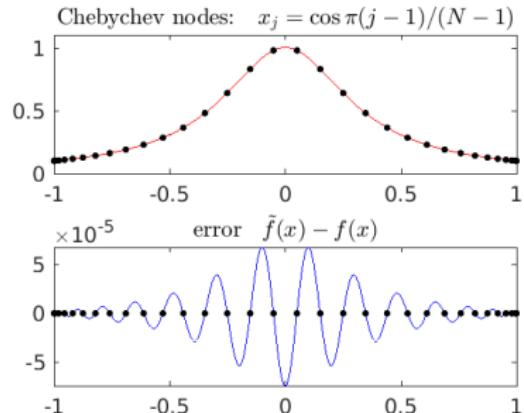
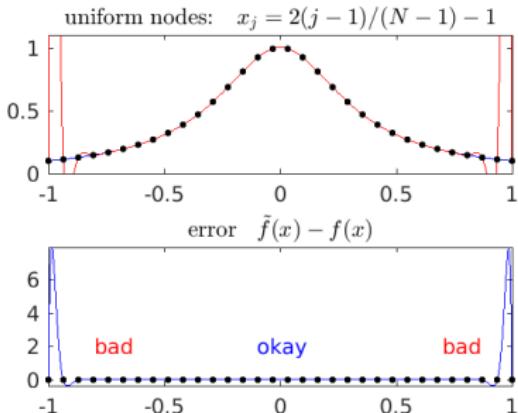
Global polynomial (Lagrange) interpolation?

Want increase order p . Use *all* data, get single $\tilde{f}(x)$, so $p = N$? "global"
 $p = N = 32$, smooth (analytic) $f(x) = \frac{1}{1+9x^2}$ on $[-1, 1]$: (Runge 1901)



Global polynomial (Lagrange) interpolation?

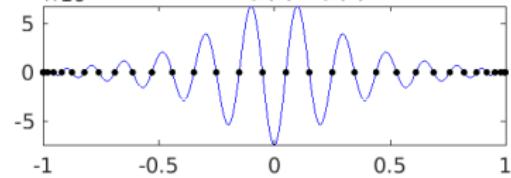
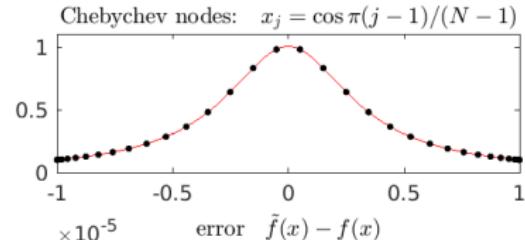
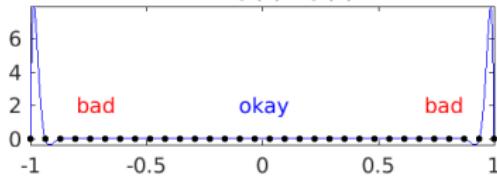
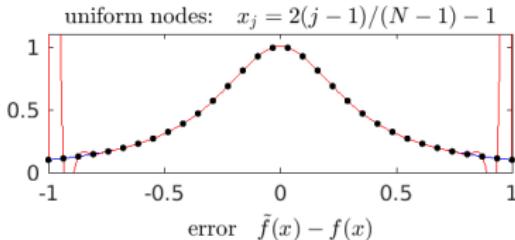
Want increase order p . Use *all* data, get single $\tilde{f}(x)$, so $p = N$? "global"
 $p = N = 32$, smooth (analytic) $f(x) = \frac{1}{1+9x^2}$ on $[-1, 1]$: (Runge 1901)



- warning: unif. grid, global interp. fails → only use locally in central region

Global polynomial (Lagrange) interpolation?

Want increase order p . Use all data, get single $\tilde{f}(x)$, so $p = N$? "global"
 $p = N = 32$, smooth (analytic) $f(x) = \frac{1}{1+9x^2}$ on $[-1, 1]$: (Runge 1901)



- warning: unif. grid, global interp. fails → only use locally in central region

But exists good choice of nodes...

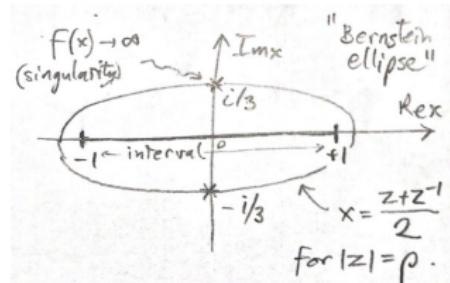
"Chebychev": means non-unif. grid density $\sim \frac{1}{\sqrt{1-x^2}}$

- our first spectral method

$$\max \text{err} = \mathcal{O}(\rho^{-N})$$

exponential conv!

$\rho > 1$ "radius" of largest ellipse in which f analytic



Node choice and adaptivity

Recap: poly approx. $f(x)$ on $[a, b]$: exist good & bad node sets $\{x_j\}_{j=1}^N$

Question: Do you get to choose the set of nodes at which f known?

- data fitting applications: No (or noisy variants: kriging, Gaussian processes, etc)
use local poly (central region only!), or something stable (eg splines) [GC12]
- almost all else, interp., quadrature, PDE solvers: Yes so pick good nodes!

Node choice and adaptivity

Recap: poly approx. $f(x)$ on $[a, b]$: exist good & bad node sets $\{x_j\}_{j=1}^N$

Question: Do you get to choose the set of nodes at which f known?

- data fitting applications: No (or noisy variants: kriging, Gaussian processes, etc)
use local poly (central region only!), or something stable (eg splines) [GC12]
- almost all else, interp., quadrature, PDE solvers: Yes so pick good nodes!

Adaptivity idea global is inefficient if f smooth in most places, but not everywhere

Node choice and adaptivity

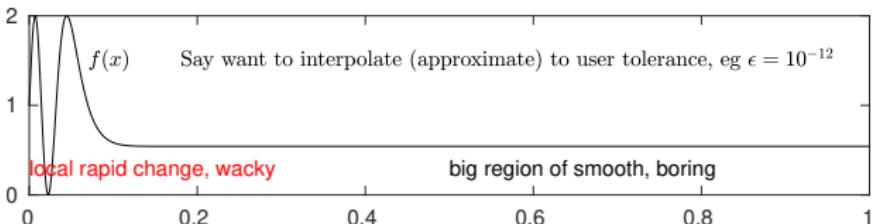
Recap: poly approx. $f(x)$ on $[a, b]$: exist good & bad node sets $\{x_j\}_{j=1}^N$

Question: Do you get to choose the set of nodes at which f known?

- data fitting applications: No (or noisy variants: kriging, Gaussian processes, etc)
use local poly (central region only!), or something stable (eg splines) [GC12]
- almost all else, interp., quadrature, PDE solvers: Yes so pick good nodes!

Adaptivity idea

global is inefficient if f smooth in most places, but not everywhere



Node choice and adaptivity

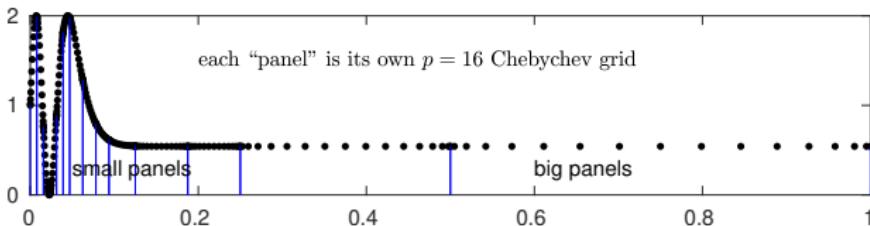
Recap: poly approx. $f(x)$ on $[a, b]$: exist good & bad node sets $\{x_j\}_{j=1}^N$

Question: Do you get to choose the set of nodes at which f known?

- data fitting applications: No (or noisy variants: kriging, Gaussian processes, etc)
use local poly (central region only!), or something stable (eg splines) [GC12]
- almost all else, interp., quadrature, PDE solvers: Yes so pick good nodes!

Adaptivity idea

global is inefficient if f smooth in most places, but not everywhere



automatically split
(recursively) panels
until max err $\leq \epsilon$

via test for local error

1D adaptive interpolator codes to try:

- [github:dbstein/function_generator](https://github.com/dbstein/function_generator) py+numba, fast (Stein '19)
- chebfun for MATLAB big- N Cheb. grids done via FFTs! (Trefethen et al.)

App.: replace nasty expensive $f(x)$ by cheap one!

optimal “look-up table”

Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Instead of poly's, use truncated series $\tilde{f}(x) = \sum_{|k| < N/2} c_k e^{ikx}$ "trig. poly"

Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

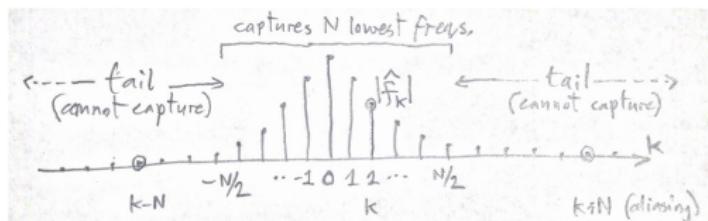
Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Instead of poly's, use truncated series $\tilde{f}(x) = \sum_{|k| < N/2} c_k e^{ikx}$ "trig. poly"

What's best you can do?

get N coeffs right $c_k = \hat{f}_k$

error \sim size of tail $\{\hat{f}_k\}_{|k| \geq N/2}$



Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

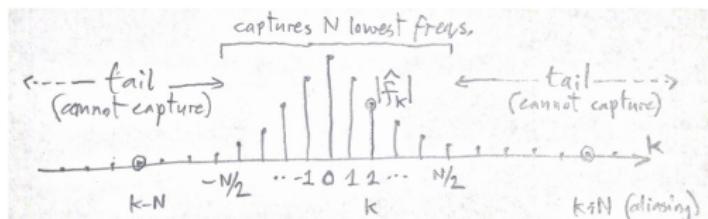
Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Instead of poly's, use **truncated** series $\tilde{f}(x) = \sum_{|k| < N/2} c_k e^{ikx}$ "trig. poly"

What's best you can do?

get N coeffs right $c_k = \hat{f}_k$

error \sim size of tail $\{\hat{f}_k\}_{|k| \geq N/2}$



How read off c_k from *samples* of f on a grid?

uniform grid best (unlike for poly's!); non-uniform needs linear solve, slow $\mathcal{O}(N^3)$ effort

Uniform grid $x_j = \frac{2\pi j}{N}$, set $c_k = \frac{1}{N} \sum_{j=1}^N e^{ikx_j} f(x_j)$ simply $\mathbf{c} = FFT[\mathbf{f}]$

Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

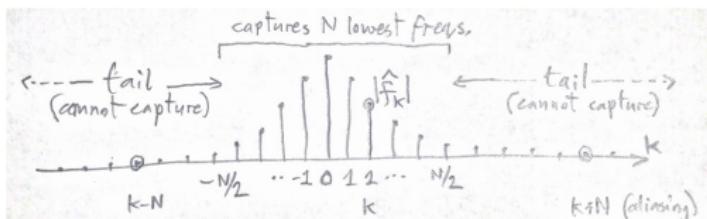
Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Instead of poly's, use **truncated** series $\tilde{f}(x) = \sum_{|k| < N/2} c_k e^{ikx}$ "trig. poly"

What's best you can do?

get N coeffs right $c_k = \hat{f}_k$

error \sim size of tail $\{\hat{f}_k\}_{|k| \geq N/2}$



How read off c_k from *samples* of f on a grid?

uniform grid best (unlike for poly's!); non-uniform needs linear solve, slow $\mathcal{O}(N^3)$ effort

Uniform grid $x_j = \frac{2\pi j}{N}$, set $c_k = \frac{1}{N} \sum_{j=1}^N e^{ikx_j} f(x_j)$ simply $\mathbf{c} = FFT[\mathbf{f}]$

easy to show $c_k = \dots + \hat{f}_{k-N} + \hat{f}_k + \hat{f}_{k+N} + \hat{f}_{k+2N} + \dots$

$= \hat{f}_k$ desired + $\sum_{m \neq 0} \hat{f}_{k+mN}$ aliasing error, small if tail small

Global interpolation of periodic functions I

Just did f on intervals $[a, b]$. global interp. (& integr., etc.) of smooth *periodic* f differs!

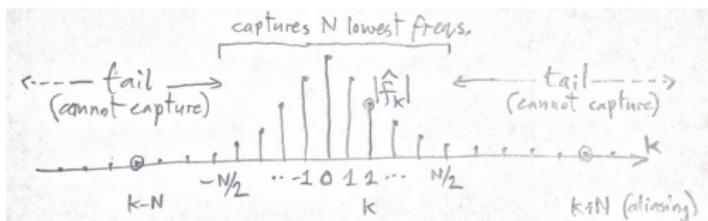
Periodic: $f(x + 2\pi) = f(x)$ for all x , $f(x) = \sum_{n \in \mathbb{Z}} \hat{f}_k e^{ikx}$ Fourier series

Instead of poly's, use **truncated** series $\tilde{f}(x) = \sum_{|k| < N/2} c_k e^{ikx}$ "trig. poly"

What's best you can do?

get N coeffs right $c_k = \hat{f}_k$

error \sim size of tail $\{\hat{f}_k\}_{|k| \geq N/2}$



How read off c_k from *samples* of f on a grid?

uniform grid best (unlike for poly's!); non-uniform needs linear solve, slow $\mathcal{O}(N^3)$ effort

Uniform grid $x_j = \frac{2\pi j}{N}$, set $c_k = \frac{1}{N} \sum_{j=1}^N e^{ikx_j} f(x_j)$ simply $\mathbf{c} = FFT[\mathbf{f}]$

easy to show $c_k = \dots + \hat{f}_{k-N} + \hat{f}_k + \hat{f}_{k+N} + \hat{f}_{k+2N} + \dots$

$= \hat{f}_k$ desired + $\sum_{m \neq 0} \hat{f}_{k+mN}$ aliasing error, small if tail small

Summary: given N samples $f(x_j)$, interp. error = truncation + aliasing

a crude bound is $\max_{x \in [0, 2\pi)} |\tilde{f}(x) - f(x)| \leq 2 \sum_{|k| \geq N/2} |\hat{f}_k|$

ie error controlled by sum of tail

Global interpolation of periodic functions II

As grow grid N , how accurate is it? just derived $\text{err} \sim \text{sum of } |\hat{f}_k| \text{ in tail } |k| \geq N/2$

$$\text{Now } \hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi} \int_0^{2\pi} f^{(p)}(x) \frac{e^{-ikx}}{(ik)^p} dx \quad \text{integr. by parts } p \text{ times}$$

So for a periodic $f \in C^p$, recall first p derivs of f bounded

$$\hat{f}_k = \mathcal{O}(k^{-p}), \text{ tail sum } \mathcal{O}(N^{1-p}) \quad (p-1)\text{th order acc.} \quad (\text{better: [Tre00]})$$

Global interpolation of periodic functions II

As grow grid N , how accurate is it? just derived $\text{err} \sim \text{sum of } |\hat{f}_k| \text{ in tail } |k| \geq N/2$

$$\text{Now } \hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi} \int_0^{2\pi} f^{(p)}(x) \frac{e^{-ikx}}{(ik)^p} dx \quad \text{integr. by parts } p \text{ times}$$

So for a periodic $f \in C^p$, recall first p derivs of f bounded

$$\hat{f}_k = \mathcal{O}(k^{-p}), \text{ tail sum } \mathcal{O}(N^{1-p}) \quad (p-1)\text{th order acc.} \quad (\text{better: [Tre00]})$$

Example of: f smoother \leftrightarrow faster \hat{f}_k tail decay \leftrightarrow faster convergence

Global interpolation of periodic functions II

As grow grid N , how accurate is it? just derived $\text{err} \sim \text{sum of } |\hat{f}_k| \text{ in tail } |k| \geq N/2$

Now $\hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi} \int_0^{2\pi} f^{(p)}(x) \frac{e^{-ikx}}{(ik)^p} dx$ integr. by parts p times

So for a periodic $f \in C^p$, recall first p derivs of f bounded

$$\hat{f}_k = \mathcal{O}(k^{-p}), \text{ tail sum } \mathcal{O}(N^{1-p}) \quad (p-1)\text{th order acc.} \quad (\text{better: [Tre00]})$$

Example of: f smoother \leftrightarrow faster \hat{f}_k tail decay \leftrightarrow faster convergence

Even smoother case: f analytic, so $f(x)$ analytic in some complex strip $|\operatorname{Im} x| \leq \alpha$
then $\hat{f}_k = \mathcal{O}(e^{-\alpha|k|})$, exp. conv. $\mathcal{O}(e^{-\alpha N/2})$ (fun proof: shift the contour)

as with Bernstein ellipse, to get exp. conv. rate need understand f off its real axis (wild!)

Global interpolation of periodic functions II

As grow grid N , how accurate is it? just derived $\text{err} \sim \text{sum of } |\hat{f}_k| \text{ in tail } |k| \geq N/2$

$$\text{Now } \hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi} \int_0^{2\pi} f^{(p)}(x) \frac{e^{-ikx}}{(ik)^p} dx \quad \text{integr. by parts } p \text{ times}$$

So for a periodic $f \in C^p$, recall first p derivs of f bounded

$$\hat{f}_k = \mathcal{O}(k^{-p}), \text{ tail sum } \mathcal{O}(N^{1-p}) \quad (p-1)\text{th order acc.} \quad (\text{better: [Tre00]})$$

Example of: f smoother \leftrightarrow faster \hat{f}_k tail decay \leftrightarrow faster convergence

Even smoother case: f analytic, so $f(x)$ analytic in some complex strip $|\operatorname{Im} x| \leq \alpha$
then $\hat{f}_k = \mathcal{O}(e^{-\alpha|k|})$, exp. conv. $\mathcal{O}(e^{-\alpha N/2})$ (fun proof: shift the contour)

as with Bernstein ellipse, to get exp. conv. rate need understand f off its real axis (wild!)

Smoothest case: “band-limited” f with $\hat{f}_k = 0, |k| > k_{\max}$,
then interpolant exact once $N > 2k_{\max}$

Global interpolation of periodic functions II

As grow grid N , how accurate is it? just derived $\text{err} \sim \text{sum of } |\hat{f}_k| \text{ in tail } |k| \geq N/2$

Now $\hat{f}_k = \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-ikx} dx = \frac{1}{2\pi} \int_0^{2\pi} f^{(p)}(x) \frac{e^{-ikx}}{(ik)^p} dx$ integr. by parts p times

So for a periodic $f \in C^p$, recall first p derivs of f bounded

$$\hat{f}_k = \mathcal{O}(k^{-p}), \text{ tail sum } \mathcal{O}(N^{1-p}) \quad (p-1)\text{th order acc.} \quad (\text{better: [Tre00]})$$

Example of: f smoother \leftrightarrow faster \hat{f}_k tail decay \leftrightarrow faster convergence

Even smoother case: f analytic, so $f(x)$ analytic in some complex strip $|\operatorname{Im} x| \leq \alpha$
then $\hat{f}_k = \mathcal{O}(e^{-\alpha|k|})$, exp. conv. $\mathcal{O}(e^{-\alpha N/2})$ (fun proof: shift the contour)
as with Bernstein ellipse, to get exp. conv. rate need understand f off its real axis (wild!)

Smoothest case: “band-limited” f with $\hat{f}_k = 0, |k| > k_{\max}$,
then interpolant exact once $N > 2k_{\max}$

That's theory. In real life you always measure your conv. order/rate!

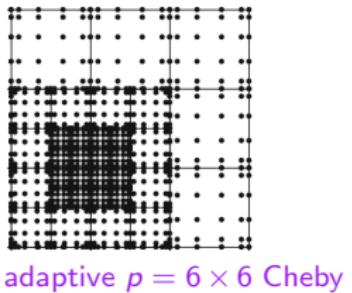
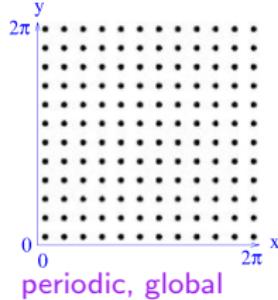
Messages:

- f smooth, periodic, global interpolation w/ uniform grid: spectral acc.
- key to spectral methods. FFT cost $\mathcal{O}(N \log N)$ swaps from $f(x_j)$ grid to \hat{f}_k

Flavor of interpolation in higher dims $d > 1$

If you *can* choose the nodes:

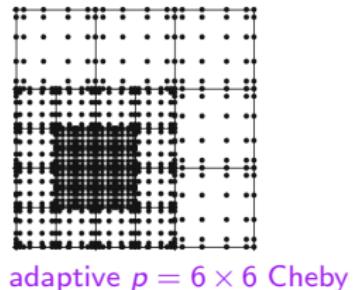
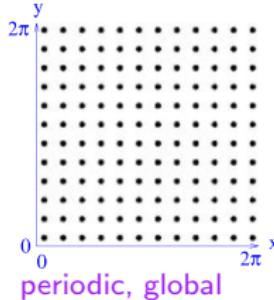
- tensor product of 1D grids
- either global
- or adaptively refined boxes



Flavor of interpolation in higher dims $d > 1$

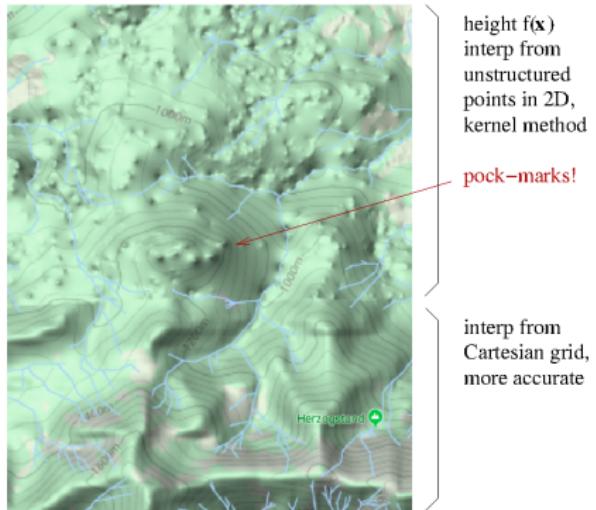
If you *can* choose the nodes:

tensor product of 1D grids
either global
or adaptively refined boxes



If *cannot* choose the nodes: interp. $f(\mathbf{x})$ from scattered data $\{\mathbf{x}_i\}$ is hard

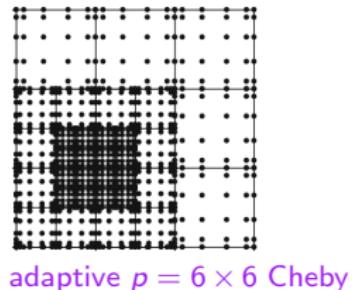
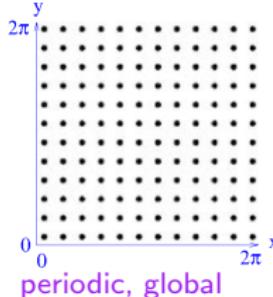
eg google terrain: $f(\mathbf{x})$ rough \rightarrow garbage:



Flavor of interpolation in higher dims $d > 1$

If you *can* choose the nodes:

tensor product of 1D grids
either global
or adaptively refined boxes



If *cannot* choose the nodes: interp. $f(\mathbf{x})$ from scattered data $\{\mathbf{x}_i\}$ is hard

eg google terrain: $f(\mathbf{x})$ rough → garbage:

But if know f smooth:

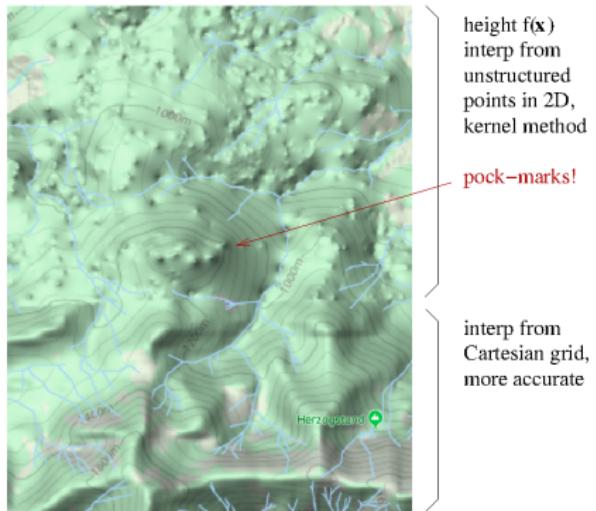
locally fit multivariate polynomials

If also data noisy, many methods:

kriging (Gauss. proc.), NUFFT, RBF...

If also high dim $d \gg 1$:

tensor train, neural networks...



Numerical integration (back to $d = 1$)

Task: eval. $\int_a^b f(x)dx$ accurately w/ least number of func. evals, N

"quadrature": nodes $\{x_j\}$, weights $\{w_j\}$, s.t. $\int_a^b f(x)dx \approx \sum_{j=1}^N w_j f(x_j)$

Idea: get interpolant \tilde{f} thru data $f(x_j) \rightarrow$ integrate that exactly

"interpolatory quadrature"

Numerical integration (back to $d = 1$)

Task: eval. $\int_a^b f(x)dx$ accurately w/ least number of func. evals, N

"quadrature": nodes $\{x_j\}$, weights $\{w_j\}$, s.t. $\int_a^b f(x)dx \approx \sum_{j=1}^N w_j f(x_j)$

Idea: get interpolant \tilde{f} thru data $f(x_j) \rightarrow$ integrate that exactly

Examples:

"interpolatory quadrature"



- local piecewise linear \rightarrow composite trapezoid rule
 $w_j = h$ except $h/2$ at ends. low-order, err $\mathcal{O}(N^{-2})$, avoid!
- N -node global poly \rightarrow gives $\{w_j\}$ integrating degree $N-1$ exactly
 f analytic: err $\mathcal{O}(\rho^{-N})$ solve lin sys $V^T w = \{\int_a^b x^k dx\}_{k=0}^{N-1}$ (Newton-Cotes)
- better: "Gaussian" $\{x_j, w_j\}$ integrates deg. $2N-1$ exactly! err $\mathcal{O}(\rho^{-2N})$

Adaptive quadrature (Gauss in each panel) excellent: codes quadgk, scipy, etc

Numerical integration (back to $d = 1$)

Task: eval. $\int_a^b f(x)dx$ accurately w/ least number of func. evals, N

"quadrature": nodes $\{x_j\}$, weights $\{w_j\}$, s.t. $\int_a^b f(x)dx \approx \sum_{j=1}^N w_j f(x_j)$

Idea: get interpolant \tilde{f} thru data $f(x_j) \rightarrow$ integrate that exactly

Examples:

"interpolatory quadrature"



- local piecewise linear \rightarrow composite trapezoid rule
 $w_j = h$ except $h/2$ at ends. low-order, err $\mathcal{O}(N^{-2})$, avoid!
 - N -node global poly \rightarrow gives $\{w_j\}$ integrating degree $N-1$ exactly
 f analytic: err $\mathcal{O}(\rho^{-N})$ solve lin sys $V^T w = \{\int_a^b x^k dx\}_{k=0}^{N-1}$ (Newton-Cotes)
 - better: "Gaussian" $\{x_j, w_j\}$ integrates deg. $2N-1$ exactly! err $\mathcal{O}(\rho^{-2N})$
- Adaptive quadrature (Gauss in each panel) excellent: codes quadgk, scipy, etc
- periodic case: $x_j = \frac{2\pi j}{N}$, $w_j = \frac{2\pi}{N}$ excellent "periodic trap. rule"
easy to check integrates e^{ikx} exactly for $|k| < N$, "Gaussian"
 f analytic in $|\operatorname{Im} x| < \alpha$ gives exp. conv. $\mathcal{O}(e^{-\alpha N})$, twice as good as interp!

Numerical integration (back to $d = 1$)

Task: eval. $\int_a^b f(x)dx$ accurately w/ least number of func. evals, N

"quadrature": nodes $\{x_j\}$, weights $\{w_j\}$, s.t. $\int_a^b f(x)dx \approx \sum_{j=1}^N w_j f(x_j)$

Idea: get interpolant \tilde{f} thru data $f(x_j) \rightarrow$ integrate that exactly

Examples:

"interpolatory quadrature"



- local piecewise linear \rightarrow composite trapezoid rule
 $w_j = h$ except $h/2$ at ends. low-order, err $\mathcal{O}(N^{-2})$, avoid!
- N -node global poly \rightarrow gives $\{w_j\}$ integrating degree $N-1$ exactly
 f analytic: err $\mathcal{O}(\rho^{-N})$ solve lin sys $V^T w = \{\int_a^b x^k dx\}_{k=0}^{N-1}$ (Newton-Cotes)
- better: "Gaussian" $\{x_j, w_j\}$ integrates deg. $2N-1$ exactly! err $\mathcal{O}(\rho^{-2N})$
Adaptive quadrature (Gauss in each panel) excellent: codes quadgk, scipy, etc
- periodic case: $x_j = \frac{2\pi j}{N}$, $w_j = \frac{2\pi}{N}$ excellent "periodic trap. rule"
easy to check integrates e^{ikx} exactly for $|k| < N$, "Gaussian"
 f analytic in $|\operatorname{Im} x| < \alpha$ gives exp. conv. $\mathcal{O}(e^{-\alpha N})$, twice as good as interp!

demo: $N=14$; $\text{sum}(\exp(\cos(2*\pi*(1:N)/N)))/N - \text{besseli}(0,1)$
ans = 1.3e-15

Advanced integration

- custom quadr. for singularity eg $f(x) = \text{smooth} \cdot |x|^{-1/2}$ (Rokhlin school)
or for arb. set of funcs. "generalized Gaussian quad." (CCM: Manas Rachh)
 - high-order end-corrections to uniform trap. rule (Alpert '99)
 - oscillatory functions: deform contour to \mathbb{C} "numerical steepest descent"
- ...

Advanced integration

- custom quadr. for singularity eg $f(x) = \text{smooth} \cdot |x|^{-1/2}$ (Rokhlin school)
or for arb. set of funcs. "generalized Gaussian quad." (CCM: Manas Rachh)
 - high-order end-corrections to uniform trap. rule (Alpert '99)
 - oscillatory functions: deform contour to \mathbb{C} "numerical steepest descent"
- ...

Higher dimensions $d > 1$

code: `integral2`, etc, `quadpy`

For $d \lesssim 5$, tensor product quadr. of 1D n -node grids in each dim

other coord systems: eg sphere can use tensor product in (θ, ϕ) . Or: iterate over dims.

adaptivity works: automatically refine boxes but soon enter research territory!

$\int_{\Omega} f(\mathbf{x}) d\mathbf{x}$ in nasty domain $\Omega \subset \mathbb{R}^d$? FEM meshing, blended conforming grids...

Advanced integration

- custom quadr. for singularity eg $f(x) = \text{smooth} \cdot |x|^{-1/2}$ (Rokhlin school)
or for arb. set of funcs. "generalized Gaussian quad." (CCM: Manas Rachh)
 - high-order end-corrections to uniform trap. rule (Alpert '99)
 - oscillatory functions: deform contour to \mathbb{C} "numerical steepest descent"
- ...

Higher dimensions $d > 1$

code: `integral2`, etc, `quadpy`

For $d \lesssim 5$, tensor product quadr. of 1D n -node grids in each dim

other coord systems: eg sphere can use tensor product in (θ, ϕ) . Or: iterate over dims.

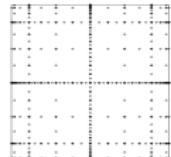
adaptivity works: automatically refine boxes but soon enter research territory!

$\int_{\Omega} f(\mathbf{x}) d\mathbf{x}$ in nasty domain $\Omega \subset \mathbb{R}^d$? FEM meshing, blended conforming grids...

Much higher $d \gg 1$

tensor prod: exp. # f evals. $N = n^d$ kills you :("curse of dim."

- "sparse grids" scale better as $N \sim n(\log n)^d$ (Smolyak '63)
- (quasi-)Monte Carlo: $\sum_{j=1}^N f(\mathbf{x}_j)$, for random \mathbf{x}_j err $\mathcal{O}(N^{-1/2})$, slow conv!
importance sampling (Thurs am session), custom transformations...



Numerical differentiation

Task: given ability to eval. $f(\mathbf{x})$ anywhere, how get $\nabla f(\mathbf{x})$? assume smooth

Numerical differentiation

Task: given ability to eval. $f(\mathbf{x})$ anywhere, how get $\nabla f(\mathbf{x})$? assume smooth

Finite differencing idea, 1D: $f'(\mathbf{x}) = \frac{f(\mathbf{x}+h) - f(\mathbf{x}-h)}{2h} + \mathcal{O}(h^2)$ Taylor's thm

"centered difference" formula

Want smallest error:

suggests taking $h \rightarrow 0$?

Let's see how that goes...

Numerical differentiation

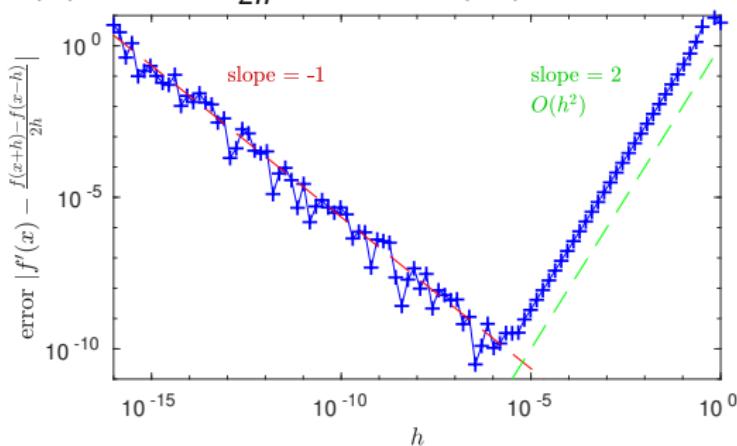
Task: given ability to eval. $f(\mathbf{x})$ anywhere, how get $\nabla f(\mathbf{x})$? assume smooth

Finite differencing idea, 1D: $f'(\mathbf{x}) = \frac{f(\mathbf{x}+h) - f(\mathbf{x}-h)}{2h} + \mathcal{O}(h^2)$ Taylor's thm

"centered difference" formula

Want smallest error:
suggests taking $h \rightarrow 0$?

Let's see how that goes...



- shrinking $\mathcal{O}(h^2)$ error gets swamped by a new growing error... what?

Numerical differentiation

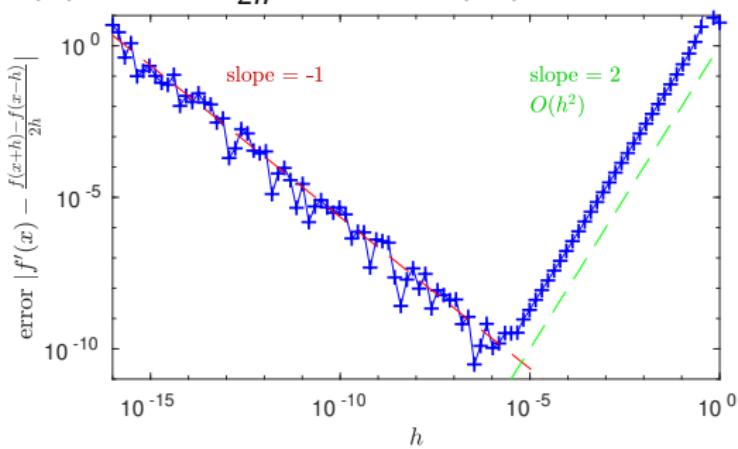
Task: given ability to eval. $f(\mathbf{x})$ anywhere, how get $\nabla f(\mathbf{x})$? assume smooth

Finite differencing idea, 1D: $f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2)$ Taylor's thm

"centered difference" formula

Want smallest error:
suggests taking $h \rightarrow 0$?

Let's see how that goes...



- shrinking $\mathcal{O}(h^2)$ error gets swamped by a new growing error... what?
- CPU arithmetic done only to relative "rounding error" $\epsilon_{\text{mach}} \sim 10^{-16}$
- subtracting v. close $f(x+h)$ and $f(x-h)$: "catastrophic cancellation"
- balance two error types: $h_{\text{best}} \sim \epsilon_{\text{mach}}^{1/3} \sim 10^{-5}$

Essential reading: floating point, backward stability [GC12, Ch. 5–6] [TBI97, Ch. 12–15]

High-order (better!) differentiation, $d = 1$

As w/ integration: get interpolant \rightarrow differentiate it exactly [Tre00, Ch. 6]

Get $N \times N$ matrix D acting on func. values $\{f(x_j)\}$ to give $\{f'(x_j)\}$. Has simple formula

High-order (better!) differentiation, $d = 1$

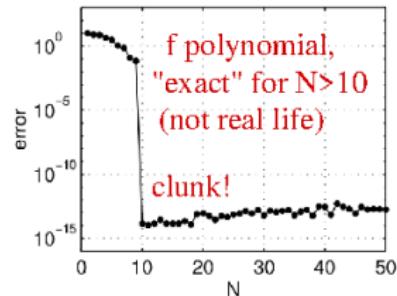
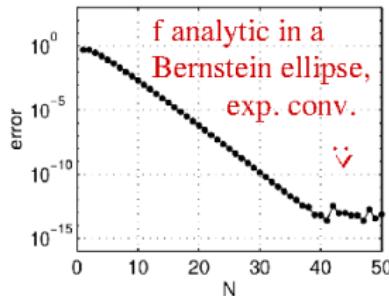
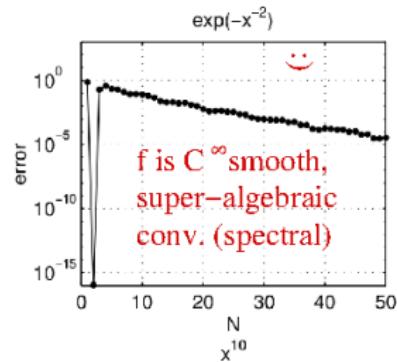
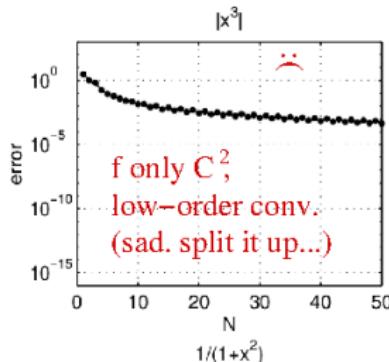
As w/ integration: get interpolant \rightarrow differentiate it exactly [Tre00, Ch. 6]

Get $N \times N$ matrix D acting on func. values $\{f(x_j)\}$ to give $\{f'(x_j)\}$. Has simple formula

Examples:

N Chebychev nodes
in $[-1, 1]$

shown: max error in f'



- for N large, the dense D is never formed, merely applied via FFT

spectral solvers for ODE/PDEs. codes: chebfun, PseudoPack, dedalus... Lecture II

Summary: we scratched the surface

Can integrate & differentiate smooth funcs given only point values $f(x_j)$

Both follow from building a good (fast-converging) interpolant

For f smooth in 1D, can & should easily get many (10+) digits accuracy

Concepts:

convergence order/rate how much effort will you have to spend to get more digits?

smoothness smooth \Leftrightarrow fast convergence; nonsmooth needs custom methods

global (one interpolation formula/basis for the whole domain)

vs local (distinct formulae for x in different regions)

spectral method global, converge v. fast, even non-per. can exploit FFT

adaptivity auto split boxes to put nodes only where they need to be

rounding error & catastrophic cancellation how not shoot self in the foot

tensor products for 2D, 3D for higher dims: randomized/NN/TN (Th/Fr sessions)

See recommended books at end, and come discuss stuff!

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*
 - Finite Volume Methods *Fluids*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*
 - Finite Volume Methods *Fluids*
 - “Traditional” Finite Elements *Mechanics*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*
 - Finite Volume Methods *Fluids*
 - “Traditional” Finite Elements *Mechanics*
 - “Modern” Finite Elements *Higher order*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*
 - Finite Volume Methods *Fluids*
 - “Traditional” Finite Elements *Mechanics*
 - “Modern” Finite Elements *Higher order*
 - Spectral Methods *Best accuracy for smooth solutions*

LECTURE II: numerical differential equations

Motivation

Produce numerical approximations to the solutions of ODEs/PDEs.

Goals for today

Basic overview of how different methods work. Understand error properties and suitability for different equations.

Families of methods:

- Finite Difference Methods *For time & space.*
- Finite Element Methods *Very general*
 - Finite Volume Methods *Fluids*
 - “Traditional” Finite Elements *Mechanics*
 - “Modern” Finite Elements *Higher order*
 - Spectral Methods *Best accuracy for smooth solutions*
- Boundary Integral Methods *Linear problems w/ boundary data*

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$
Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$

Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$

Others: local chemical/nuclear reactions ($\mathbf{u}(t)$ is vector of multiple components)

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$
Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$
Others: local chemical/nuclear reactions ($\mathbf{u}(t)$ is vector of multiple components)
- Time-independent PDEs: eg Poisson eqn $\Delta u(\mathbf{x}) = g(\mathbf{x})$
Task: solve $u(\mathbf{x})$ given forcing, boundary conditions
Steady state of heat/diffusion, Gauss's law for conservative forces
 $u(\mathbf{x})$ is chemical concentration, gravitational/electric potential
 Δu means Laplacian $\partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \dots =$ curvature of u
 $g(\mathbf{x})$ = volume source of chemical, mass or charge density

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$
Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$
Others: local chemical/nuclear reactions ($\mathbf{u}(t)$ is vector of multiple components)
- Time-independent PDEs: eg Poisson eqn $\Delta u(\mathbf{x}) = g(\mathbf{x})$
Task: solve $u(\mathbf{x})$ given forcing, boundary conditions
Steady state of heat/diffusion, Gauss's law for conservative forces
 $u(\mathbf{x})$ is chemical concentration, gravitational/electric potential
 Δu means Laplacian $\partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \dots =$ curvature of u
 $g(\mathbf{x})$ = volume source of chemical, mass or charge density
Others: Stokes eqn for velocity field \mathbf{u} in viscous fluid
Others: t-indep. Schrödinger eqn for quantum systems: $\Delta\psi = (V - E)\psi$

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$
Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$
Others: local chemical/nuclear reactions ($\mathbf{u}(t)$ is vector of multiple components)
- Time-independent PDEs: eg Poisson eqn $\Delta u(\mathbf{x}) = g(\mathbf{x})$
Task: solve $u(\mathbf{x})$ given forcing, boundary conditions
Steady state of heat/diffusion, Gauss's law for conservative forces
 $u(\mathbf{x})$ is chemical concentration, gravitational/electric potential
 Δu means Laplacian $\partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \dots =$ curvature of u
 $g(\mathbf{x})$ = volume source of chemical, mass or charge density
Others: Stokes eqn for velocity field \mathbf{u} in viscous fluid
Others: t-indep. Schrödinger eqn for quantum systems: $\Delta\psi = (V - E)\psi$
- Time-dependent PDEs: eg advection-diffusion $\partial_t c + \nabla \cdot (\mathbf{u}c) = \Delta c$
Task: solve $c(\mathbf{x}, t)$ given initial & boundary conditions
Others: Navier-Stokes, magnetohydrodynamics, ...

Reminder of types and applications of diff. eq.

- ODEs: eg pendulum $u''(t) + \sin(u(t)) = 0$

Task: solve $u(t)$ given initial conditions e.g. $u(0) = 1, u'(0) = 0$

Others: local chemical/nuclear reactions ($\mathbf{u}(t)$ is vector of multiple components)

- Time-independent PDEs: eg Poisson eqn $\Delta u(\mathbf{x}) = g(\mathbf{x})$

Task: solve $u(\mathbf{x})$ given forcing, boundary conditions

Steady state of heat/diffusion, Gauss's law for conservative forces

$u(\mathbf{x})$ is chemical concentration, gravitational/electric potential

Δu means Laplacian $\partial^2 u / \partial x^2 + \partial^2 u / \partial y^2 + \dots =$ curvature of u

$g(\mathbf{x})$ = volume source of chemical, mass or charge density

Others: Stokes eqn for velocity field \mathbf{u} in viscous fluid

Others: t-indep. Schrödinger eqn for quantum systems: $\Delta\psi = (V - E)\psi$

- Time-dependent PDEs: eg advection-diffusion $\partial_t c + \nabla \cdot (\mathbf{u}c) = \Delta c$

Task: solve $c(\mathbf{x}, t)$ given initial & boundary conditions

Others: Navier-Stokes, magnetohydrodynamics, ...

[Mike will overview the three flavors of PDE in next talk]

BCs: simple (eg periodic cube) vs complicated ($u = 0$ on a nasty surface)

Typical solution strategies

Time-independent PDEs:

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting algebraic system

Typical solution strategies

Time-independent PDEs:

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting algebraic system

Time-dependent PDEs: “method of lines”

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting coupled ODEs

Typical solution strategies

Time-independent PDEs:

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting algebraic system

Time-dependent PDEs: “method of lines”

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting coupled ODEs

ODEs:

- Treat spatial problems as time-indep. PDEs “boundary value problems”

Typical solution strategies

Time-independent PDEs:

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting algebraic system

Time-dependent PDEs: “method of lines”

- ① Discretize variables (grid points, cells, basis functions)
- ② Discretize operators/equations (derivatives)
- ③ Solve resulting coupled ODEs

ODEs:

- Treat spatial problems as time-indep. PDEs “boundary value problems”
- Evolve temporal problems with finite differences “initial value problems”

Finite difference methods

Basic viewpoint:

- Discretize variables on a discrete grid
- Construct Taylor-series approximations to values at neighboring points
- For N points, expand to N terms (error $\mathcal{O}(h^N)$)
- Eliminate to get approximation to d -th derivative (error $\mathcal{O}(h^{N-d})$)

Finite difference methods

Basic viewpoint:

- Discretize variables on a discrete grid
- Construct Taylor-series approximations to values at neighboring points
- For N points, expand to N terms (error $\mathcal{O}(h^N)$)
- Eliminate to get approximation to d -th derivative (error $\mathcal{O}(h^{N-d})$)

E.g. Centered differences on 3 points: $x - h, x, x + h$

$$u(x + h) = u(x) + u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (1)$$

$$u(x - h) = u(x) - u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (2)$$

Finite difference methods

Basic viewpoint:

- Discretize variables on a discrete grid
- Construct Taylor-series approximations to values at neighboring points
- For N points, expand to N terms (error $\mathcal{O}(h^N)$)
- Eliminate to get approximation to d -th derivative (error $\mathcal{O}(h^{N-d})$)

E.g. Centered differences on 3 points: $x - h, x, x + h$

$$u(x + h) = u(x) + u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (1)$$

$$u(x - h) = u(x) - u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (2)$$

To approximate $u'(x)$, subtract to eliminate $u''(x)$:

$$u'(x) = \frac{u(x + h) - u(x - h)}{2h} + \mathcal{O}(h^2) \quad (3)$$

Finite difference methods

Basic viewpoint:

- Discretize variables on a discrete grid
- Construct Taylor-series approximations to values at neighboring points
- For N points, expand to N terms (error $\mathcal{O}(h^N)$)
- Eliminate to get approximation to d -th derivative (error $\mathcal{O}(h^{N-d})$)

E.g. Centered differences on 3 points: $x - h, x, x + h$

$$u(x + h) = u(x) + u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (1)$$

$$u(x - h) = u(x) - u'(x)h + u''(x)h^2/2 + \mathcal{O}(h^3) \quad (2)$$

To approximate $u'(x)$, subtract to eliminate $u''(x)$:

$$u'(x) = \frac{u(x + h) - u(x - h)}{2h} + \mathcal{O}(h^2) \quad (3)$$

To approximate $u''(x)$, add to eliminate $u'(x)$:

$$u''(x) = \frac{u(x + h) - 2u(x) + u(x - h)}{h^2} + \mathcal{O}(h^2) \quad (4)$$

Extra order here due to symmetry

Finite difference methods

Alternate viewpoint:

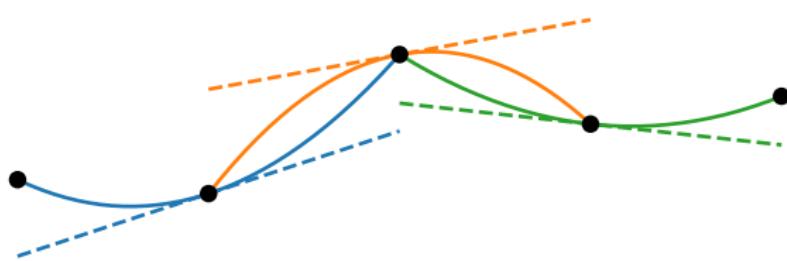
- Discretize variables on a discrete grid
- Construct interpolating polynomial of N nearest points.
 Unique, degree $N-1$.
- Differentiate the local interpolant to approximate derivatives.

Finite difference methods

Alternate viewpoint:

- Discretize variables on a discrete grid
- Construct interpolating polynomial of N nearest points.
Unique, degree $N-1$.
- Differentiate the local interpolant to approximate derivatives.

E.g. Centered differences using 3 points:



Implicit & Explicit Timestepping

Consider temporal ODE $u'(t) = f(u(t))$.

Timesteppers solve using finite differences to advance $u_n \rightarrow u_{n+1}$

Implicit & Explicit Timestepping

Consider temporal ODE $u'(t) = f(u(t))$.

Timesteppers solve using finite differences to advance $u_n \rightarrow u_{n+1}$

Explicit schemes: just need $f(u_n)$. Simple but unstable for large steps

- E.g. forward Euler: use 1st-order forward difference

$$u'(t) = -\lambda u(t) \quad \lambda > 0 \quad (5)$$

$$u_{n+1} - u_n = -k\lambda u_n \quad (6)$$

$$u_{n+1} = (1 - k\lambda)u_n \quad (7)$$

$k\lambda < 2$ for stability

Implicit & Explicit Timestepping

Consider temporal ODE $u'(t) = f(u(t))$.

Timesteppers solve using finite differences to advance $u_n \rightarrow u_{n+1}$

Explicit schemes: just need $f(u_n)$. Simple but unstable for large steps

- E.g. forward Euler: use 1st-order forward difference

$$u'(t) = -\lambda u(t) \quad \lambda > 0 \quad (5)$$

$$u_{n+1} - u_n = -k\lambda u_n \quad (6)$$

$$u_{n+1} = (1 - k\lambda)u_n \quad (7)$$

$k\lambda < 2$ for stability

Implicit schemes: require solving $f(u^{n+1}) = \dots$ Stable but expensive

- E.g. backward Euler: use 1st-order backward difference

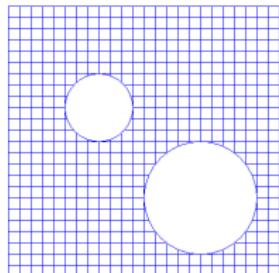
$$u'(t) = -\lambda u(t) \quad \lambda > 0 \quad (8)$$

$$u_{n+1} - u_n = -k\lambda u_{n+1} \quad (9)$$

$$u_{n+1} = (1 + k\lambda)^{-1}u_n \quad (10)$$

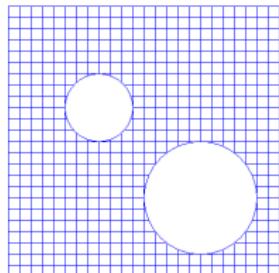
Finite difference methods

- Extends to multiple dimensions with regular grids
- Simple to adjust order of accuracy
- Restricted to simple geometries / regular grids
- Some more advanced techniques:
 - Conservative schemes
 - Select stencils term by term “upwinding”
 - Adaptive stencil selection for jumps “WENO”



Finite difference methods

- Extends to multiple dimensions with regular grids
- Simple to adjust order of accuracy
- Restricted to simple geometries / regular grids
- Some more advanced techniques:
 - Conservative schemes
 - Select stencils term by term “upwinding”
 - Adaptive stencil selection for jumps “WENO”

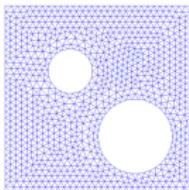


Resources: LeVeque “Finite Difference Methods for ODE/PDE”

Codes: e.g. Pencil code (magnetohydrodynamics)

Finite element methods

Partition domain into elements, represent unknown within each element using basis functions (usually polynomials)



Convert equations to weak form

- Give example
- Weak form advantages: lowers order, works if lower derivs are integrable (good for discontinuities, hyperbolic), can use tent functions
Replace weak form with Galerkin/weighted-residual form to get algebraic system

Traditional FEM methods use tent functions, produce continuous solutions. Fluxes at cell boundaries can be computed from internal representation, but can be discontinuous (non conservative).

Finite volume methods

Case of FEM taking functions to be piecewise constant inside elements

Resulting system is exactly conservative (good for hyperbolic), but requires Riemann solve

- Order increased by using neighbors to reconstruct higher order internal representations or fluxes (slope/flux reconstruction)
- Reconstruction is nonlinear to control oscillations around discontinuities (TVD, ENO/WENO).

Very common in CFD, CCA

Advanced finite element methods

- Discontinuous Galerkin: allow discontinuities, need Riemann solvers again
- Spectral elements: move towards large p for better internal representations

Resources: ...

Codes: ...

Spectral methods

Limit of very few elements with very large p : exponential accuracy for smooth problems

Traditional techniques: Fourier spectral methods.

- Fast due to FFT: optimal complexity with exponential accuracy
- Limited to simple geometries / equations with symmetries

Polynomial spectral methods

- More flexible in terms of equations
- Still limited to simple geometries: cubes, spheres, cylinders, etc.
- Still a weak method, but don't integrate by parts. Apply Galerkin directly.

Modern research: sparse methods for arbitrary equations.

Resources: ...

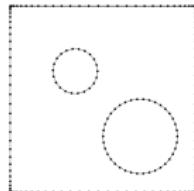
Codes: ...

Boundary integral methods

Use knowledge of PDEs in constructing numerical solutions:

A Green's func G (fundamental soln) needed

eg $\Delta G = \delta$



For linear PDEs dominated by boundary rather than bulk terms, compute solution by forming integral equation of the fundamental solution/Green's function. Reduced dimensionality. Improved conditioning. Low-rank interactions and fast methods. Reconstruct solution in the bulk.

Examples: Stokes flow, Helmholtz, Maxwell, typically with homogeneous media

heavily uses high-order integration methods

- see Jun Wang's talk later today!

Summary

Recommended accessible reading

- [GC12] A Greenbaum and T P Chartier, *Numerical methods*, Princeton University Press, 2012.
- [TBI97] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, SIAM, 1997.
- [Tre00] Lloyd N. Trefethen, *Spectral methods in MATLAB*, Software, Environments, and Tools, vol. 10, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.
- [Tre13] L. N. Trefethen, *Approximation theory and approximation practice*, SIAM, 2013,
<http://www.maths.ox.ac.uk/chebfun/ATAP>.

This document: <https://github.com/ahbarnett/fwam-numpde>

See code directory for MATLAB code used to generate figures