Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Outline

Organization

- Time and location: Mondays and Wednesdays 4:55–6:10PM, WWH 1302
- Office hours: Mondays, 6.10 – 7.10pm, stop by or make an appointment (please email). My office number is WWH #421
- Course webpage: https://docs.google.com/document/d/1VxdM4s-wiV-_C4uBDrP4ioiOdscfLwim8o1mokuj2oQ/edit?usp=sharing

  You need to be logged in with your NYU-Google account to access it
- Brightspace https://brightspace.nyu.edu/d2l/home/400947

Required work and grading:

- 7 homework assignments (60% of your grade).

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.
  - Follow the guidelines/rules provided with the homework to present and submit your solution.

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.
  - Follow the guidelines/rules provided with the homework to present and submit your solution.
  - If you find errors/typos in the slides or homework assignments, please let me know.

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.
  - Follow the guidelines/rules provided with the homework to present and submit your solution.
  - If you find errors/typos in the slides or homework assignments, please let me know.
  - Late homework policy: We understand that there are sometimes unexpected circumstances. You can hand in one homework late by 24h this semester if you send me and the grader an email at least 24h before the deadline that your work will be late.

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.
  - Follow the guidelines/rules provided with the homework to present and submit your solution.
  - If you find errors/typos in the slides or homework assignments, please let me know.
  - Late homework policy: We understand that there are sometimes unexpected circumstances. You can hand in one homework late by 24h this semester if you send me and the grader an email at least 24h before the deadline that your work will be late.
- A final (40% of grade)

Required work and grading:

- 7 homework assignments (60% of your grade).
  - These will be mixed paper&pencil and computational/programming. You hand in solutions in LaTeX and Matlab code
  - Grader is Qi (Winston) Liang, lq504@nyu.edu
  - Grader's office hour: tbd
  - You are welcome to discuss with your colleagues, but *you've to write up your solution independently and write every line of code yourself.*
  - The first homework assignment has been posted.
  - Follow the guidelines/rules provided with the homework to present and submit your solution.
  - If you find errors/typos in the slides or homework assignments, please let me know.
  - Late homework policy: We understand that there are sometimes unexpected circumstances. You can hand in one homework late by 24h this semester if you send me and the grader an email at least 24h before the deadline that your work will be late.
- A final (40% of grade)
  - Expect to write *some* code too

Organization issues

Prerequisites:

- Basic linear algebra; calculus; experience in Matlab (or Python or another programming language)

There is a part II of this class...

- ...in the Spring semester. You should take both parts to get a reasonably complete overview of Numerical Methods.
- If you consider taking only one semester of Numerical Methods, I recommend taking Scientific Computing this semester instead of this class.

Topics covered in Numerical Methods I

Numerical Methods and their Analysis

- Stability; sources of errors; error propagation, representation of numbers in computers
- Numerical linear algebra: direct solution of sparse/dense linear systems; solution of least square systems; eigenvalue problems; iterative solution of linear systems
- Nonlinear systems; Newton's method; Nonlinear least squares
- Numerical optimization
- Interpolation and Approximation
- Numerical integration

Computing Issues

- What makes some computer codes faster than others?
- Where are numerical methods used, and what is their role in science research?
- How large/complicated problems can we solve today? Where are the challenges and limits of what we can do?

Topics of Numerical Methods II

Main topics covered in Numerical Methods II in the Spring semester

- Approximation of ordinary differential equations (ODEs)
- Approximation of partial differential equations (PDEs)
- Solvers for the resulting (high-dimensional) discrete problems

Programming

Programming the methods we discuss is an integral part of this course. To really understand methods & algorithms, one needs to implement them and experiment with them.

- Make sure you have access to MATLAB (CIMS, student license), you will need it for the first homework assignment.
- Alternatives to MATLAB: Octave, Python or Julia.
- We will talk about a few best coding practices, and how to present results.

Recommended textbooks/literature:

Text books:

- P. Deuflhard, A. Hohmann: *Numerical Analysis in Modern Scientific Computing. An Introduction*, 2nd edition, Springer, 2003.
- L. N. Trefethen, D. Bau: *Numerical Linear Algebra*, SIAM, 1997.
- A. Quarteroni, R. Sacco, F. Saleri: *Numerical Mathematics*, 2nd edition, Springer, 2007.
- M. Overton: Numerical Computing with IEEE Floating Point Arithmetic, SIAM, 2004.

Matlab/Programming:

- W. Gander, M. J. Gander, F. Kwok: *Scientific Computing - An Introduction Using Maple and MATLAB*. Texts in Computation Science and Engineering. Springer, 2014.
- C. Moler: *Numerical Computing with Matlab*, SIAM, 2007.

Numerical mathematics

Computer simulations have had a big influence on research and development; sometimes the ability to simulate phenomena is referred to as the third pillar of science.

Numerical mathematics is a part of mathematics that develops, analyzes and applies methods from scientific computing to

- analysis
- linear algebra
- optimization
- differential equations
- ...

It has applications accross many applied sciences, including:

- physics
- economics
- biology
- finance
- ...

Development of Numerical Methods at Courant

A few examples...

- Eigenvalue problems (Overton)
- Fast multipole method (Greengard, O'Neil, Zorin)
- Finite elements and contact problems (Panozzo, Zorin)
- Methods for studying dynamical systems, multiscale methods (Vanden-Eijnden)
- Methods for free boundary problems in fluid dynamics (Shelley)
- Scalable implicit solvers for viscous flows (Stadler)
- Sampling methods and Uncertainty Quantification (Goodman, Stadler, Peherstorfer)
- Scientific machine learning (Vanden-Eijnden, Stadler, Peherstorfer)
- ...

Applications of Numerical Methods at Courant

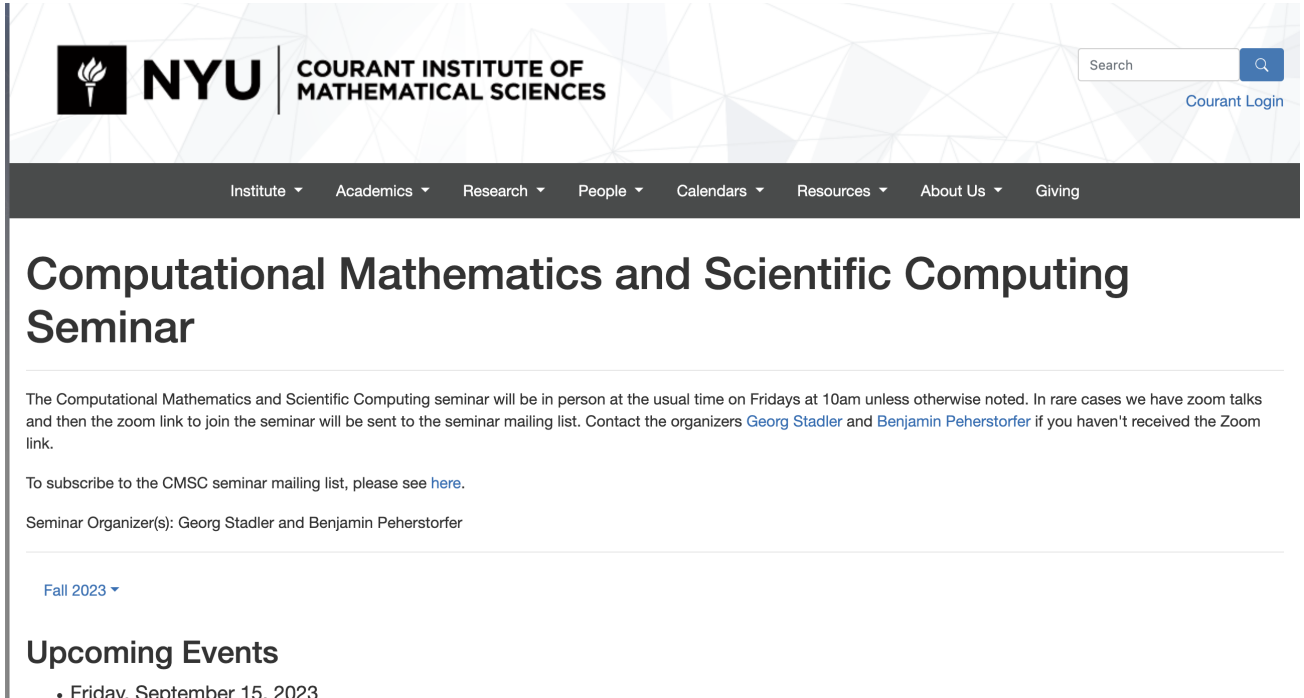
A few examples. . .

- Simulation and analysis of natural and artificial heart valves (Peskin)
- Simulation of plate tectonics and mantle convection (Stadler)
- The physics of cell's interiors and their motion (Shelley)
- Optimal complexity wave simulations (Greengard)
- Simulation of blood cells-resolving blood flow (Zorin)
- Plasmas (Stadler, Kaptanoglu)
- . . .

Seminars

Computational Mathematics and Scientific Computing seminar

- Fridays at 10:00, WWH 1302
- Talks about current research
- https://cims.nyu.edu/dynamic/calendars/seminars/computational-mathematics-and-scientific-computing-seminar/



Modeling and Simulation meeting

- Thursdays at 12:30, WWH 1302
- Student-driven meeting on topics related to computational mathematics
- https://math.nyu.edu/dynamic/research/pages/research-and-training-group-mathematical-modeling-and-simulation/activities/group-meeting/

Modeling and Simulation meeting

- Thursdays at 12:30, WWH 1302
- Student-driven meeting on topics related to computational mathematics
- https://math.nyu.edu/dynamic/research/pages/research-and-training-group-mathematical-modeling-and-simulation/activities/group-meeting/

Mathematics colloquium

- Mondays at 3:45, WWH 1302
- https://math.nyu.edu/dynamic/calendars/seminars/mathematics-colloquium/

Modeling and Simulation meeting

- Thursdays at 12:30, WWH 1302
- Student-driven meeting on topics related to computational mathematics
- https://math.nyu.edu/dynamic/research/pages/research-and-training-group-mathematical-modeling-and-simulation/activities/group-meeting/

Mathematics colloquium

- Mondays at 3:45, WWH 1302
- https://math.nyu.edu/dynamic/calendars/seminars/mathematics-colloquium/

Math and data

- Thursdays at 2.00, Auditorium Hall 150, Center for Data Science, NYU, 60 5th ave.
- Interface of Applied Mathematics, Statistics and Machine Learning
- https://mad.cds.nyu.edu/seminar/

Outline

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation?

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

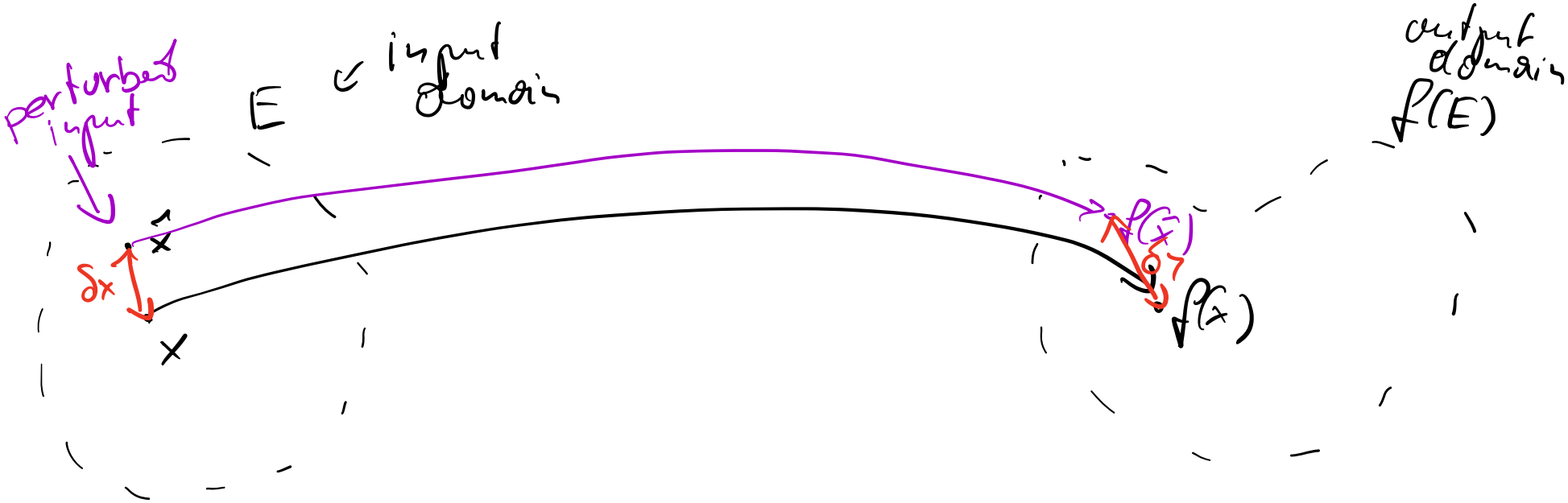
$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation? $\rightsquigarrow$ change the problem)

Condition of a problem (visualization)



Notice: $\delta y = f(\hat{x}) - f(x)$

is independent of any algorithm that we use. There is no numerics here!

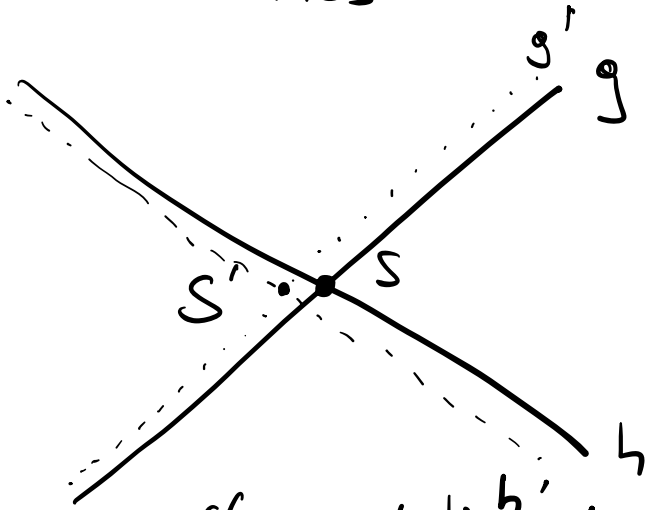
Condition of a problem (intersecting lines)

Consider the problem of determining the intersection point of two lines

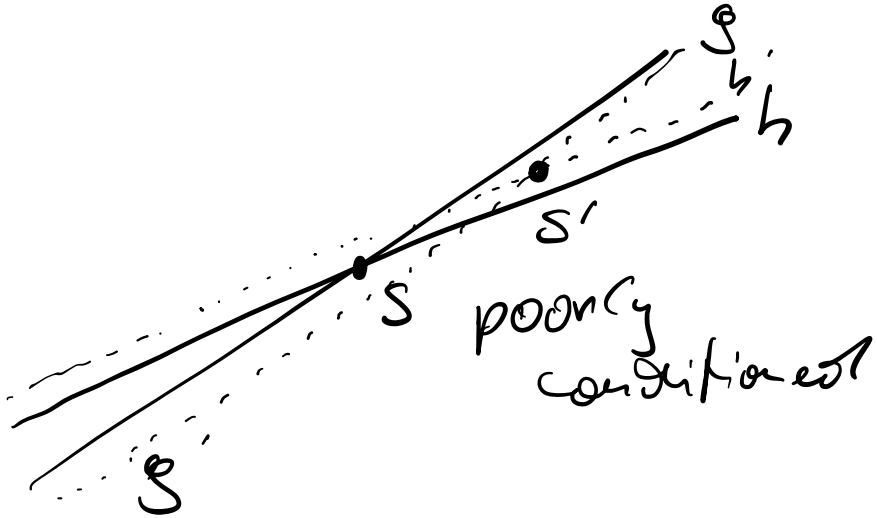
$$f : (g, h) \mapsto s$$

lines

intersection point



well-conditioned



poorly conditioned

Condition of a problem (cont'd)

- Terms such as "little bit" and a "small amount" already point to that we need to measure something
- Therefore, we assume the map f is given as

$$f : U \subset \mathbb{R}^n \to \mathbb{R}^m$$

  and we are interested in the norm $\|\cdot\|$
- The input error is then

$$\|x - \hat{x}\| \leq \delta \text{ (absolute)} \qquad \|x - \hat{x}\| \leq \delta \|x\| \text{ (relative)}$$

- Correspondingly we measure the output error $f(x) - f(\hat{x})$ in $\|\cdot\|$ (we could also have looked at a componentwise error)

Condition of a problem (cont'd)

- Absolute condition number at x is

$$\kappa_{\text{abs}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\|}{\|x - \hat{x}\|}$$

- Relative condition number at x is

$$\kappa_{\text{rel}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\| / \|f(x)\|}{\|x - \hat{x}\| / \|x\|}$$

- If f is differentiable in x, then

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\| \,,$$

where $\|f'(x)\|$ is the norm of the Jacobian $f'(x)$ in the operator norm

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

- Another way to interpret the condition number at x is via the bounds

$$\|f(\hat{x}) - f(x)\| \leq \kappa_{\text{abs}} \|\hat{x} - x\|$$

and

$$\frac{\|f(\hat{x}) - f(x)\|}{\|f(x)\|} \leq \kappa_{\text{rel}} \frac{\|\hat{x} - x\|}{\|x\|},$$

for infinitesimal δ (or $\hat{x} \to x$)

Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$,

Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$,

Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$, then the problem is poorly conditioned: Small relative input error can lead to large relative output error
- If κ_{rel} (and κ_{abs}) do not exist, then the problem is ill conditioned.

- What is poorly conditioned depends on desired accuracy: if the input accuracy is low but we expect a high output accuracy, then problems are quickly poorly conditioned. If we are happy with a less accurate output, we might consider the problem still well conditioned.
- Sometimes, the possibly large error in the output does not matter and so we can solve poorly conditioned problems (think of early design stages, rapid prototyping, etc); but we should be very much aware of the condition of the problem.

Condition of a problem: Example

Condition number of „addition"

linear map

$$f: \mathbb{R}^2 \to \mathbb{R}, \quad (a,b) \mapsto f(a,b) = a+b$$

derivative

$$f'(a,b) = [1, 1] \in \mathbb{R}^{1\times 2}$$

choose 1-norm $\|\cdot\|_1$ on $\mathbb{R}^2$

$$\left\| \begin{bmatrix} a \\ b \end{bmatrix} \right\|_1 = |a| + |b|$$

Induced operator norm is max column sum

$$\| f'(a,b) \|_1 = \| [1, 1] \|_1 = 1$$

$$\boxed{\begin{aligned} k_{abs} &= \| f'(a,b) \|_1 = 1 \\ k_{rel} &= \frac{\|x\|}{\|f(x)\|} \| f'(x) \| = \\ &= \frac{|a| + |b|}{|a+b|} \cdot 1 \end{aligned}}$$

a, b have the same sign

if $a \cdot b \geq 0$, $k_{rel} = 1$ $\rightarrow$ perfect

a, b don't have same sign

$a \cdot b \leq 0$ $\rightarrow$ subtraction

set $a > 0$, $b = -(a + \varepsilon)$, $0 < \varepsilon \ll 1$

$$k_{rel} = \frac{a + a + \varepsilon}{|a - (a + \varepsilon)|} = \frac{2a + \varepsilon}{\varepsilon} \gg 1$$

$\rightsquigarrow$ poorly conditioned

Condition number of Mat Vec: $x \mapsto Ax$

$$f(x) = Ax$$
$$f'(x) = A$$

$$\kappa_{abs} = \|f'(x)\| = \|A\| = \sup_{\|x\|=1} \|Ax\|$$

$$\kappa_{rel} = \frac{\|x\|}{\|Ax\|} \|A\|$$

Norm plays a critical role!

Norm of $\|A\|$ is important

Example: take $\|\cdot\|_2$ on input x, then operator norm

$$\|A\|_2 = \sigma_{max}(A)$$

max singular value

singular values tell us by how much a vector gets 'stretched' by matrix

perturbation

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Condition of problems

Today

- More on condition of problems
- Stability of algorithms
- Matlab recap

Announcements

- Homework 1 was posted last week; is due in two weeks Mon, Sep 23 *before class*

Recap: Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation?

Recap: Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation? $\rightsquigarrow$ change the problem)

Recap: Condition of a problem (cont'd)

- Absolute condition number at x is

$$\kappa_{\text{abs}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\|}{\|x - \hat{x}\|}$$

- Relative condition number at x is

$$\kappa_{\text{rel}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\| / \|f(x)\|}{\|x - \hat{x}\| / \|x\|}$$

- If f is differentiable in x, then

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\| \,,$$

where $\|f'(x)\|$ is the norm of the Jacobian $f'(x)$ in the operator norm

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$,

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$,

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$, then the problem is poorly conditioned: Small relative input error can lead to large relative output error
- If κ_{rel} (and κ_{abs}) do not exist, then the problem is ill conditioned.

- What is poorly conditioned depends on desired accuracy: if the input accuracy is low but we expect a high output accuracy, then problems are quickly poorly conditioned. If we are happy with a less accurate output, we might consider the problem still well conditioned.
- Sometimes, the possibly large error in the output does not matter and so we can solve poorly conditioned problems (think of early design stages, rapid prototyping, etc); but we should be very much aware of the condition of the problem.

Condition of a problem: Example

Condition number of MatVec: $x \mapsto Ax$

$$f(x) = Ax$$

$$f'(x) = A$$

$$k_{abs} = \|f'(x)\| = \|A\|$$

$$k_{rel} = \frac{\|x\|}{\|Ax\|} \|A\|$$

norm plays a critical role!

if we take $\|\cdot\|_2$ on input x, then operator norm

$$\|A\|_2 = \sigma_{max}(A)$$

Singular values tell us by how much a vector gets "stretched" by matrix

perturbation

Condition number of solving linear systems

$$Ax = b$$

(1) Problem: fix A, consider $b \mapsto A^{-1}b$

$$f : \mathbb{R}^n \to \mathbb{R}^n, \quad f(b) = A^{-1}b$$

differentiable

$$k_{abs} = \|f'(b)\| = \|A^{-1}\|$$

$$k_{rel} = \frac{\|b\|}{\|A^{-1}b\|} \|A^{-1}\|$$

if $A^{-1}b = x$

$$k_{rel} = \frac{\|Ax\|}{\|x\|} \|A^{-1}\| \leq \frac{\|A\| \, \|x\|}{\|x\|} \cdot \|A^{-1}\|$$

$$= \|A\| \, \|A^{-1}\| = k(A)$$

"condition number of matrix A"

If we take $\|\cdot\|_2$ on input b, then operator norm

$$\|A^{-1}\|_2 = \sigma_{max}(A^{-1}) = \frac{1}{\sigma_{min}(A)}$$

then $$\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$$

---

(2) Problem: fix b, consider $A \mapsto A^{-1}b$

$f(A) = A^{-1}b$ $\rightsquigarrow$ textbook by Deuflhard

is differentiable

$$\|f'(A)\| \leq \|A^{-1}\| \, \|x\|$$

and $x = A^{-1}b$

$$\kappa_{abs} = \|f'(A)\| \leq \|A^{-1}\| \, \|x\|$$

$$\kappa_{rel} = \frac{\|A\|}{\|f(A)\|} \|f'(A)\|$$

$$\leq \frac{\|A\|}{\cancel{\|x\|}} \|A^{-1}\| \, \cancel{\|x\|} = \|A\| \cdot \|A^{-1}\| = \kappa(A)$$

Condition number $\kappa(A)$ of matrix A, tells us how well system of linear equations $Ax = b$ is conditioned.

Outline

Stability of algorithms

Is $\tilde{f}(x)$, computed with an algorithm $\tilde{f}$, a good approximation of $f(x)$?

We are happy if the error due to the algorithm

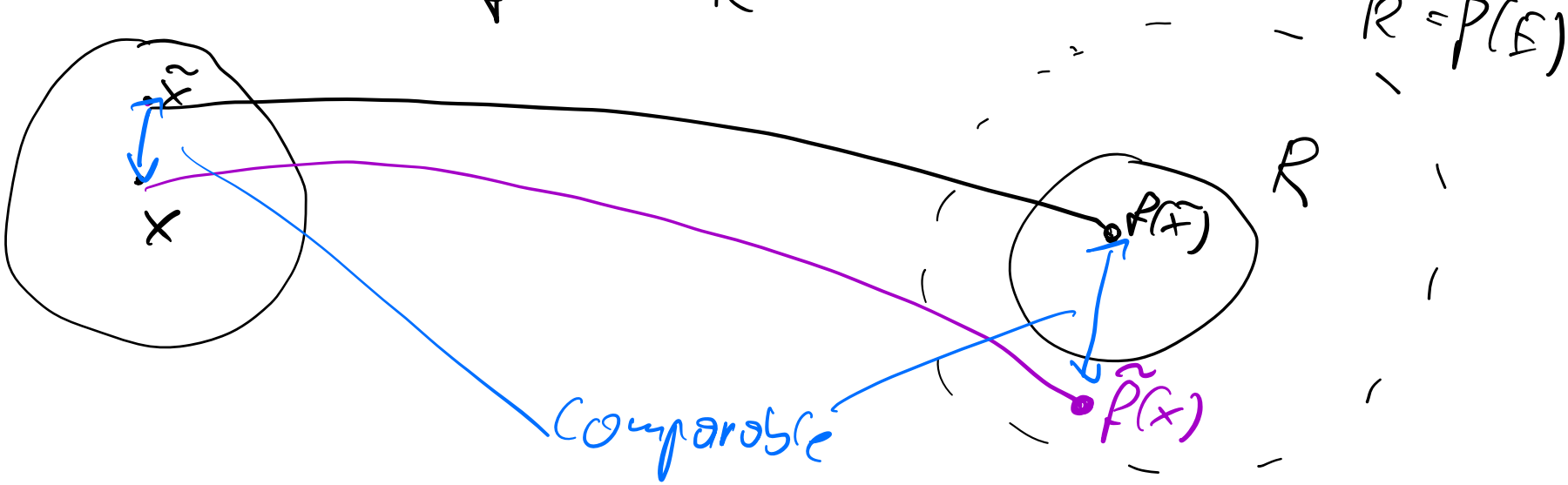
$$\tilde{f}(x) - f(x)$$

lies within reasonable bounds of the error due to the input

$$f(\tilde{x}) - f(x)$$

$f : E \to R$

$\tilde{f} : E \to \tilde{R}$



Stability: Stability

We say that an algorithm $\tilde{f}$ for a problem f is stable if for each $x \in E$ the error

$$\frac{\|\tilde{f}(x) - f(\tilde{x})\|}{\|f(\tilde{x})\|}$$

is small for $\tilde{x}$ with small

$$\frac{\|\tilde{x} - x\|}{\|x\|}$$

> *A stable algorithm gives nearly the right answer ($\tilde{f}(x)$) to nearly the right question ($f(\tilde{x})$).*

In forward error analysis one tries to establish stability by showing error bounds on the result in each operation in the algorithm in order to bound the error in the end result

Stability: Backward stability

Backward stability: Pass the errors of the algorithm back and interpret as input errors.

An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

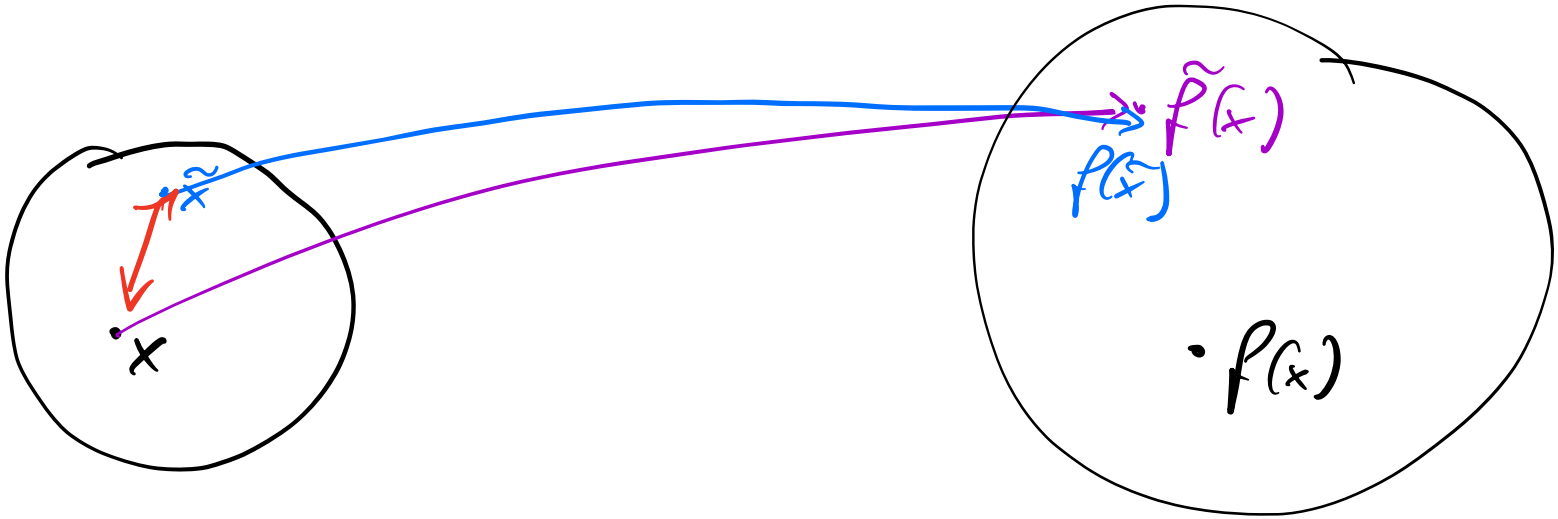
$$\frac{\|\tilde{x} - x\|}{\|x\|}$$

small

This is a tightening of the definition of stability of the previous slide:

> *A backward stable algorithm gives exactly the right answer to nearly the right question.*

In backward error analysis one calculates, for a given output, how much one would need to perturb the input in order for the answer to be exact.



algorithmic error is interpreted as input error

Errors and error analyses

Relative errors:

$$\frac{\|x - x_n\|}{\|x\|}$$

Absolute error:

$$\|x - x_n\|$$

- Used for theoretical arguments
- In numerical practice: exact solution is not available, so these errors must be approximated.

A priori analysis is performed before a specific solution is computed. Typically, the analysis is performed for a large class of possible inputs.
A posteriori analysis bounds the error for a specific numerical solution $\hat{x}$ (computed with a specific numerical method), and uses, e.g., residuals for the a posteriori analysis.

Computational errors

Numerical algorithms try to control or minimize, rather then eliminate, the various computational errors:

- **Approximation error** due to replacing the computational problem with an easier-to-solve approximation. Also called discretization error for ODEs/PDEs.
- **Truncation error** due to replacing limits and infinite sequences and sums by a finite number of steps. Closely related to approximation error.
- **Roundoff error** due to finite representation of real numbers and arithmetic on the computer, $x \neq \hat{x}$.
- **Propagated error** due to errors in the data from user input or previous calculations in iterative methods.
- **Statistical error** in stochastic calculations such as Monte Carlo calculations.

Intuition: Stability, Consistency, Convergence

Instead of solving $F(x, y) = 0$ directly, many numerical methods generate a solution sequentially

$$\bar{F}(x_i, x_{i-1}) = 0\,, \qquad i = 1, 2, 3, \dots\,,$$

with $x_0 = x$ and sequence (x_i) converging to y

Additionally, we use a numerical method $\hat{F}_n$ instead of $\bar{F}$

$$\hat{F}_n(\hat{x}_i, \hat{x}_{i-1}) = 0\,, \qquad i = 1, 2, 3, \dots\,,$$

with method $\hat{F}_n$ depending on a parameter n: Increasing n typically means investing more computational time for a hopefully more accurate result

Consistent: A numerical method is consistent if the local error made at each step vanishes for $n \to \infty$

$$\hat{F}_n(x_i, x_{i-1}) \to \bar{F}(x_i, x_{i-1}) \qquad (n \to \infty)$$

This is one of the most basic requirements that we have on a numerical approach. If it is not consistent, it means we can invest more computational time (more effort) and certainly won't get lower errors.

Stability: Because we use $\hat{F}_n$ instead of $\bar{F}$, in each iteration we make a local error (see above). We have $\hat{x}_i$ at iteration i rather than x_i. Stability means here that the local error can be amplified only by a constant that is independent of n.

Convergence: If the numerical error can be made arbitrarily small by increasing the computational effort $n \to \infty$

$$\boxed{\text{consistency} + \text{stability} \to \text{convergence}}$$

A concrete and formal description of these concepts for finite difference approximations can be found in Chapter 2 of LeVeque's textbook on finite difference methods.

Speed of convergence

Let $x_n \to x$ in a normed space $X, \|\cdot\|$ for $n \to \infty$.

$$\lim_{n\to\infty} \frac{\|x - x_{n+1}\|}{\|x - x_n\|^q} < C$$

with $C > 0$ and $q \geq 1$

- Linear convergence: $q = 1$ and $C < 1$

$$\|x - x_{n+1}\| \leq C\|x - x_n\|$$

- Quadratic convergence: $q = 2$

$$\|x - x_{n+1}\| \leq C\|x - x_n\|^2$$

Beyond convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:

Beyond convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:
- **Accuracy** How much computational work do you need to expand to get an answer to a desired relative error?

Beyond convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:
- **Accuracy** How much computational work do you need to expand to get an answer to a desired relative error?
- **Robustness** Does the algorithm work (equally) well for all (reasonable) input data d?

Beyond convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:
- **Accuracy** How much computational work do you need to expand to get an answer to a desired relative error?
- **Robustness** Does the algorithm work (equally) well for all (reasonable) input data d?
- **Efficiency** How fast does the implementation produce the answer? This depends on the algorithm, on the computer, the programming language, the programmer, etc. (more next class)

Beyond convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:
- **Accuracy** How much computational work do you need to expand to get an answer to a desired relative error?
- **Robustness** Does the algorithm work (equally) well for all (reasonable) input data d?
- **Efficiency** How fast does the implementation produce the answer? This depends on the algorithm, on the computer, the programming language, the programmer, etc. (more next class)
- **Difficulty** How easy is it to implement and apply in practice? Do I need to spend 5 years of my time to implement it or can I code it up in 2 lines of code?

Matlab peculiarities [Following slides: A. Donev]

- MATLAB is an **interpreted language**, meaning that commands are interpreted and executed as encountered. MATLAB caches some stuff though...
- Many of MATLAB's **intrinsic routines** are however compiled and optimized and often based on well-known libraries (BLAS, LAPACK, FFTW, etc.).
- Variables in scripts/worspace are global and persist throughout an interactive session (use *whos* for info and *clear* to clear workspace).
- Every variable in MATLAB is, unless specifically arranged otherwise, a matrix, **double precision float** if numerical.
- Vectors (column or row) are also matrices for which one of the dimensions is 1.
- **Complex arithmetic** and complex matrices are used where necessary.

Matrices [Slide: A. Donev]

```
>> format compact; format long
>> x=-1; % A scalar that is really a 1x1 matrix
>> whos('x')
  Name      Size            Bytes  Class     Attributes
  x         1x1                 8  double

>> y=sqrt(x) % Requires complex arithmetic
y =                   0 + 1.000000000000000i
>> whos('y')
  Name      Size            Bytes  Class     Attributes
  y         1x1                16  double    complex

>> size(x)
ans =      1      1
>> x(1)
ans =     -1
>> x(1,1)
ans =     -1
>> x(3)=1;
>> x
x =     -1      0      1
```

Vectorization/Optimization [Slide: A. Donev]

- MATLAB uses **dynamic memory management** (including garbage collection), and matrices are re-allocated as needed when new elements are added.
- It is however much better to **pre-allocate space** ahead of time using, for example, *zeros*.
- The **colon notation** is very important in accessing array sections, and x is different from $x(:)$.
- **Avoid for loops** unless necessary: Use array notation and intrinsic functions instead.
- To see how much CPU (computing) time a section of code took, use *tic* and *toc* (but beware of timing small sections of code).
- MATLAB has built-in **profiling tools** (*help profile*).

Pre-allocation [Slide: A. Donev]

```
format compact; format long
clear; % Clear all variables from memory

N=100000; % The number of iterations

% Try commenting this line out:
f=zeros(1,N); % Pre-allocate f

tic;
f(1)=1;
for i=2:N
    f(i)=f(i-1)+i;
end
elapsed=toc;

fprintf('The result is f(%d)=%g, computed in %g s\n', ...
        N, f(N), elapsed);
```

Vectorization [Slide: A. Donev]

```
function vect(vectorize)
   N=1000000; % The number of elements
   x=linspace(0,1,N); % Grid of N equi-spaced points

   tic;
   if(vectorize) % Vectorized
      x=sqrt(x);
   else % Non-vectorized
      for i=1:N
         x(i)=sqrt(x(i));
      end
   end
   elapsed=toc;

   fprintf('CPU time for N=%d is %g s\n', N, elapsed);
end
```

Matlab examples [Slide: A. Donev]

```
>> fibb % Without pre-allocating
The result is f(100000)=5.00005e+09, computed in 6.53603 s

>> fibb % Pre-allocating
The result is f(100000)=5.00005e+09, computed in 0.000998 s

>> vect(0) % Non-vectorized
CPU time for N=1000000 is 0.074986 s

>> vect(1) % Vectorized -- don't trust the actual number
CPU time for N=1000000 is 0.002058 s
```

Vectorization/Optimization [Slide: A. Donev]

- Recall that everything in MATLAB is a double-precision matrix, called **array**.
- Row vectors are just matrices with first dimension 1. Column vectors have row dimension 1. Scalars are 1×1 matrices.
- The syntax x' can be used to construct the **conjugate transpose** of a matrix.
- The **colon notation** can be used to select a subset of the elements of an array, called an **array section**.
- The default arithmetic operators, +, -, *, / and ^ are **matrix addition/subtraction/multiplication**, linear solver and matrix power.
- If you prepend a **dot before an operator** you get an **element-wise operator** which works for arrays of the same shape.

Matrices [Slide: A. Donev]

```
>> x=[1 2 3; 4 5 6] % Construct a matrix
x =     1     2     3
        4     5     6

>> size(x) % Shape of the matrix x
ans =     2     3

>> y=x(:) % All elements of y
y =     1     4     2     5     3     6

>> size(y)
ans =     6     1

>> x(1,1:3)
ans =     1     2     3

>> x(1:2:6)
ans =     1     2     3
```

Matrices [Slide: A. Donev]

```
>> sum(x)
ans =
     5     7     9

>> sum(x(:))
ans =
    21

>> z=1i; % Imaginary unit

>> y=x+z
y =
   1.0000 + 1.0000i   2.0000 + 1.0000i   3.0000 + 1.0000i
   4.0000 + 1.0000i   5.0000 + 1.0000i   6.0000 + 1.0000i

>> y'
ans =
   1.0000 - 1.0000i   4.0000 - 1.0000i
   2.0000 - 1.0000i   5.0000 - 1.0000i
   3.0000 - 1.0000i   6.0000 - 1.0000i
```

Matrices [Slide: A. Donev]

```
>> x*y
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> x.*y
ans =
   1.0000 + 1.0000i   4.0000 + 2.0000i   9.0000 + 3.0000i
  16.0000 + 4.0000i  25.0000 + 5.0000i  36.0000 + 6.0000i

>> x*y'
ans =
  14.0000 - 6.0000i  32.0000 - 6.0000i
  32.0000 -15.0000i  77.0000 -15.0000i

>> x'*y
ans =
  17.0000 + 5.0000i  22.0000 + 5.0000i  27.0000 + 5.0000i
  22.0000 + 7.0000i  29.0000 + 7.0000i  36.0000 + 7.0000i
  27.0000 + 9.0000i  36.0000 + 9.0000i  45.0000 + 9.0000i
```

Coding guidelines [Slide: A. Donev]

- Learn to reference the **MATLAB help**: Including reading the examples and "fine print" near the end, not just the simple usage.
- **Indendation, comments, and variable naming** make a big difference! Code should be readable by others.
- Spending a few extra moments on the code will pay off when using it.
- Spend some time learning how to **plot in MATLAB**, and in particular, how to plot with different symbols, lines and colors using *plot, loglog, semilogx, semilogy.*
- Learn how to **annotate plots**: *xlim, ylim, axis, xlabel, title, legend.* The intrinsics *num2str* or *sprintf* can be used to create strings with embedded parameters.
- Finer controls over fonts, line widths, etc., are provided by the intrinsic function *set*...including using the LaTex interpreter to typeset mathematical notation in figures.

Today

Last time

- Condition of problems
- Stability of algorithms

Today

- Float-point numbers in IEEE format
- Rounding, propagation of errors, and cancellation
- Truncation errors

Announcements

- Homework 1 was posted last week; is due next week Mon, Sep 23 *before class*

Recap: Condition of a problem

- Terms such as "little bit" and a "small amount" already point to that we need to measure something
- Therefore, we assume the map f is given as

$$f : U \subset \mathbb{R}^n \to \mathbb{R}^m$$

  and we are interested in the norm $\|\cdot\|$
- The input error is then

$$\|x - \hat{x}\| \leq \delta \text{ (absolute)} \qquad \|x - \hat{x}\| \leq \delta \|x\| \text{ (relative)}$$

- Correspondingly we measure the output error $f(x) - f(\hat{x})$ in $\|\cdot\|$ (we could also have looked at a componentwise error)

Recap: Condition of a problem (cont'd)

- Absolute condition number at x is

$$\kappa_{\text{abs}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\|}{\|x - \hat{x}\|}$$

- Relative condition number at x is

$$\kappa_{\text{rel}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\| / \|f(x)\|}{\|x - \hat{x}\| / \|x\|}$$

- If f is differentiable in x, then

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\| \,,$$

where $\|f'(x)\|$ is the norm of the Jacobian $f'(x)$ in the operator norm

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$,

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$,

Recap: Condition of a problem (cont'd)

- If $\kappa_{\text{rel}} \sim 1$, then the problem is well conditioned: If the relative error in the data/input is small, then the relative error in the answer/output is similarly small
- If $\kappa_{\text{rel}} \gg 1$, then the problem is poorly conditioned: Small relative input error can lead to large relative output error
- If κ_{rel} (and κ_{abs}) do not exist, then the problem is ill conditioned.

- What is poorly conditioned depends on desired accuracy: if the input accuracy is low but we expect a high output accuracy, then problems are quickly poorly conditioned. If we are happy with a less accurate output, we might consider the problem still well conditioned.
- Sometimes, the possibly large error in the output does not matter and so we can solve poorly conditioned problems (think of early design stages, rapid prototyping, etc); but we should be very much aware of the condition of the problem.

Recap: Condition number of a matrix

Consider a matrix $A \in \mathbb{R}^{n \times n}$. Its condition number is

$$\kappa(A) = \|A\| \|A^{-1}\|$$

Widely used is the $\| \cdot \|_2$ norm and then

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

with the maximal and minimal singular value $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ of A

Consider a system of linear equations $Ax = b$. Then, the problems $A \mapsto A^{-1}b$ and $b \mapsto A^{-1}b$ have relative condition numbers

$$\kappa_{\text{rel}} \leq \kappa(A)$$

Recap: Backward stability

Backward stability: Pass the errors of the algorithm back and interpret as input errors.

An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

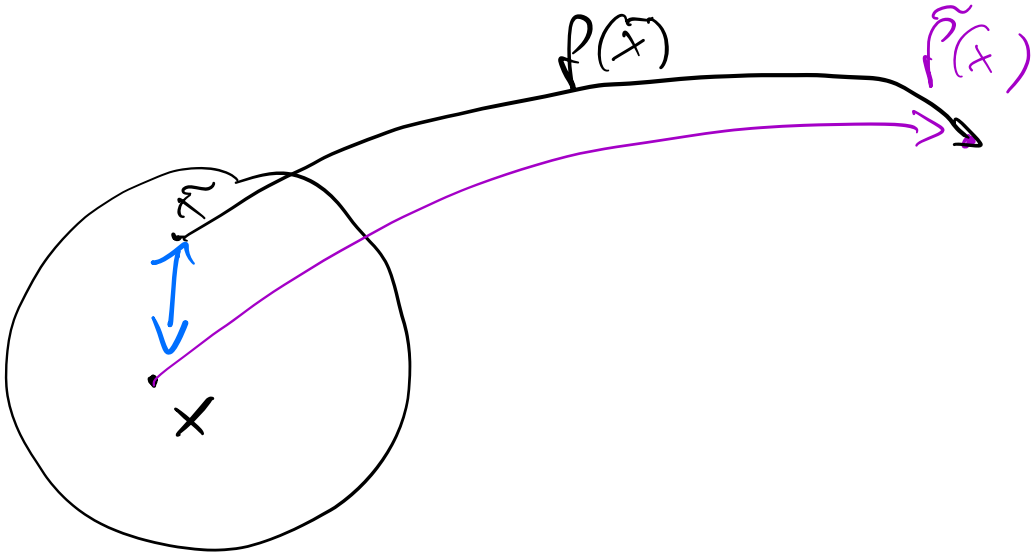
$$\frac{\|\tilde{x} - x\|}{\|x\|}$$

small

> *A backward stable algorithm gives exactly the right answer to nearly the right question.*

In backward error analysis one calculates, for a given output, how much one would need to perturb the input in order for the answer to be exact.

Recap: Backward stability (cont'd)



Representing real numbers

Representing real numbers

- Computers represent everything using bit strings, i.e., integers in base 2. A finite number of integers can thus be exactly represented. But not real numbers! This leads to roundoff errors.

- Assume we have N digits to represent real numbers on a computer that can represent integers using a given number system, say decimal for human purposes.

- Fixed-point representation of numbers

$$x = (-1)^s \cdot [a_{N-2} a_{N-3} \cdots a_k . a_{k-1} \cdots a_0]$$

  has a problem of representing either small or larger numbers because the decimal point . is fixed at position k

What could we do?

Floating-point numbers

- Instead, let's use floating-point representation

$$x = (-1)^s \cdot [0\,.\,a_1 a_2 \cdots a_t] \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}$$

  similar to the common scientific number representation

$$0.1156 \cdot 10^1 = 1156 \cdot 10^{-3} \qquad t = 4$$

- A floating-point number in base β is represented using one sign bit $s = 0$ or 1, a t-digit integer mantissa

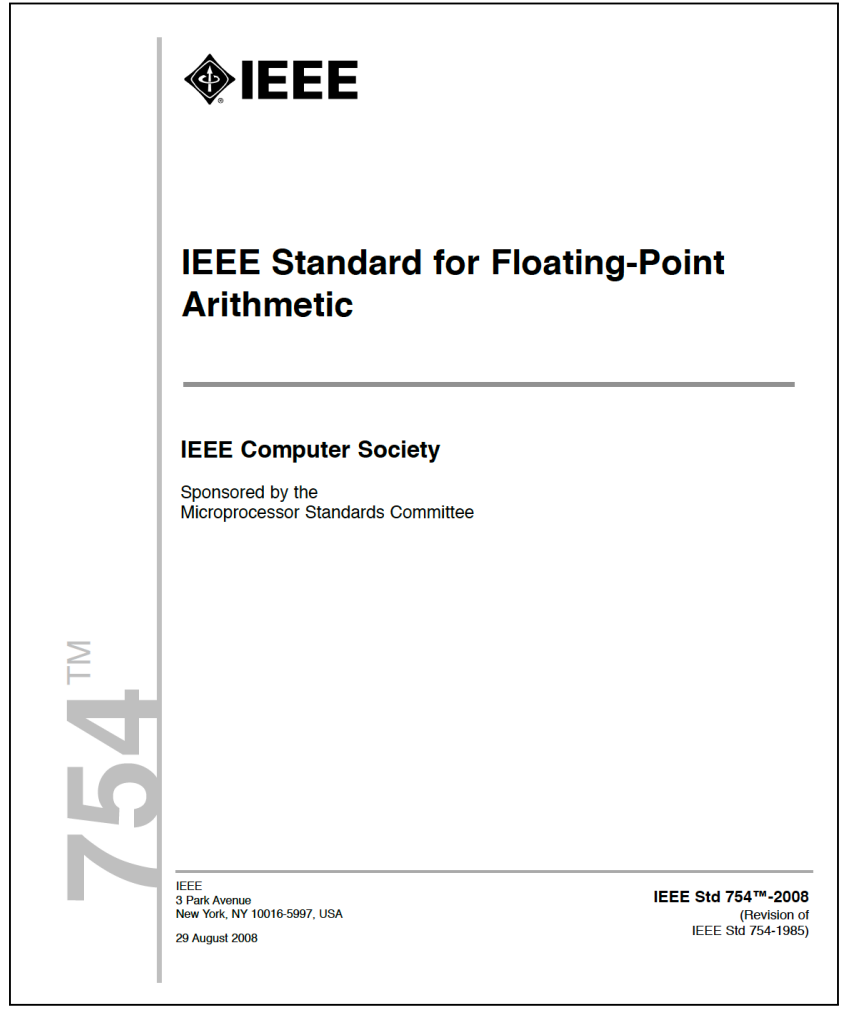
$$0 \leq m = [a_1 a_2 \cdots a_t] \leq \beta^t - 1$$

  and an integer exponent $L \leq e \leq U$

- Computers today use binary numbers and so $\beta = 2$

IEEE 754 standard

- Formats for representing and encoding real numbers using bit strings (single and double precision).
- Rounding algorithms for performing accurate arithmetic operations (e.g., addition, subtraction, division, multiplication) and conversions (e.g., single to double precision).
- Exception handling for special situations (e.g., division by zero and overflow).



IEEE

IEEE Standard for Floating-Point Arithmetic

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

754™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

29 August 2008

IEEE Std 754™-2008
(Revision of
IEEE Std 754-1985)

Single precision IEEE floating-point numbers have the standardized storage format:

$$\text{sign} + \text{power} + \text{fraction}$$

with

$$N_s + N_p + N_f = 1 + 8 + 23 = 32 \text{ bits}$$

and are interpreted as

$$x = (-1)^s \cdot 2^{p-127} \cdot (1.f)_2$$

- Sign $s = 1$ for negative numbers
- Power $1 \leq p \leq 254$ determines the exponent
- Fractional part of the mantissa f
- `single` in Matlab, `float` in C/C++, REAL in Fortran

IEEE representation example

Take the number $x = 2752 = \underline{0.2752 \cdot 10^4}$.

Binary $2752/2$ remainder 0

$1376/2$ remainder 0

$\vdots$

$\downarrow \quad \downarrow \quad \downarrow\downarrow$

$101011\ 000\ 000$

$11 \quad 9 \quad 76$

$$x = \underline{2^{11}} + 2^9 + 2^7 + 2^6$$

$$x = 2^{11} \cdot (1.01011)_2$$

$$= (-1)^0 \, 2^{138-127} \cdot (1.01011)$$

$$(138)_{10} = (10001010)_2$$

$$x = (-1)^0 \, 2^{(10001010)_2 - 127} \cdot (1.\underbrace{01011}_{f})_2$$

$$x = [s \mid p \mid f] = \boxed{[0 \mid 10001010 \mid 01011\,0\ldots0]}$$

$$= \underline{(452c\,000)_{16}}$$

IEEE representation example

Take the number $x = 2752 = 0.2752 \cdot 10^4$. Converting 2752 to the binary number system

$$\begin{aligned} x &= 2^{11} + 2^9 + 2^7 + 2^6 = (101011000000)_2 = 2^{11} \cdot (1.01011)_2 \\ &= (-1)^0 2^{138-127} \cdot (1.01011)_2 = (-1)^0 2^{(10001010)_2 - 127} \cdot (1.01011)_2 \end{aligned}$$

On the computer:

$$\begin{aligned} x &= [s \quad | \quad p \quad | \quad f] \\ &= [0 \quad | \quad 100,0101,0 \quad | \quad 010,1100,0000,0000,0000,0000] \\ &= (452c0000)_{16} \end{aligned}$$

(0100 0101 0010 1100 0000 0000 0000 0000)_2
=(452c0000)_16

```
format hex;
>> a=single(2.752E3)
a =
    452c0000
```

Double precision IEEE numbers

Double precision IEEE numbers (default in Matlab, `double` in C/C++) follow the same principle but use 64 bits to give higher precision and range

$$N_s + N_p + N_f = 1 + 11 + 52 = 64 \text{ bits}$$

$$x = (-1)^s \cdot 2^{p-1023} \cdot (1.f)_2$$

Even higher (extended) precision formats are not really standardized or widely implemented/used.

There is also software-emulated variable precision arithmetic in, e.g., Maple

Extremal exponent values

The extremal exponent values have special meaning (here single precision)

value	power p	fraction f
± 0	0	0
$\pm\infty$	255	0
Not a number (NaN)	255	> 0

Important facts about floating-point numbers

- Not all real numbers x can be represented exactly as a floating-point number. Instead, they must be *rounded* to the nearest floating point number $\hat{x} = \mathrm{fl}(x)$

- Floating-point numbers have a *relative rounding error* that is smaller than the *machine precision* or *roundoff-unit* u

$$\frac{|\hat{x} - x|}{|x|} \leq u = 2^{-(N_f+1)} = \begin{cases} 2^{-24} \sim 6.0 \cdot 10^{-8}\,, & \text{for single precision} \\ 2^{-53} \sim 1.1 \cdot 10^{-16}\,, & \text{for double precision}\,. \end{cases}$$

- Often the machine precision/roundoff-unit is denoted as ϵ

- **The rule of thumb is that single precision gives 7-8 digits of precision and double 16 digits.**

- There is a smallest and largest possible number due to limit for the exponent.

Two axioms

Ignoring over- and underflow, we assume the following two "axioms" to hold for computers we work with:

1. For all $x \in \mathbb{R}$, there exists ϵ with $|\epsilon| \leq u$ (roundoff unit) such that

$$\mathrm{fl}(x) = x(1+\epsilon)\,,$$

   where $\mathrm{fl}(\cdot)$ rounds to the the closest floating point approximation.

2. Consider two floating point numbers x, y. The floating-point operation $\circledast$ (=add, sub, mult, div) of $*$ (=add, sub, mult, div) satisfies

$$x \circledast y = \mathrm{fl}(x * y)$$

Axiom 1 and 2 imply that for two floating-point numbers x, y, there exists ϵ with $|\epsilon| \leq u$ such that

$$x \circledast y = (x * y)(1+\epsilon)\,.$$

Floating-point exceptions

Computing with floating point values may lead to exceptions, which may halt the program:

Floating-point exceptions

Computing with floating point values may lead to exceptions, which may halt the program:

- **Divide-by-zero**: if the result is $\pm\infty$, e.g., $1/0$

Floating-point exceptions

Computing with floating point values may lead to exceptions, which may halt the program:

- **Divide-by-zero**: if the result is $\pm\infty$, e.g., $1/0$
- **Invalid**: If the result is a *NaN*, e.g., taking $\sqrt{-1}$ (note that Matlab supports complex numbers...)

```
>>> x = math.sqrt(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

Floating-point exceptions

Computing with floating point values may lead to exceptions, which may halt the program:

- **Divide-by-zero**: if the result is $\pm\infty$, e.g., $1/0$
- **Invalid**: If the result is a *NaN*, e.g., taking $\sqrt{-1}$ (note that Matlab supports complex numbers...)

```
>>> x = math.sqrt(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

- **Overflow**: If the result is too large to be represented, e.g., adding two numbers, each on the order of *realmax*

Floating-point exceptions

Computing with floating point values may lead to exceptions, which may halt the program:

- **Divide-by-zero**: if the result is $\pm\infty$, e.g., $1/0$
- **Invalid**: If the result is a *NaN*, e.g., taking $\sqrt{-1}$ (note that Matlab supports complex numbers...)

```
>>> x = math.sqrt(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
```

- **Overflow**: If the result is too large to be represented, e.g., adding two numbers, each on the order of *realmax*
- **Underflow**: If the result is too small to be represented, e.g., dividing a number close to *realmin* by a large number.

Avoiding overflow

Numerical software needs to be careful about avoiding exceptions:

Mathematically equivalent expressions are not necessarily computationally equivalent!

- For example, computing $\sqrt{x^2 + y^2}$ may lead to overflow in computing $x^2 + y^2$ even though the result does not overflow
- Matlab's hypot function guards against this:

$$\sqrt{x^2 + y^2} = |x|\sqrt{1 + \left(\frac{y}{x}\right)^2} \text{ ensuring that } |x| > |y|$$

  works correctly
- These kind of careful constructions may have higher computational cost (more CPU operations) or make roundoff errors worse.

Floating-point in practice

- Most scientific software **uses double precision** to avoid range and accuracy issues with single precision (better be safe then sorry). Single precision may offer speed/memory/vectorization advantages however (e.g. GPU computing).
- **Do not compare floating point numbers** (especially for loop termination), or more generally, do not rely on logic from pure mathematics.
- Optimization, especially in compiled languages, can rearrange terms or perform operations using **unpredictable** alternate forms (e.g., wider internal registers).
  Using parenthesis helps , e.g. $(x + y) - z$ instead of $x + y - z$, but does not eliminate the problem.
- Library functions such as sin and ln will typically be computed almost to full machine accuracy, but do not rely on that for special/complex functions.

Propagation of errors

- Assume that we are calculating something with numbers that are not exact, e.g., a rounded floating-point number $\hat{x}$ versus the exact real number x.
- For IEEE representations, recall that

$$\frac{|\hat{x} - x|}{|x|} \leq u = 2^{-(N_f+1)} = \begin{cases} 2^{-24} \sim 6.0 \cdot 10^{-8}\,, & \text{for single precision} \\ 2^{-53} \sim 1.1 \cdot 10^{-16}\,, & \text{for double precision}\,. \end{cases}$$

- In general, the *absolute error* $\delta x = \hat{x} - x$ may have contributions from each of the different types of error (roundoff, truncation, propagated, statistical).
- Assume we have an estimate or bound for the relative error

$$\left|\frac{\delta x}{x}\right| \lesssim \epsilon_x \ll 1$$

based on some analysis, e.g., for roundoff error the IEEE standard determines $\epsilon_x = u$ (roundoff-unit)

Propagation of errors

How does the relative error change (propagate) during numerical calculations?

Propagation of errors

multiplication / division

$$\varepsilon_{xy} = \left| \frac{(x+\delta x)(y+\delta y) - xy}{xy} \right|$$

$$= \left| \cancel{\frac{xy}{xy}} + \frac{\delta x \cancel{y}}{x \cancel{y}} + \frac{\cancel{x}\delta y}{\cancel{x} y} + \frac{\delta x \delta y}{xy} - \cancel{\frac{xy}{xy}} \right|$$

$$= \left| \frac{\delta x}{x} + \frac{\delta y}{y} + \frac{\delta x \delta y}{xy} \right|$$

Now use

$$\left|\frac{\delta x}{x}\right| \ll 1 \,, \quad \left|\frac{\delta y}{y}\right| \ll 1$$

$$\left|\frac{\delta x \delta y}{xy}\right| \leq \max\left\{ \left|\frac{\delta x}{x}\right|, \left|\frac{\delta y}{y}\right| \right\}$$

Then

$$\left| \frac{\delta x}{x} + \frac{\delta y}{y} + \frac{\delta x \delta y}{xy} \right| \leq 2(\varepsilon_x + \varepsilon_y)$$

$$\underset{\sim}{<} \varepsilon_x + \varepsilon_y$$

up to constant

$\Rightarrow$ rel. errors are added

$\Rightarrow$ safe because accurate input gives output of similar accuracy

Addition / subtraction

this is more dangerous

$\Rightarrow$ lead to catastrophic errors

Example

$x = 1$, $y = 0.999$, $\delta x = 10^{-6}$, $\delta y = 9 \times 10^{-7}$

$$\left|\frac{\delta x}{x}\right| = \underline{10^{-6}}, \quad \left|\frac{\delta y}{y}\right| = \left|\frac{9 \times 10^{-7}}{9.99 \times 10^{-1}}\right| \approx \underline{10^{-6}}$$

$$\left|\frac{(x+\delta x) - (y+\delta y) - (x-y)}{x-y}\right|$$

$$= \left|\frac{\delta x - \delta y}{x-y}\right| = \frac{10^{-7}}{10^{-3}} = 10^{-4} \gg 10^{-6}$$

$\Rightarrow$ errors can propogate quickly in +/−

Propagation of errors: Numerical experiment [From A. Donev]

Harmonic sum

$$H(N) = \sum_{i=1}^{N} \frac{1}{i}$$

```
function nhsum = harmonic(N)
nhsum = 0;
for i = 1:N
    nhsum = nhsum + 1.0/i;
end
end
```

What are the numerical issues of this implementation?

Propagation of errors: Numerical experiment [From A. Donev]

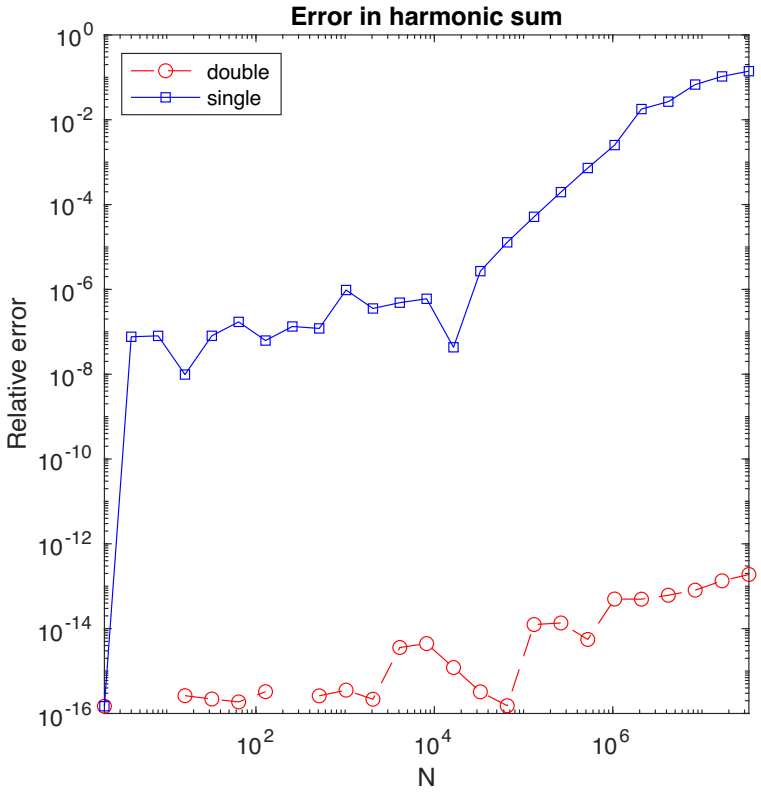
Harmonic sum

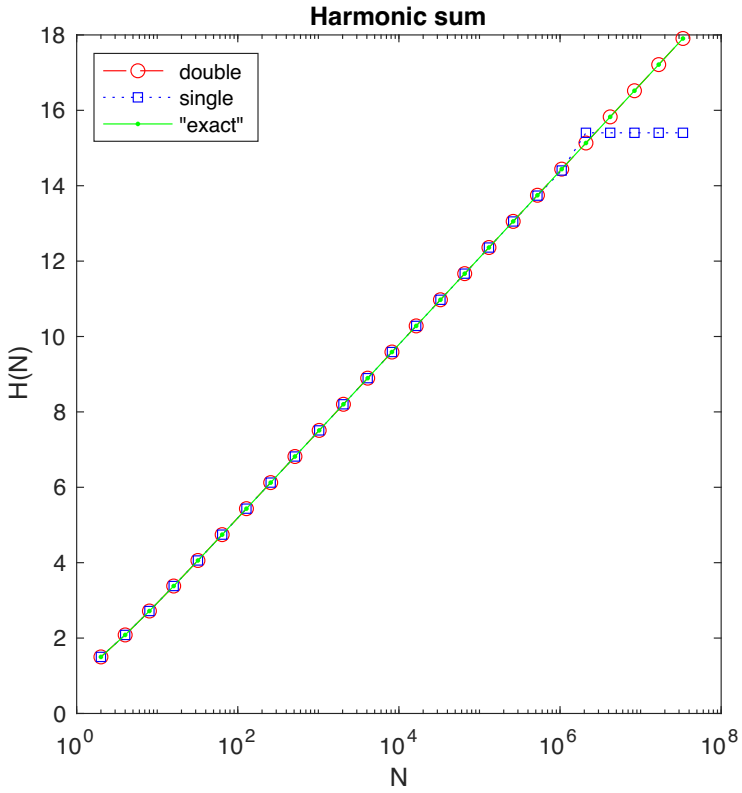
$$H(N) = \sum_{i=1}^{N} \frac{1}{i}$$

```
function nhsum = harmonic(N)
nhsum = 0;
for i = 1:N
    nhsum = nhsum + 1.0/i;
end
end
```

What are the numerical issues of this implementation?

$\rightsquigarrow$ Adds very small number $1/i$ to potentially large number `nhsum`



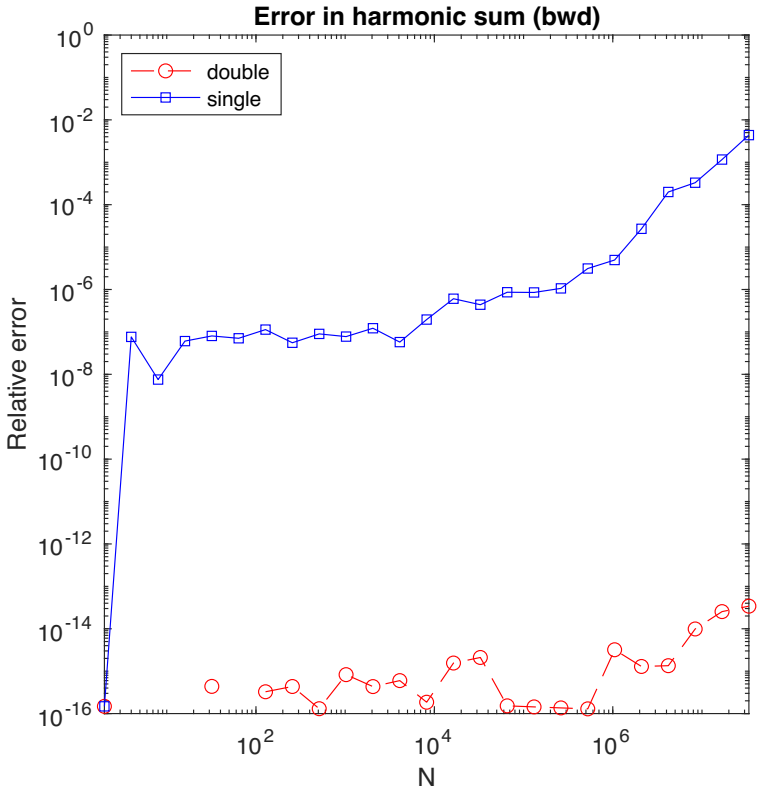


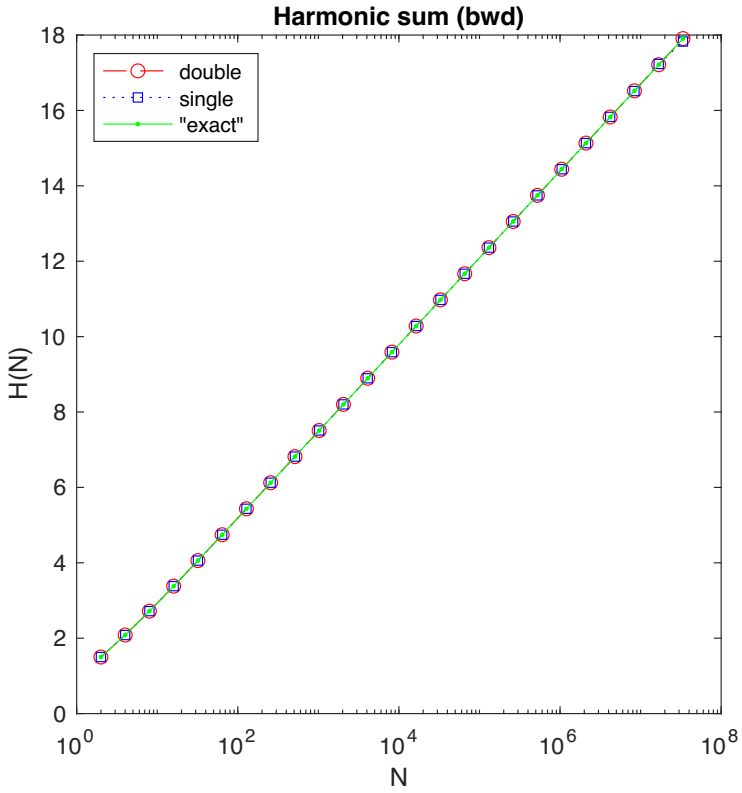
What can we do about it?

Implementation with backward summation

```
function nhsum = harmonicBwd(N)
nhsum = 0;
for i = N:-1:1
    nhsum = nhsum + 1.0/i;
end
end
```

Better, because adds small numbers to small numbers and larger numbers to large numbers.


Error in harmonic sum (bwd)
double
single
Relative error
N
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{2}
10^{4}
10^{6}


Harmonic sum (bwd)
double
single
"exact"
H(N)
N
0
2
4
6
8
10
12
14
16
18
10^{0}
10^{2}
10^{4}
10^{6}
10^{8}

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Float-point numbers in IEEE format
- Rounding
- Propagation of errors

Today

- Cancellation
- Truncation errors
- Solving linear systems

Announcements

- Homework 1 was posted last week; is due next week Mon, Sep 23 *before class*

→ Grader office hour: Fri, 3-4pm, WWH 803

Recap: Floating-point numbers

- Let's use floating-point representation

$$x = (-1)^s \cdot [0\,.\,a_1 a_2 \cdots a_t] \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t}$$

  similar to the common scientific number representation

$$0.1156 \cdot 10^1 = 1156 \cdot 10^{-3} \qquad t = 4$$

- A floating-point number in base β is represented using one sign bit $s = 0$ or 1, a t-digit integer mantissa

$$0 \leq m = [a_1 a_2 \cdots a_t] \leq \beta^t - 1$$

  and an integer exponent $L \leq e \leq U$

- Computers today use binary numbers and so $\beta = 2$

Recap: Single precision

Single precision IEEE floating-point numbers have the standardized storage format:

$$\text{sign} + \text{power} + \text{fraction}$$

with

$$N_s + N_p + N_f = 1 + 8 + 23 = 32 \text{ bits}$$

and are interpreted as

$$x = (-1)^s \cdot 2^{p-127} \cdot (1.f)_2$$

- Sign $s = 1$ for negative numbers
- Power $1 \leq p \leq 254$ determines the exponent
- Fractional part of the mantissa f
- `single` in Matlab, `float` in C/C++, REAL in Fortran

Recap: Important facts about floating-point numbers

- Not all real numbers x can be represented exactly as a floating-point number. Instead, they must be *rounded* to the nearest floating point number $\hat{x} = \mathrm{fl}(x)$

- Floating-point numbers have a *relative rounding error* that is smaller than the *machine precision* or *roundoff-unit* u

$$\frac{|\hat{x} - x|}{|x|} \leq u = 2^{-(N_f+1)} = \begin{cases} 2^{-24} \sim 6.0 \cdot 10^{-8}\,, & \text{for single precision} \\ 2^{-53} \sim 1.1 \cdot 10^{-16}\,, & \text{for double precision}\,. \end{cases}$$

- Often the machine precision/roundoff-unit is denoted as ϵ

- **The rule of thumb is that single precision gives 7-8 digits of precision and double 16 digits.**

- There is a smallest and largest possible number due to limit for the exponent.

Recap: Two axioms

Ignoring over- and underflow, we assume the following two "axioms" to hold for computers we work with:

1. For all $x \in \mathbb{R}$, there exists ϵ with $|\epsilon| \leq u$ (roundoff unit) such that

$$\mathrm{fl}(x) = x(1+\epsilon)\,,$$

   where $\mathrm{fl}(\cdot)$ rounds to the the closest floating point approximation.

2. Consider two floating point numbers x, y. The floating-point operation $\circledast$ (=add, sub, mult, div) of $*$ (=add, sub, mult, div) satisfies

$$x \circledast y = \mathrm{fl}(x * y)$$

Axiom 1 and 2 imply that for two floating-point numbers x, y, there exists ϵ with $|\epsilon| \leq u$ such that

$$x \circledast y = (x * y)(1+\epsilon)\,.$$

Numerical cancellation

If x and y are close to each other, then $x - y$ can have reduced accuracy due to catastrophic cancellation.

Consider computing the smaller root of the quadratic equation

$$x^2 - 2x + c = 0$$

for $|c| \ll 1$ and focus on propagation/accumulation of roundoff errors.

$$x^2 - 2x + c = 0 \quad , \quad |c| \ll 1$$

Solution: $x = 1 - \sqrt{1-c}$

Before we start: Taylor

$$f(z) = \sqrt{1+z} \quad , \quad f'(z) = \frac{1}{2\sqrt{1+z}}$$

$$f(z) = f(0) + \frac{f'(0)}{1!}(z-0) + \frac{f''(0)}{2!}(z-0)^2 + \dots$$

truncate: $\sqrt{1+z} \doteq 1 + \frac{1}{2\sqrt{1+0}} z = 1 + \frac{z}{2}$

Thus, for $|c|$ small, get

$$x = 1 - \sqrt{1-c} \doteq 1 - \left(1 - \frac{c}{2}\right) = \frac{c}{2}$$

Case 1: $|c| < |u|$ ← round-off unit

$$fl(1-c) = 1$$

no point in moving forward

Case 2: $|u| < |c| \ll 1$

Calculate $1-c$ in floating point

$$fl(1) \ominus fl(c) = fl(fl(1) - fl(c))$$

$$= (fl(1) - fl(c))(1+\varepsilon) \qquad |\varepsilon| \le u$$

$$= \underbrace{(1(1+\varepsilon_1) - c(1+\varepsilon_c))}_{(1-c) + (\varepsilon_1 - c\varepsilon_c)}(1+\varepsilon) \quad , \quad |\varepsilon_1| \le u, \ |\varepsilon_c| \le u$$

$$fl(1) \ominus fl(c) - (1-c) = \underbrace{(1-c)}_{\text{order } 1}\varepsilon + \underbrace{\underbrace{(\varepsilon_1 - \overbrace{c\varepsilon_c}^{\text{order } |u|})}_{\text{order } |u|}\underbrace{(1+\varepsilon)}_{\text{order } |u|}}_{\text{order } |u|}$$

$\left| \frac{\delta(1-c)}{1-c} \right|$ also order $|u|$ because $1-c$ close to 1

Assume sqrt() computes root to within relative accuracy of round-off unit u

$$\sqrt{x+\delta x} = \sqrt{x}\left(1+\frac{\delta x}{x}\right)^{1/2}$$
$$\doteq \sqrt{x}\left(1+\frac{\delta x}{2x}\right)$$

So for shows that $\sqrt{1-c}$ has abs and relative error of order $|u|$

We have $x = 1-\sqrt{1-c}$

already know

$$fl(1) \ominus fl(y) - (1-y) = \delta(1-y)$$

is of order $|1-y|\cdot|u| + |u|$

Relative error:

$$\left| \frac{\delta(1-y)}{1-y} \right| \approx \frac{|1-y|\cdot|u| + |u|}{|1-y|} = |u| + \frac{|u|}{|1-y|}$$

It is higher than $|u|$ for $1-y \approx 0$

To avoid cancellation, we should not directly implement $1 - \sqrt{1-c}$

Rather, we can take the Taylor approximate $x \approx \frac{c}{2}$, which provides a good approximation for small c.

Even better, we could use the **mathematically equivalent but numerically preferred form**:

$$1 - \sqrt{1-c} = \frac{c}{1 + \sqrt{1-c}}$$

which does not suffer from cancellation problems as c becomes smaller.
(Notice that $1 - \sqrt{\ldots}$ is avoided and therefore the cancellation problem shown by our analysis is avoided. We showed that $\sqrt{1-c}$ is safe.)

To avoid cancellation, we should not directly implement $1 - \sqrt{1-c}$

Rather, we can take the Taylor approximate $x \approx \frac{c}{2}$, which provides a good approximation for small c.

Even better, we could use the **mathematically equivalent but numerically preferred form**:

$$1 - \sqrt{1-c} = \frac{c}{1+\sqrt{1-c}}$$

which does not suffer from cancellation problems as c becomes smaller.
(Notice that $1 - \sqrt{\ldots}$ is avoided and therefore the cancellation problem shown by our analysis is avoided. We showed that $\sqrt{1-c}$ is safe.)

```
>>> c = 1e-10 # solution roughly 5.000000000125 x 10^-11
>>> 1 - math.sqrt(1 - c)
5.000000413701855e-11
>>> c/(1 + math.sqrt(1 - c))
5.000000000125e-11
```

Big $\mathcal{O}$ notation

Useful to compare growth of functions.

We write $f \in \mathcal{O}(g)(x \to \infty)$ if there exists constant $C > 0$ such that for an x_0 the following holds

$$\forall x \geq x_0 : \quad |f(x)| \leq C|g(x)|$$

We also write $f \in \mathcal{O}(g)(x \to 0)$ if there exists a constant $C > 0$ such that for an $x_0 > 0$ the following holds

$$\forall |x| \leq x_0 : \quad |f(x)| \leq C|g(x)|$$

In many cases we do not write explicitly whether we mean $x \to \infty$ or $x \to 0$ because it is clear from the context.

Question: In practice, would you prefer an algorithm with costs growing as $c_1(x) \in \mathcal{O}(x)$ or $c_2(x) \in \mathcal{O}(x^2)$? Why?

Question: In practice, would you prefer an algorithm with costs growing as $c_1(x) \in \mathcal{O}(x)$ or $c_2(x) \in \mathcal{O}(x^2)$? Why?

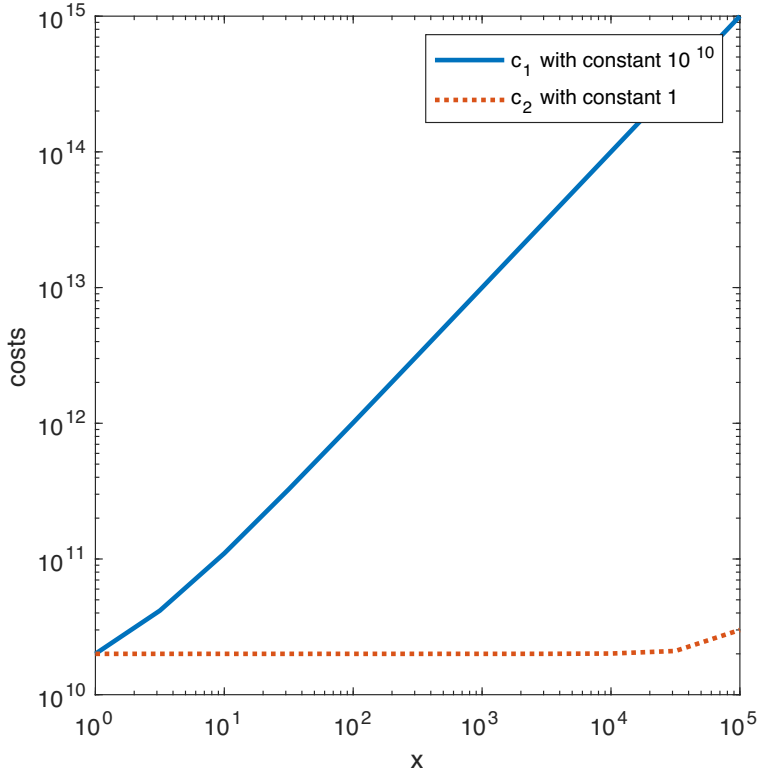
Answer: It depends on the hidden constants C and x_0. If c_1 and c_2 have roughly the same constants, then probably c_1.

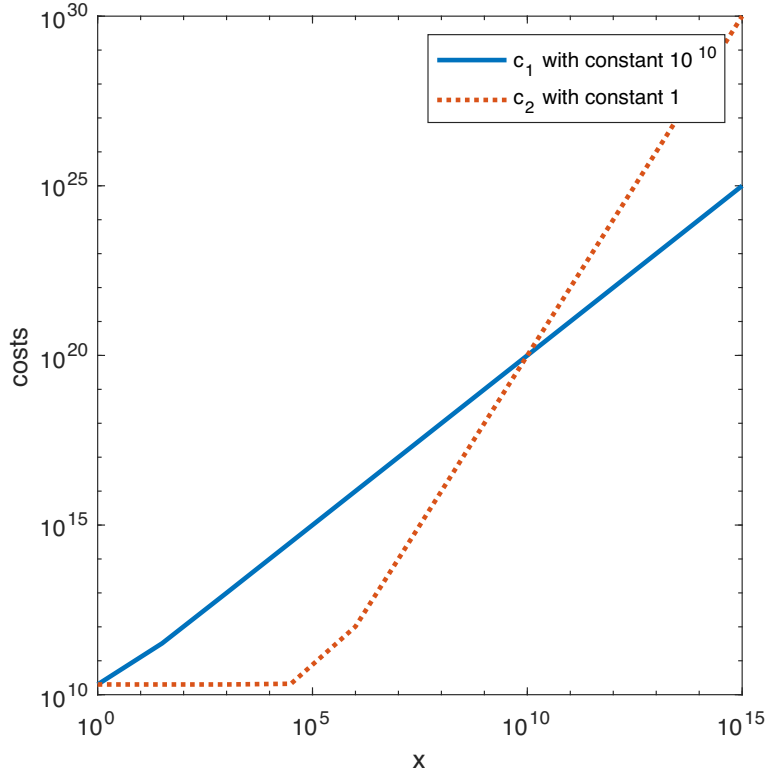
However, if the constant for c_1 is $x_0 = 10^{10}$ and the constant for c_2 is $x_0 = 1$, then in most practical situations we prefer c_2 because we most likely will never reach the asymptotics of $x > x_0$ for c_1 in practice!

Warning: The Big $\mathcal{O}$ notation tells us something about the asymptotics. The constants x_0 and C that are hidden in $\mathcal{O}(\cdot)$ do matter in practice!

Costs (i.e., $x \to \infty$) of $c_1(x) = 10^{10} + 10^{10}x$ and $c_2(x) = 2 \times 10^{10} + x^2$. Then, $c_1 \in \mathcal{O}(x)$ and $c_2 \in \mathcal{O}(x^2)$ for $x \to \infty$. I.e., asymptotically the costs of c_2 grow faster than c_1.

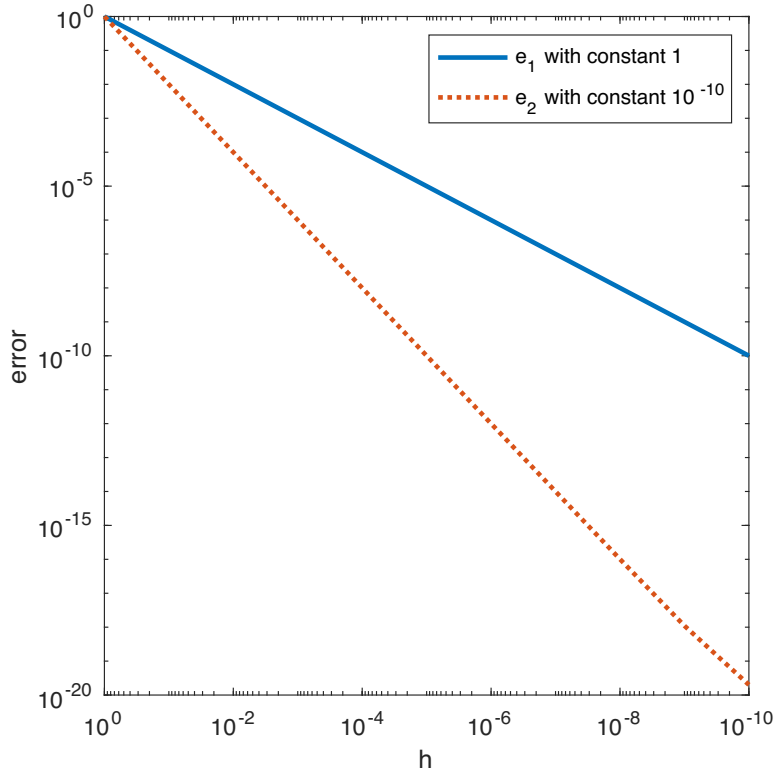
Costs (i.e., $x \to \infty$) of $c_1(x) = 10^{10} + 10^{10}x$ and $c_2(x) = 2 \times 10^{10} + x^2$. Then, $c_1 \in \mathcal{O}(x)$ and $c_2 \in \mathcal{O}(x^2)$ for $x \to \infty$. I.e., asymptotically the costs of c_2 grow faster than c_1.

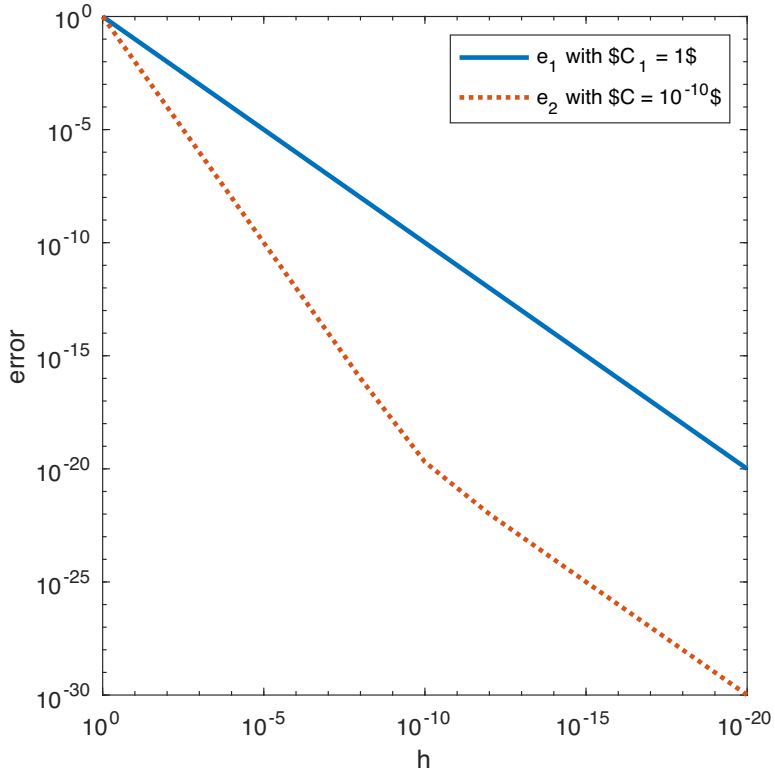




Warning: The Big $\mathcal{O}$ notation tells us something about the asymptotics. The constants x_0 and C that are hidden in $\mathcal{O}(\cdot)$ do matter in practice!

Set now error: $e_1(h) = h$ and $e_2(h) = 10^{-10}h + h^2$. Then, $e_1 \in \mathcal{O}(h)$ and $e_2 \in \mathcal{O}(h)$ for $h \to 0$.





Warning: The Big $\mathcal{O}$ notation tells us something about the asymptotics. The constants h_0 and C that are hidden in $\mathcal{O}(\cdot)$ do matter in practice!

Revisiting stability

Recall that we said: An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|}$$

small.

We now can be more precise: An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} \in \mathcal{O}(u)\,,$$

where u is the roundoff unit

- Recall that, loosely speaking, the symbol $\mathcal{O}(u)$ means "on the order of the roundoff unit."
- By allowing $u \to 0$ (which is implied here by the $\mathcal{O}$), we consider an idealization of a computer (in practice, u is fixed). So what we mean is that the error should decrease in proportion to u or faster.

Suppose a backward stable algorithm is applied to solve a problem $f : X \to Y$ with relative condition number κ. Then, the relative errors satisfy

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \in \mathcal{O}(\kappa(x)u)\,.$$

Proof board

Let $\tilde{f}$ be a backward stable algorithm of $f: X \to Y$ with rel condition k, then

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \in O(k(x)\, u)$$

where u is the round off unit.

Backward stability tells us $\tilde{f}(x) = f(\tilde{x})$ for $\tilde{x} \in X$ satisfying

$$\frac{\|\tilde{x} - x\|}{\|x\|} \in O(u)$$

Condition number

$$k = \lim_{\delta \to 0} \sup_{\|x - \tilde{x}\| \le \delta} \frac{\|f(x) - f(\tilde{x})\|}{\|f(x)\|} \Big/ \frac{\|x - \tilde{x}\|}{\|x\|}$$

thus with $\tilde{f}(x) = f(\underset{\tilde{x}}{\underset{\parallel}{\tilde{x}}})$ we have

$$\frac{\|f(x) - \tilde{f}(x)\|}{\|f(x)\|} \le k(x) \underbrace{\frac{\|x - \tilde{x}\|}{\|x\|}}_{\in O(u)} \qquad u \to 0$$

$$\in O(k(x)\, u)$$

Local truncation error

- **Approximation error** comes about when we replace a mathematical problem with some easier to solve approximation.
- This error is separate from and in addition to any numerical algorithm or computation used to actually solve the approximation itself, such as roundoff or propagated error.
- Truncation error is a common type of approximation error that comes from replacing infinitesimally small quantities with finite step size and truncating infinite sequences/series with finite ones.
- This is the most important type of error in methods for numerical interpolation, integration, solving differential equations, and others.

Local truncation error (cont'd)

- Analysis of local truncation error is almost always based on using Taylor series to approximate a function about a given point x

$$f(x+h) = \sum_{n=0}^{\infty} \frac{h^n}{n!} f^{(n)}(x) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \dots ,$$

  where we call h the step size
- We cannot do a series (infinite number of terms) numerically, so we truncate

$$f(x+h) \approx F_p(x,h) = \sum_{n=0}^{p} \frac{h^n}{n!} f^{(n)}(x)$$

- Question: What is the truncation error in this approximation? $\rightsquigarrow$ **This kind of error estimate is one of the most commonly used in numerical analysis.**

- The remainder theorem of calculus provides a formula for the error: If the derivatives of f up to order $p+1$ exist and are continuous in the interval $(x, x+h)$, then there is a $\xi \in [x, x+h]$ so that

$$f(x+h) - F_p(x,h) = \frac{h^{p+1}}{(p+1)!} f^{(p+1)}(\xi)$$

- If we set

$$C = \frac{1}{(p+1)!} \max_{y \in [x, x+h]} \left| f^{(p+1)}(y) \right|$$

then

$$|f(x+h) - F_p(x,h)| \leq C h^{p+1}$$

- Intuition: If h is small and $f^{(p+1)}$ smooth, then $x \approx \arg\max_{y \in [x, x+h]} \left| f^{(p+1)}(y) \right|$ and the remainder term is nearly equal to the first neglected term

$$f(x+h) - F_p(x,h) \approx \frac{h^{p+1}}{(p+1)!} f^{(p+1)}(x)$$

For example, let $e(h) = |f(x+h) - F_p(x,h)|$ and let the remainder theorem from the previous slide apply, then

$$e(h) \in \mathcal{O}(h^{p+1}), \qquad h \to 0$$

Conclusions and summary

- No numerical method can compensate for a poorly conditioned problem. But not every numerical method will be a good one for a well conditioned problem.
- A numerical method needs to control the various computational errors (approximation, truncation, roundoff, propagated, statistical) while balancing computational cost.
- A numerical method must be consistent and stable in order to converge to the correct answer.
- The IEEE standard standardizes the single and double precision floating-point formats, their arithmetic, and exceptions. It is widely implemented.
- Numerical overflow, underflow and cancellation need to be carefully considered and avoided: Mathematically equivalent forms are not numerically equivalent.

Linear systems

Linear systems of equations

It is said that 70-80% of computational mathematics research involves solving systems of m linear equations in n unknowns

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \,, \qquad i = 1, \ldots, m \,.$$

Linear systems arise directly from discrete models (e.g., in machine learning). Or through representing some abstract linear operator (such as a differential operator) in a finite basis as when numerically solving partial differential equations.

The common abstract way of writing systems of linear equations is

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \,,$$

with matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^m$, and solution $\boldsymbol{x} \in \mathbb{R}^n$

The goal is to calculate solution $\boldsymbol{x}$ given data $\boldsymbol{A}, \boldsymbol{b}$ in a numerically stable and computationally efficient way.

The matrix inverse

- A square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is invertible or nonsingular if there exists a matrix inverse $\boldsymbol{A}^{-1} = \boldsymbol{B} \in \mathbb{R}^{n \times n}$ such that

$$\boldsymbol{AB} = \boldsymbol{BA} = \boldsymbol{I}\,,$$

  where $\boldsymbol{I}$ is the identity matrix.
- Matrix norm induced by a given vector norm

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|} \implies \|\boldsymbol{Ax}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{x}\|$$

  with sub-multiplicativity: $\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$
- Special case of interest: The 2-norm or spectral norm: $\|\boldsymbol{A}\|_2 = \sigma_1$ (largest singular value)
- The Euclidean or Frobenius norm is *not* an induced norm

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$$

  but still is sub-multiplicative: $\|\boldsymbol{AB}\|_F \leq \|\boldsymbol{A}\|_F \|\boldsymbol{B}\|_F$

Condition of solving system of linear equations

Recall that we derived the condition number $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ of a matrix $\boldsymbol{A}$

Condition of solving system of linear equations (cont'd)

Now consider the general perturbations of the data

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

One obtains the condition (proof in Quarteroni et al., Sec. 3.1)

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(\boldsymbol{A})}{1 - \kappa(\boldsymbol{A})\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|}} \left(\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} + \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \right)$$

Important practical estimate: Roundoff error in the data, with rounding unit u (recall $\approx 10^{-16}$ for double precision), produces a relative error

$$\frac{\|\delta\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \leq \frac{2}{1 - \kappa(\boldsymbol{A})u} u\kappa(\boldsymbol{A})$$

$\implies$ makes *no* sense to try to numerically solve systems with $\kappa(\boldsymbol{A}) > 10^{16}$ in double precision

Numerical solution of linear systems

There are many numerical methods for solving a system of linear equations

The most appropriate method depends on the properties of $\boldsymbol{A}$

- General dense matrices, where the entries in $\boldsymbol{A}$ are mostly non-zero and nothing special is known $\rightsquigarrow$ we focus on Gaussian elimination today
- General sparse matrices, where only a small fraction of $a_{ij} \neq 0$ (sparse typically means that $\mathcal{O}(n)$ entries are non-zero in an $n \times n$ matrix)
- Symmetric and positive-definite matrices
- Special structured sparse matrices, often arising from specific physical properties of the underlying system

It is also important to consider how many times a linear system with the same or related matrix or right-hand side needs to be solved.

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Cancellation
- Truncation errors
- Solving linear systems

Today

- Solving linear systems

Announcements

- Homework 1 was posted last week; is due next week Mon, Sep 23 *before class*

Recap: Linear systems of equations

It is said that 70-80% of computational mathematics research involves solving systems of m linear equations in n unknowns

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \,, \qquad i = 1, \ldots, m \,.$$

Linear systems arise directly from discrete models (e.g., in machine learning). Or through representing some abstract linear operator (such as a differential operator) in a finite basis as when numerically solving partial differential equations.

The common abstract way of writing systems of linear equations is

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \,,$$

with matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, right-hand side $\boldsymbol{b} \in \mathbb{R}^m$, and solution $\boldsymbol{x} \in \mathbb{R}^n$

The goal is to calculate solution $\boldsymbol{x}$ given data $\boldsymbol{A}, \boldsymbol{b}$ in a numerically stable and computationally efficient way.

Recap: The matrix inverse

- A square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is invertible or nonsingular if there exists a matrix inverse $\boldsymbol{A}^{-1} = \boldsymbol{B} \in \mathbb{R}^{n \times n}$ such that

$$\boldsymbol{AB} = \boldsymbol{BA} = \boldsymbol{I}\,,$$

  where $\boldsymbol{I}$ is the identity matrix.
- Matrix norm induced by a given vector norm

$$\|\boldsymbol{A}\| = \sup_{\boldsymbol{x} \neq \boldsymbol{0}} \frac{\|\boldsymbol{Ax}\|}{\|\boldsymbol{x}\|} \implies \|\boldsymbol{Ax}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{x}\|$$

  with sub-multiplicativity: $\|\boldsymbol{AB}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$
- Special case of interest: The 2-norm or spectral norm: $\|\boldsymbol{A}\|_2 = \sigma_1$ (largest singular value)
- The Euclidean or Frobenius norm is *not* an induced norm

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$$

  but still is sub-multiplicative: $\|\boldsymbol{AB}\|_F \leq \|\boldsymbol{A}\|_F \|\boldsymbol{B}\|_F$

Recap: Condition of solving system of linear equations (cont'd)

Now consider the general perturbations of the data

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

One obtains the condition (proof in Quarteroni et al., Sec. 3.1)

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(\boldsymbol{A})}{1 - \kappa(\boldsymbol{A})\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|}} \left(\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} + \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \right)$$

Important practical estimate: Roundoff error in the data, with rounding unit u (recall $\approx 10^{-16}$ for double precision), produces a relative error

$$\frac{\|\delta\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \leq \frac{2}{1 - \kappa(\boldsymbol{A})u} u\kappa(\boldsymbol{A})$$

$\implies$ makes *no* sense to try to numerically solve systems with $\kappa(\boldsymbol{A}) > 10^{16}$ in double precision

Recap: Numerical solution of linear systems

There are many numerical methods for solving a system of linear equations

The most appropriate method depends on the properties of $\boldsymbol{A}$

- General dense matrices, where the entries in $\boldsymbol{A}$ are mostly non-zero and nothing special is known $\rightsquigarrow$ we focus on Gaussian elimination today
- General sparse matrices, where only a small fraction of $a_{ij} \neq 0$ (sparse typically means that $\mathcal{O}(n)$ entries are non-zero in an $n \times n$ matrix)
- Symmetric and positive-definite matrices
- Special structured sparse matrices, often arising from specific physical properties of the underlying system

It is also important to consider how many times a linear system with the same or related matrix or right-hand side needs to be solved.

Gauss elimination and LU factorization

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

$\Downarrow$ eliminate

multipliers

$$\ell_{21} = a_{21}^{(1)} / a_{11}^{(1)}$$

$$\ell_{31} = a_{31}^{(1)} / a_{11}^{(1)}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} - l_{21} a_{11}^{(1)} = 0 & a_{22}^{(1)} - l_{21} a_{12}^{(1)} & a_{23}^{(1)} - l_{21} a_{13}^{(1)} \\ 0 & a_{32}^{(1)} - l_{31} a_{12}^{(1)} & a_{33}^{(1)} - l_{31} a_{13}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} - l_{21} b_1^{(1)} \\ b_3^{(1)} - l_{31} b_1^{(1)} \end{bmatrix}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ & a_{22}^{(2)} & a_{23}^{(2)} \\ & 0 & a_{33}^{(3)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}$$

$$a_{33}^{(3)} = a_{33}^{(2)} - a_{22}^{(2)} \cdot l_{32}$$

$$b_3^{(3)} = b_3^{(2)} - b_2^{(2)} \cdot l_{32}$$

Upper triangular system

$$x_3 = b_3^{(3)} / a_{33}^{(3)} \Rightarrow \text{eliminate } x_3$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} \\ 0 & a_{22}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} - a_{13}^{(1)} x_3 \\ b_2^{(2)} - a_{23}^{(2)} x_3 \end{bmatrix}$$

solve $x_2 = \dfrac{b_2^{(2)} - a_{23}^{(2)} x_3}{a_{22}^{(2)}}$

$\Rightarrow x_1 = \cdots$

<u>Idea</u>: store the multipliers in the lower triangle of A

Matrix at step k

$$\begin{bmatrix} & U^{(k)} \\ L^{(k)} & A^{(k)} \end{bmatrix}$$

$k=2$

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & a_{22}^{(2)} & a_{23}^{(2)} \\ l_{31} & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix}$$

$$\begin{array}{c} U \\ \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & u_{22} & u_{23} \\ l_{31} & l_{32} & u_{33} \end{bmatrix} \\ L \end{array}$$

$$A = LU$$

$$\overset{A}{\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix}} \overset{x}{\begin{bmatrix} \\ \\ \\ \end{bmatrix}} = \overset{b}{\begin{bmatrix} 5 \\ 6 \\ 13 \end{bmatrix}}$$

$$\ell_{21} = \frac{2}{1}$$

$$\ell_{31} = \frac{3}{1}$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 0 & -4 \\ 3 & 3 & -5 \end{bmatrix}$$

swap rows

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & 3 & -5 \\ 2 & 0 & -4 \end{bmatrix}$$

Gauss elimination in Matlab

```
function A = mylu(A) % In-place LU factorization
% need square matrix
[n, m] = size(A);
assert(n == m);
for k=1:(n-1) % for variable x(k)
    % Assumed A(k, k) non-zero and then
    % calculate multipliers in column k
    A((k + 1):n, k) = A((k + 1):n, k)/A(k, k);
    for j = (k + 1):n
        % eliminate variable x(k)
        A((k + 1):n, j) = A((k + 1):n, j) - A((k + 1):
            n, k)*A(k, j);
    end
end
end
```

- Gaussian elimination is a general method for dense matrices and is commonly used
- Implementing Gaussian elimination efficiently is difficult and we will not discuss it $\rightsquigarrow$ course on HPC
- The LAPACK public-domain library is the main repository for excellent implementations of dense linear solvers
- Matlab (and numpy) use highly optimized variants of GEM by default, mostly based on LAPACK
- Matlab (and numpy) have specialized solvers for special cases of matrices, so always check help pages!

Problem?

Problem?

```
>> A = [1 1 3; 2 2 2; 3 6 4]

A =

     1     1     3
     2     2     2
     3     6     4

>> mylu(A)

ans =

     1     1     3
     2     0    -4
     3   Inf   Inf
```

LU with pivoting

Zero diagonal entries (pivots) pose a problem $\rightsquigarrow$ pivoting by swapping rows

$\rightsquigarrow$ board

$$\begin{bmatrix} 1 & 1 & 3 \\ 3 & 3 & -5 \\ 2 & 0 & 1 \end{bmatrix} \qquad \begin{matrix} 1 & & \\ & 1 & \\ & & -4 \end{matrix}$$

$$LU = PA$$

permutation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
>> A = [1 1 3; 2 2 2; 3 6 4];
>> P = [1 0 0; 0 0 1; 0 1 0] % swap row 2 and 3

P =

     1     0     0
     0     0     1
     0     1     0

>> mylu(P*A)

ans =

     1     1     3
     3     3    -5
     2     0    -4
```

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures exists of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- **Q:** What else could pivoting be useful for?
  $\rightsquigarrow$ let's see what Matlab does

```
>> [L, U, P] = lu(A) % built-in lu
L =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000
U =
    3.0000    6.0000    4.0000
         0   -2.0000   -0.6667
         0         0    2.0000
P =
     0     0     1
     0     1     0
     1     0     0
>> norm(L*U - P*A)
ans = 0
```

```
>> [L, U, P] = lu(A) % built-in lu
L =
    1.0000         0         0
    0.6667    1.0000         0
    0.3333    0.5000    1.0000
U =
    3.0000    6.0000    4.0000
         0   -2.0000   -0.6667
         0         0    2.0000
P =
     0     0     1
     0     1     0
     1     0     0
>> norm(L*U - P*A)
ans = 0
```

Reverses order of rows rather than just swapping 2 and 3.
Leads to entries of L with magnitude ≤ 1

$\rightsquigarrow$ board

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}$$

$$\ell_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}} = \frac{1}{10^{-20}} = 10^{20}$$

$$\begin{bmatrix} 10^{-20} & 1 \\ \boxed{\frac{10^{-20}}{10^{20}}} & \nearrow \end{bmatrix} = \begin{bmatrix} 10^{-20} & 1 \\ 10^{20} & 1-10^{20} \end{bmatrix}$$

$$1 - 1 \cdot 10^{20}$$
$$= 1 - 10^{20}$$

$$L = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1-10^{20} \end{bmatrix}$$

$$LU = A$$

floating point: $1 - 10^{20} \ominus -10^{20}$ ↯

⇝ board

```
>> A = [1e-20 1; 1 1]

A =

    1.0000e-20   1.0000e+00
    1.0000e+00   1.0000e+00

>> LUmat = mylu(A);
L = [1 0; LUmat(2, 1) 1];
U = LUmat; U(2, 1) = 0;
L*U

ans =

    1.0000e-20   1.0000e+00
    1.0000e+00            0
```

```
>> [L, U, P] = lu(A)
L =
   1.0000e+00             0
   1.0000e-20   1.0000e+00
U =
     1     1
     0     1
P =
     0     1
     1     0
>> P'*L*U
ans =
   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00
>> A
A =
   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00
```

Instability of LU decomposition *without* pivoting

If $\boldsymbol{A}$ has an $\boldsymbol{LU}$ factorization, then the computed $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$ obtained in floating-point arithmetic with Gaussian elimination satisfy $\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} = \boldsymbol{A} + \delta\boldsymbol{A}$ with the bound

$$\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{L}\|\|\boldsymbol{U}\|} \in \mathcal{O}(u)\,,$$

where u is the roundoff unit.

- Notice that we would have liked to bound $\|\delta\boldsymbol{A}\|/\|\boldsymbol{A}\|$ but we got $\|\delta\boldsymbol{A}\|/(\|\boldsymbol{L}\|\|\boldsymbol{U}\|)$
- Thus, for matrices with $\|\boldsymbol{L}\|\|\boldsymbol{U}\| \approx \|\boldsymbol{A}\|$, the algorithm will show stable behavior
- However, if $\|\boldsymbol{L}\|\|\boldsymbol{U}\| \not\approx \|\boldsymbol{A}\|$, then we can get an unstable result

$\rightsquigarrow$ Gaussian elimination is *not* stable in general

Example $\rightsquigarrow$ board

$$\boldsymbol{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \boldsymbol{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{bmatrix}$$

Compare $\|\boldsymbol{L}\|_\infty\|\boldsymbol{U}\|_\infty$ and $\|\boldsymbol{A}\|_\infty$

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1-10^{20} \end{bmatrix}$$

Recall: operator norm induced by

$$\|x\|_\infty = \max_i |x_i|$$

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

$$\|A\|_\infty = 2$$

$$\|L\|_\infty = 1 + 10^{20}$$

$$\|U\|_\infty$$

$$\|L\|_\infty \|U\|_\infty \gg \|A\|_\infty$$

$$\approx 10^{20}$$

LU with row pivoting to maximum element

```
function [A, P] = myplu(A) % In-place LU factorization
[n, m] = size(A); P = eye(n);
for k=1:(n-1) % for variable x(k)
    [~, selI] = max(abs(A(k:n, k))); % select pivot
    c = A(k, k:end); d = P(k, :);
    A(k, k:end) = A(selI + (k-1), k:end); P(k, :) = P(selI + (k-1), :);
    A(selI + (k-1), k:end) = c; P(selI + k-1, :) = d;
    % calculate multipliers in column k
    A((k + 1):n, k) = A((k + 1):n, k)/A(k, k);
    for j = (k + 1):n % eliminate variable x(k)
        A((k + 1):n, j) = A((k + 1):n, j) - A((k + 1):n, k)*A(k, j);
    end
end
end
```

```
>> A = [1e-20 1; 1 1]
[LUmat, P] = myplu(A);
L = [1 0; LUmat(2, 1) 1];
U = LUmat; U(2, 1) = 0;
P'*L*U

A =

   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00


ans =

   1.0000e-20   1.0000e+00
   1.0000e+00   1.0000e+00
```

Stability of Gaussian elimination with pivoting

For $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$, introduce the growth factor

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}$$

where u_{ij} is the i, j-th element of $\boldsymbol{U}$

Consider the factorization $\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ with partial row pivoting w.r.t. taking the maximum element for a matrix $\boldsymbol{A}$ of dimension $n \times n$. Gaussian elimination gives $\tilde{\boldsymbol{P}}, \tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}$ that satisfy

$$\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} = \tilde{\boldsymbol{P}}\boldsymbol{A} + \delta\boldsymbol{A}\,, \qquad \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \in \mathcal{O}(\rho u)\,,$$

where u is the roundoff unit. If all off-diagonal entries of $\boldsymbol{L}$ are < 1, implying that there are no ties in the selection of pivots in exact arithmetic, then $\boldsymbol{P} = \tilde{\boldsymbol{P}}$ for sufficiently small u.

This means that Gaussian elimination with partial pivoting is backward stable if ρ holds uniformly for matrices with $n \times n$.

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures exists of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- Furthermore, pivoting (w.r.t. $\max |a_{ij}|$) leads to a backward stable algorithm. **However**, the growth factor ρ can be huge and grow with the dimension of $\boldsymbol{A}$! Fortunately, large factors ρ "never seem to appear in real applications." (Trefethen & Bau, Chapter 22)

- There also is full pivoting (rows + columns)

$$\boldsymbol{PAQ} = \boldsymbol{LU}$$

  to further increases stability but it usually is not worth it in practice (higher costs to search for pivoting element over rows and columns but little improvement in terms of stability)

Solving linear systems

- Once an LU factorization is available, solving a linear system is cheap:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{L}\boldsymbol{U}\boldsymbol{x} = \boldsymbol{L}(\boldsymbol{U}\boldsymbol{x}) = \boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$$

- Solve for $\boldsymbol{y}$ using *forward substitution*
- Solve for $\boldsymbol{x}$ by using *backward substitution* $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$

What is *forward/backward substitution*? $\rightsquigarrow$ board

$$\begin{bmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & \cdots & \cdots & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}$$

Forward substitution

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 \cdots & 0 \\ \vdots & & & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$x_1 = b_1 / l_{11}$$

$$x_2 = (b_2 - l_{21} x_1) / l_{22}$$

$$\vdots$$

$$x_n = (b_n - l_{n1} x_1 - \cdots \quad l_{n,n-1} x_{n-1}) / l_{nn}$$

- If row pivoting is used, the same process works by also permuting the right-hand side $\boldsymbol{b}$

$$\boldsymbol{PAx} = \boldsymbol{LUx} = \boldsymbol{Ly} = \boldsymbol{Pb}$$

or formally (**never implement inverse for solving linear systems of equations**)

$$\boldsymbol{x} = (\boldsymbol{LU})^{-1}\boldsymbol{Pb} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{Pb}$$

- Because $\boldsymbol{P}$ is orthonormal, we have $\boldsymbol{P}^{-1} = \boldsymbol{P}^T$ and thus

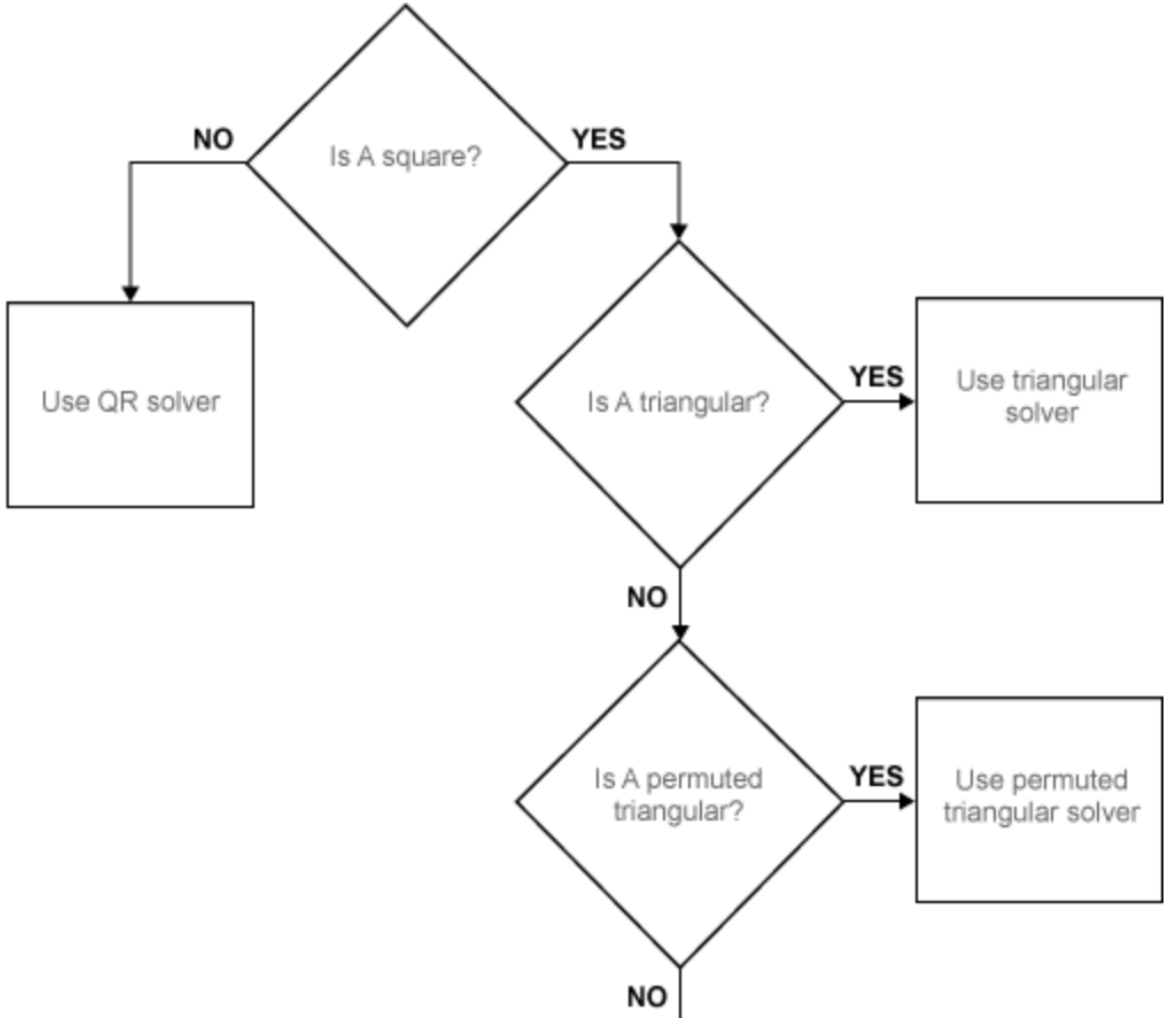
$$\boldsymbol{A} = \boldsymbol{P}^{-1}\boldsymbol{LU} = (\boldsymbol{P}^T\boldsymbol{L})\boldsymbol{U} = \tilde{\boldsymbol{L}}\boldsymbol{U}\,,$$

with $\tilde{\boldsymbol{L}}$ a row permutation of a unit lower triangular matrix

In Matlab, the backslash operator solves linear systems (see help `mldivide`)

```
A = [1 2 3; 4 5 6; 7 8 0];
b = [2; 1; -1];
x = A\b; x'
[L, U] = lu(A)
y = L\b; x = U\y; x'
ans =
  -2.5556e+00   2.1111e+00   1.1111e-01
L =
   1.4286e-01   1.0000e+00            0
   5.7143e-01   5.0000e-01   1.0000e+00
   1.0000e+00            0            0
U =
   7.0000e+00   8.0000e+00            0
            0   8.5714e-01   3.0000e+00
            0            0   4.5000e+00
ans =
  -2.5556e+00   2.1111e+00   1.1111e-01
```

Is the permuted triangular matrix $\tilde{\boldsymbol{L}}$ (which we get from `[L, U] = lu(A)`) detected as such? Yes!



Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Solving linear systems
- LU decomposition
- Pivoting

Today

- Cost analysis of LU decomposition
- Solving linear systems with sparse matrices
- Least-squares problems

Announcements

- Homework 2 has been posted; is due next week Mon, Oct 7 *before class*

Recap: Condition of solving system of linear equations (cont'd)

Now consider the general perturbations of the data

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

One obtains the condition (proof in Quarteroni et al., Sec. 3.1)

$$\frac{\|\delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(\boldsymbol{A})}{1 - \kappa(\boldsymbol{A})\frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|}} \left(\frac{\|\delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|} + \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \right)$$

Important practical estimate: Roundoff error in the data, with rounding unit u (recall $\approx 10^{-16}$ for double precision), produces a relative error

$$\frac{\|\delta\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \leq \frac{2}{1 - \kappa(\boldsymbol{A})u} u\kappa(\boldsymbol{A})$$

$\implies$ makes *no* sense to try to numerically solve systems with $\kappa(\boldsymbol{A}) > 10^{16}$ in double precision

Recap: LU decomposition

Gauss elimination and LU factorization

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

index of step

multiply first row by l_{11} and subtract from 2nd row $\longrightarrow$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(1)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}$$

$$l_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$$

eliminate using

$$l_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}$$

$$\longrightarrow \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}$$

Idea: Store multipliers l_{21}, l_{31}, l_{32} in matrix:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21} & u_{22} & u_{23} \\ l_{31} & l_{32} & u_{33} \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} U \\ L \end{bmatrix}$$

$$A = LU, \quad L = \begin{bmatrix} 1 & & \\ l_{21} & 1 & \\ l_{31} & l_{32} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & \cdots & u_{13} \\ 0 & & \vdots \\ 0 & 0 & \end{bmatrix}$$

Recap: Pivoting

LU with pivoting

Zero diagonal entries (pivots) pose a problem $\rightsquigarrow$ pivoting by swapping rows

$\rightsquigarrow$ board $\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix}$ $\xrightarrow{\text{Gauss elimination}}$ $\begin{bmatrix} 1 & 1 & 3 \\ l_{21}=2 & 0 & -4 \\ l_{31}=3 & 3 & -5 \end{bmatrix}$

Problem! Cannot divide by zero, pivot element is zero.

$\xrightarrow[2 \,\&\, 3]{\text{Swap rows}}$ $\begin{bmatrix} 1 & 1 & 3 \\ 3 & 3 & -5 \\ 2 & 0 & -4 \end{bmatrix}$

done – In general, use 3 as pivot.

$$\boxed{LU = PA}$$

P.... permutation matrix.

$P_2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

Theorem: This always exists with appropriate perm. matrix P

Recap: Stability of Gaussian elimination with pivoting

For $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$, introduce the growth factor

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}$$

where u_{ij} is the i, j-th element of $\boldsymbol{U}$

Consider the factorization $\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ with partial row pivoting w.r.t. taking the maximum element for a matrix $\boldsymbol{A}$ of dimension $n \times n$. Gaussian elimination gives $\tilde{\boldsymbol{P}}, \tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}$ that satisfy

$$\tilde{\boldsymbol{L}}\tilde{\boldsymbol{U}} = \tilde{\boldsymbol{P}}\boldsymbol{A} + \delta\boldsymbol{A}\,, \qquad \frac{\|\delta\boldsymbol{A}\|}{\|\boldsymbol{A}\|} \in \mathcal{O}(\rho u)\,,$$

where u is the roundoff unit. If all off-diagonal entries of $\boldsymbol{L}$ are < 1, implying that there are no ties in the selection of pivots in exact arithmetic, then $\boldsymbol{P} = \tilde{\boldsymbol{P}}$ for sufficiently small u.

This means that Gaussian elimination with partial pivoting is backward stable if ρ holds uniformly for matrices with $n \times n$.

Recap: LU with row pivoting to maximum element

```
function [A, P] = myplu(A) % In-place LU factorization
[n, m] = size(A); P = eye(n);
for k=1:(n-1) % for variable x(k)
    [~, selI] = max(abs(A(k:n, k))); % select pivot
    c = A(k, k:end); d = P(k, :);
    A(k, k:end) = A(selI + (k-1), k:end); P(k, :) = P(selI + (k-1), :);
    A(selI + (k-1), k:end) = c; P(selI + k-1, :) = d;
    % calculate multipliers in column k
    A((k + 1):n, k) = A((k + 1):n, k)/A(k, k);
    for j = (k + 1):n % eliminate variable x(k)
        A((k + 1):n, j) = A((k + 1):n, j) - A((k + 1):n, k)*A(k, j);
    end
end
end
```

Costs: Forward/backward substitution

$\rightsquigarrow$ board

Forward substitution

$$\begin{bmatrix} l_{11} & 0 & \cdots & & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ \vdots & & & & \\ l_{n1} & l_{n2} & & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$x_1 = b_1 / l_{11}$ [$\rightsquigarrow$ 1 division]

$x_2 = (b_2 - l_{21} x_1) / l_{22}$ [$\rightsquigarrow$ 1 div, 1 mul, 1 add]

$\vdots$ $\vdots$

$x_n = (b_n - l_{n1} x_1 - \cdots l_{n,n-1} x_{n-1}) / l_{nn}$

[$\rightsquigarrow$ 1 div, $(n-1)\times$ mul, $(n-1)\times$ add]

add and multiplications

$$\sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

divisions: n

total FLOPs: $n + 2\frac{n(n-1)}{2} \in \mathcal{O}(n^2)$

Costs: Forward/backward substitution

$\rightsquigarrow$ board

Forward substitution requires

$$\frac{n(n-1)}{2} \text{ multiplications/additions,}$$
$$n \text{ divisons.}$$

Overall: $\sim n^2$ floating point operations (flops) $\rightsquigarrow$ costs scale as $\mathcal{O}(n^2)$

Similarly, backward substitution has costs that scale as $\mathcal{O}(n^2)$

We count flops to estimate the computational time/effort. What else matters?

Costs: Forward/backward substitution

$\rightsquigarrow$ board

Forward substitution requires

$$\frac{n(n-1)}{2} \text{ multiplications/additions,}$$
$$n \text{ divisons.}$$

Overall: $\sim n^2$ floating point operations (flops) $\rightsquigarrow$ costs scale as $\mathcal{O}(n^2)$

Similarly, backward substitution has costs that scale as $\mathcal{O}(n^2)$

We count flops to estimate the computational time/effort. What else matters? Besides floating point operations, computer memory access has a significant influence on the efficiency of numerical methods.

Costs

For forward [backward] substitution at step k there are $\approx k$ $[(n-k)]$ multiplications and subtractions plus a few divisions. The total over all n steps is

$$\sum_{k=1}^{n} k \in \mathcal{O}(n^2)$$

$\rightsquigarrow$ the number of floating-point operations (FLOPs) scales as $\mathcal{O}(n^2)$

For Gaussian elimination, at step k, there are $\approx (n-k)^2$ operations. Thus, the total scales as

$$\sum_{k=1}^{n} (n-k)^2 \in \mathcal{O}(n^3)$$

Directly applying Gaussian elimination scales as

Directly applying Gaussian elimination scales as

$$\mathcal{O}(n^3)$$

Computing LU decomposition scales as

Directly applying Gaussian elimination scales as

$$\mathcal{O}(n^3)$$

Computing LU decomposition scales as

$$\mathcal{O}(n^3)$$

Forward/backward substitution scales as

Directly applying Gaussian elimination scales as

$$\mathcal{O}(n^3)$$

Computing LU decomposition scales as

$$\mathcal{O}(n^3)$$

Forward/backward substitution scales as

$$\mathcal{O}(n^2)$$

LU + forward/backward scales as

Directly applying Gaussian elimination scales as

$$\mathcal{O}(n^3)$$

Computing LU decomposition scales as

$$\mathcal{O}(n^3)$$

Forward/backward substitution scales as

$$\mathcal{O}(n^2)$$

LU + forward/backward scales as

$$\mathcal{O}(n^3)$$

$\rightsquigarrow$ why useful?

Directly applying Gaussian elimination scales as

$$\mathcal{O}(n^3)$$

Computing LU decomposition scales as

$$\mathcal{O}(n^3)$$

Forward/backward substitution scales as

$$\mathcal{O}(n^2)$$

LU + forward/backward scales as

$$\mathcal{O}(n^3)$$

$\rightsquigarrow$ why useful? can *reuse* LU for other b

Choleski factorization

A matrix is symmetric positive definite (spd), if $A = A^T$ and for all $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq 0$, the inner product $\langle A\boldsymbol{x}, \boldsymbol{x} \rangle > 0$.

For spd matrices, we can compute the factorization:

$$A = LDL^T,$$

where L is a lower triangular matrix with 1's on the diagonal, and D is a positive diagonal matrix.

The Choleski factorization is obtained by multiplying the square root of D (which exists!) with L:

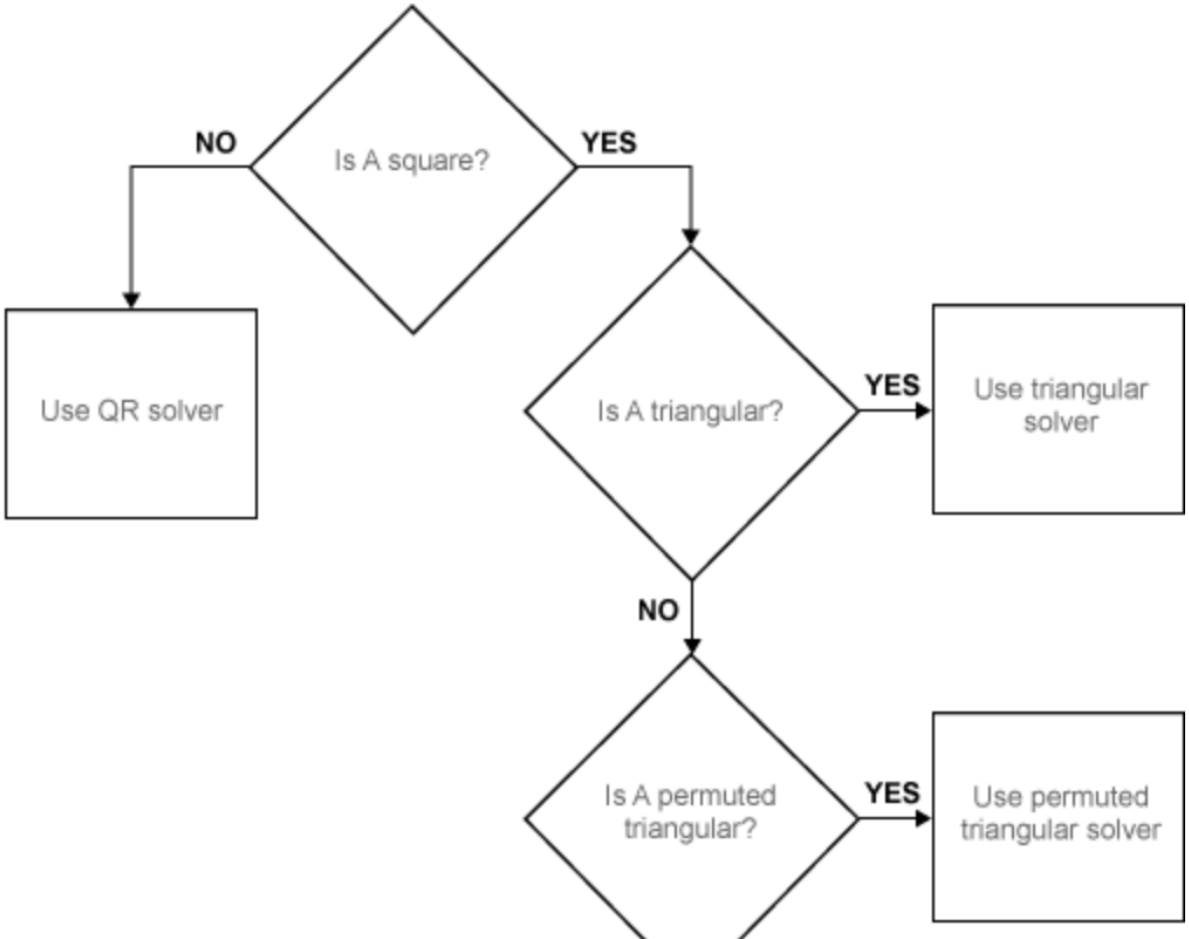
$$A = \bar{L}\bar{L}^T.$$

Algorithms for Choleski factorization are about twice as fast as Gaussian elimination but also scale as $\mathcal{O}(n^3)$.

```
>> A = randn(1000, 1000)*diag(linspace(1, 10, 1000))*
   randn(1000, 1000); A = A'*A;
>> tic; chol(A); toc
Elapsed time is 0.004863 seconds.
>> tic; lu(A); toc
Elapsed time is 0.010114 seconds.
```

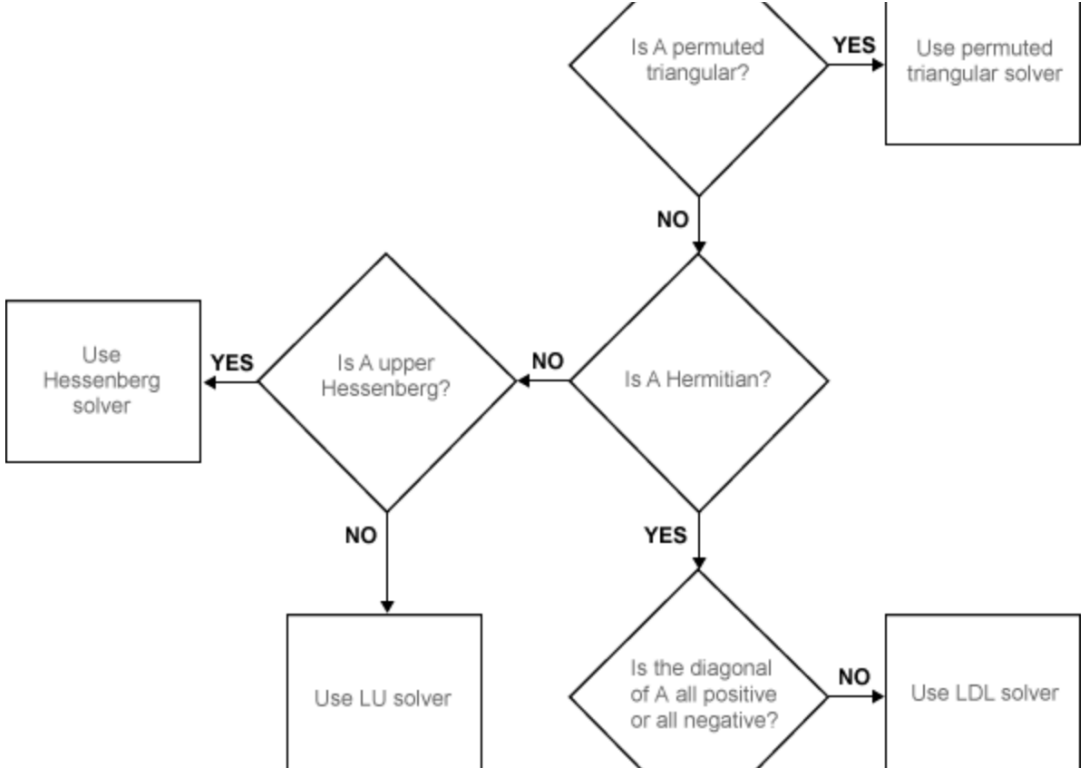
Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



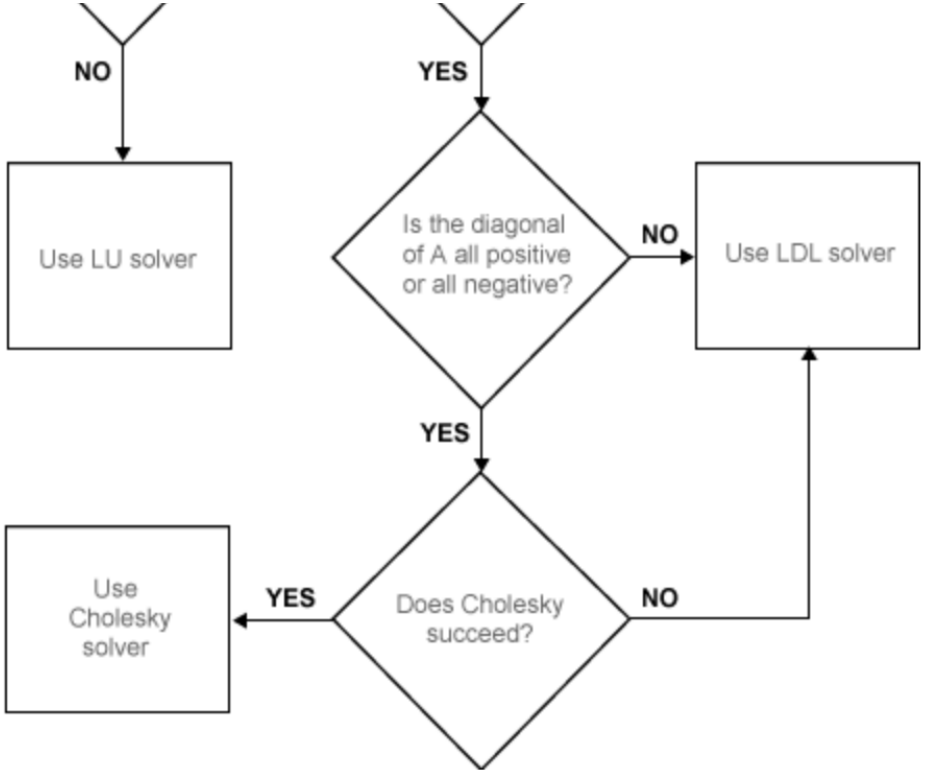
Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



Special matrices in Matlab

Matlab tests for special matrices automatically and chooses a good decomposition/solver



Conclusions/summary

- The condition of solving a linear system $\boldsymbol{Ax} = \boldsymbol{b}$ is determined by the condition number of the matrix $\boldsymbol{A}$
$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\| \geq 1$$
- Gaussian elimination can be used to solve general square linear systems and produces a factorization, if it exists
$$\boldsymbol{A} = \boldsymbol{LU}$$
- Partial pivoting is sufficient for existence and stability of the LU decomposition
$$\boldsymbol{PA} = \boldsymbol{LU}\,, \qquad \boldsymbol{A} = \tilde{\boldsymbol{L}}\boldsymbol{U}$$
- The Cholesky factorization $\boldsymbol{A} = \boldsymbol{LL}^T$ exists if $\boldsymbol{A}$ is spd and then it is the better choice (cheaper) than LU
- Rely on the highly optimized routines in Matlab (LAPACK) and other software packages than implementing these algorithms yourself $\rightsquigarrow$ take the course on HPC next spring to learn more about the efficient *implementation* of these algorithms

Sparse matrices

Sparse matrix

- A matrix where a substantial fraction of the entries are zero is called a sparse matrix. Typically, only $\mathcal{O}(N)$ non-zero entries in an $N \times N$ matrix are allowed for sparse algorithms to show benefit over dense linear algebra routines.

- If we have only $\mathcal{O}(N)$ non-zero entries, then store only those; in contrast to dense matrices. Exploiting sparsity is important (life saving) for large matrices

- The structure of a sparse matrix refers to the set of indices i, j such that $|a_{ij}| > 0$ and is visualized in Matlab with `spy`

- The structure of sparse matrices comes from the nature of the problem, e.g., in an inter-city road transportation problem it corresponds to the pairs of cities connected by a road.

- In fact, just counting the number of non-zero elements is not enough: the sparsity structure is the most important property that determines whether an efficient method exists

Banded matrices

Banded matrices are a very special but common type of sparse matrices, e.g., tridiagonal matrices

$$\begin{bmatrix} a_1 & c_1 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & b_n & a_n \end{bmatrix}$$

For example, think of the Laplace problem $u''(x) = f(x)$ on the unit interval and a finite-difference discretization

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

on an equidistant grid. This leads to a system of equations with a tridiagonal matrix
$\rightsquigarrow$ Numerical Methods II (Spring semester)

There exist special techniques for banded matrices that are much faster than the general case, e.g., only $8n$ FLOPs

Decomposing sparse matrices

There also are general methods for dealing with sparse matrices, such as the sparse LU factorization.

How well they work depends on the structure of the matrix. What could go wrong?

Decomposing sparse matrices

There also are general methods for dealing with sparse matrices, such as the sparse LU factorization.

How well they work depends on the structure of the matrix. What could go wrong?

When factorizing sparse matrices, the factors, e.g., L and U, can be much less sparse than $A \rightsquigarrow$ fill-in

For many sparse matrices, there is a large fill-in

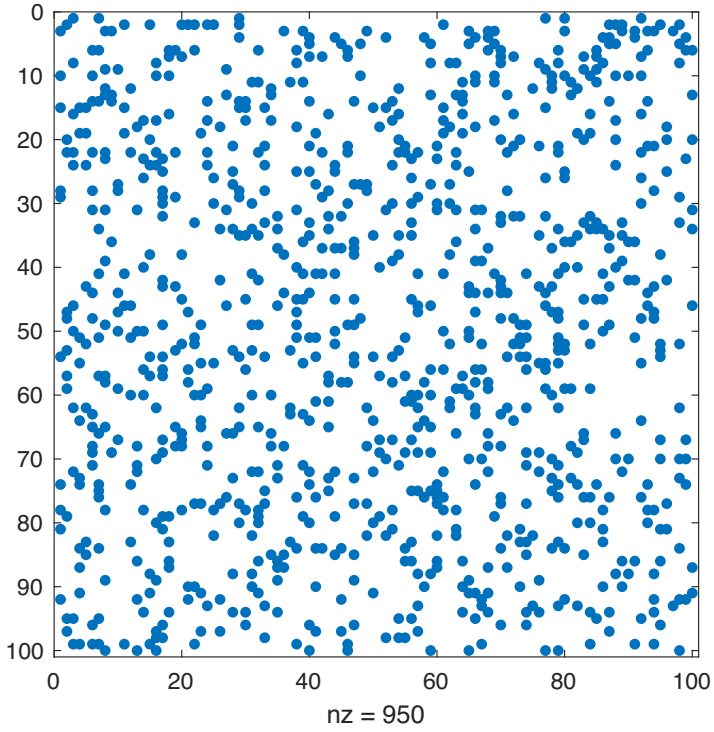
- Pivoting can help to reduce fill-in
- However, often "good" pivoting for sparsity leads to less stable behavior and vice versa

Sparse matrices in Matlab

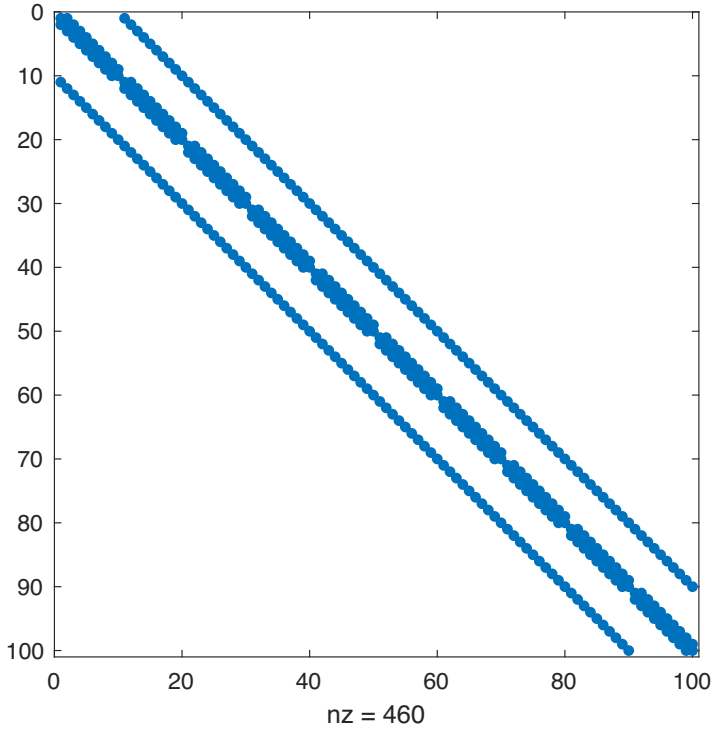
```
% S = sparse(i,j,v) generates a sparse matrix S
% from the triplets i, j, v with S(i(k),j(k)) = v(k).
>> A = sparse([1 2 2 4 4], [3 1 4 2 3], 1:5)
A =
   (2,1)        2
   (4,2)        4
   (1,3)        1
   (4,3)        5
   (2,4)        3
>> whos A
  Name      Size                Bytes  Class
     Attributes
  A         4x4                   120  double    sparse
>> nnz(A)
ans =
     5
```

```
% S = sparse(i,j,v,m,n,nz) allocates space for nz nonzero elements.
% Use this syntax to allocate extra space for nonzero values to be filled in after construction.
>> A = sparse([], [], [], 4, 4, 5);
>> A(2, 1) = 2; A(4, 2) = 4; A(1, 3) = 1; A(4, 3) = 5; A(2, 4) = 3;
>> full(A)

ans =

     0     0     1     0
     2     0     0     3
     0     0     0     0
     0     4     5     0
```

```
% generate a random sparse matrix with density 10% and size 100x100
>> B = sprand(100, 100, 0.1);
% the sparse block tridiagonal matrix of order n^2 resulting from discretizing Poisson's equation with the 5-point operator on an n-by-n mesh.
>> X = gallery('poisson', 10);
>> spy(B);
>> spy(X);
```

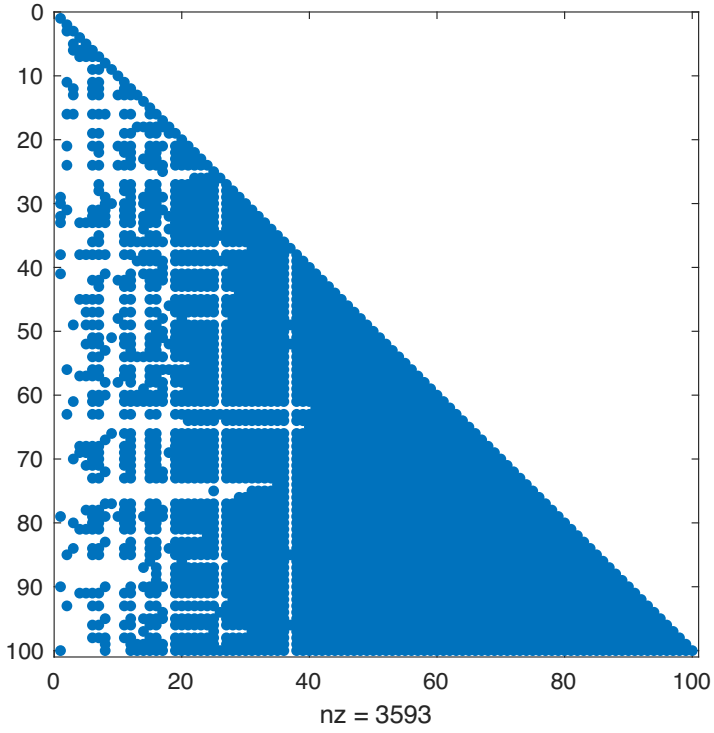


(a) matrix B

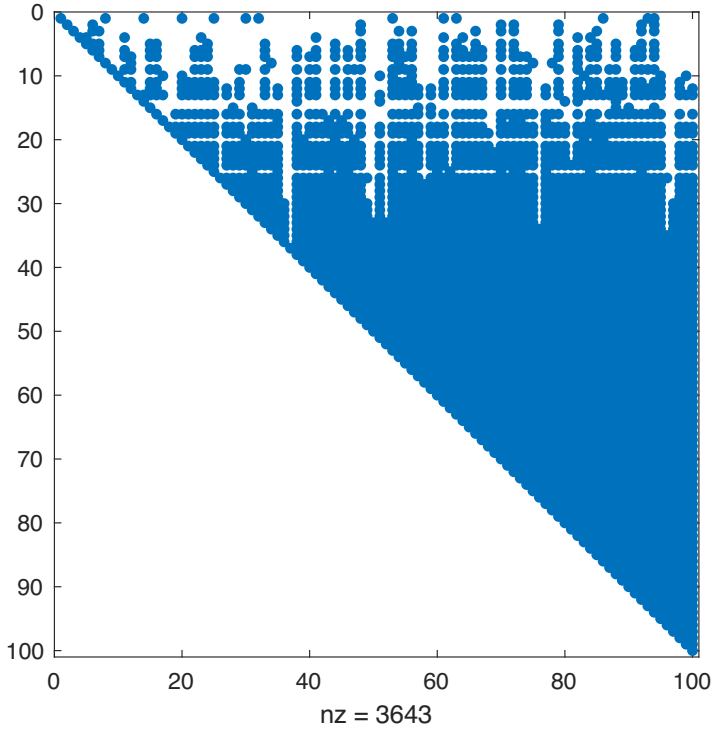


(b) matrix X

```
>> [L, U, P] = lu(B);
>> spy(L);
>> spy(U);

>> [L, U, P] = lu(X);
>> spy(L);
>> spy(U);
```

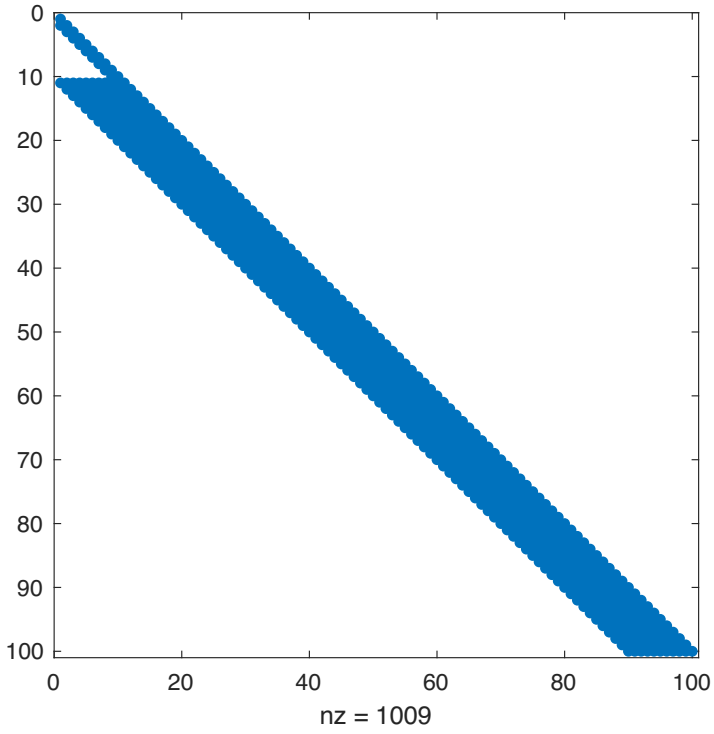


(a) factor L of B

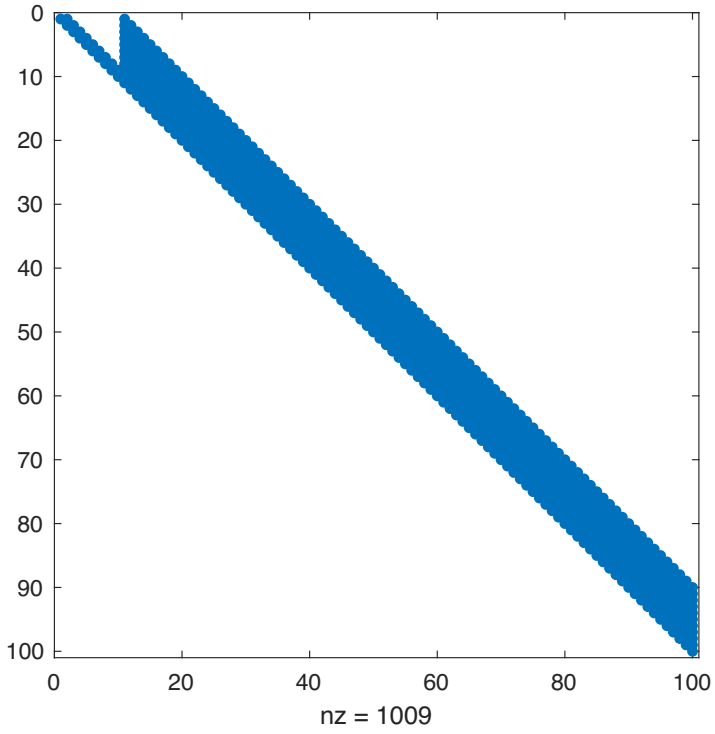


(b) factor U of B

A lot of fill-in! Factors L and U are not sparse, even though matrix B is sparse



(a) factor L of X



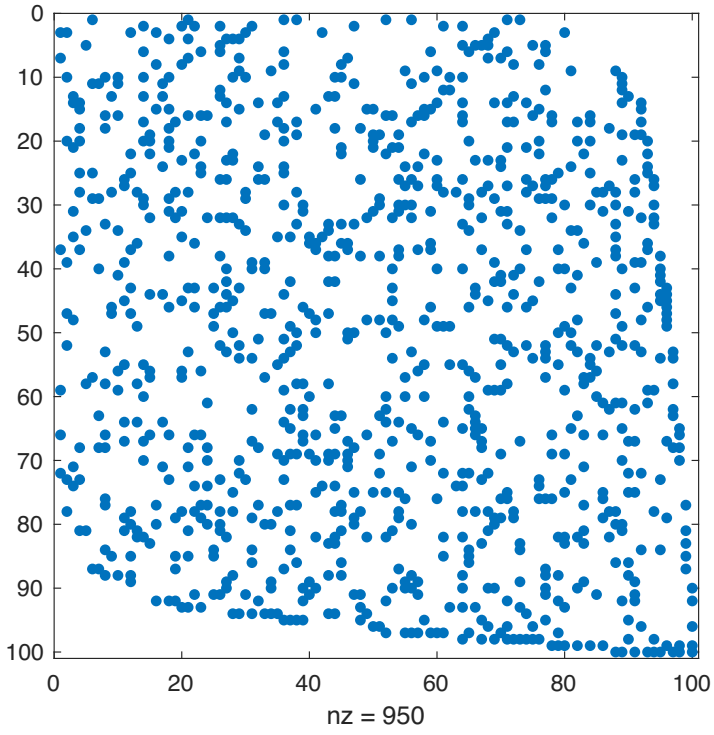
(b) factor U of X

Though better than for matrix B with random sparsity structure, there still are many more non-zero entries in the factors of X than in X itself.

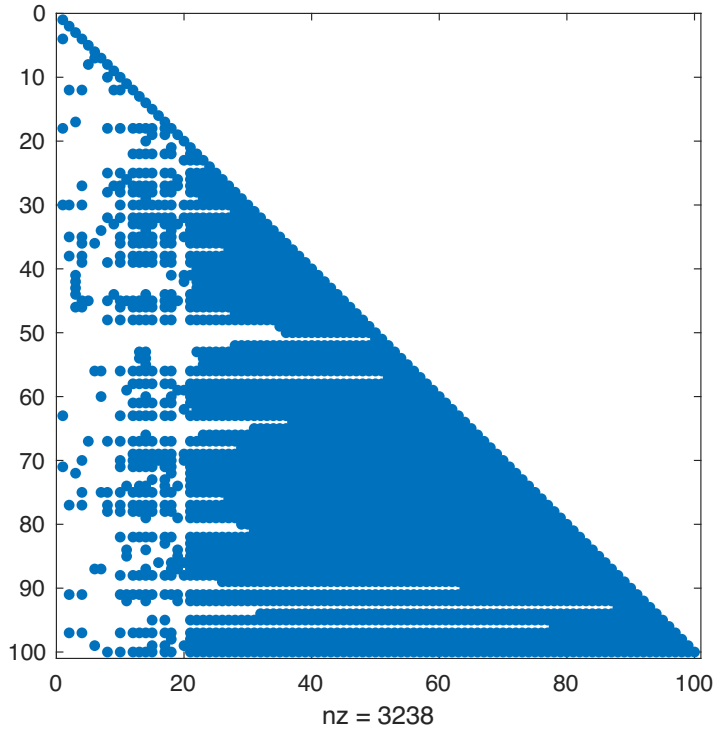
Changing the sparsity structure via re-ordering the matrix can help to reduce fill-in. For example, in Matlab the sparse reverse Cuthill-McKee ordering is implemented.

The re-ordered matrix tends to have its nonzero elements closer to the diagonal. This is a good preordering for LU or Cholesky factorization of matrices.

```
>> p = symrcm(B);
>> spy(B(p, p));
>> [L, U, P] = lu(B(p, p));
>> spy(L);
>> spy(U);

>> p = symrcm(X)
>> spy(X(p, p));
>> [L, U, P] = lu(X(p, p));
>> spy(L);
>> spy(U);
```

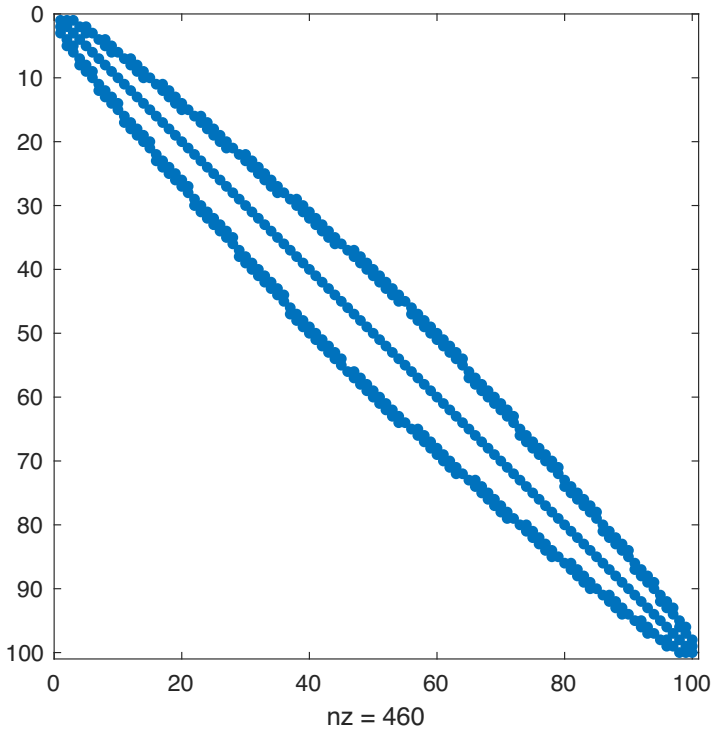


(a) re-ordered B matrix

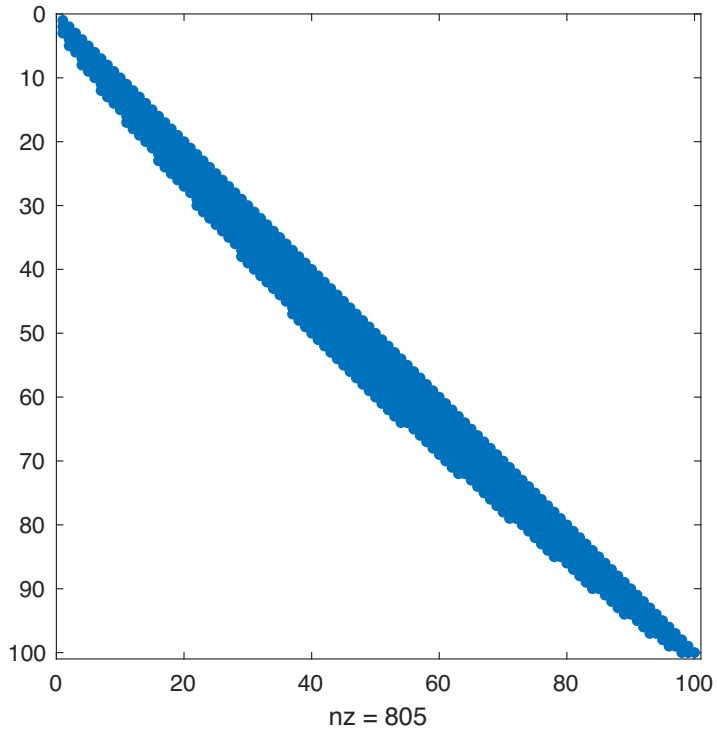


(b) factor L of re-ordered B

- Notice how the non-zero elements tend to be closer to the diagonal of the re-ordered B compared to the original B
- The fill-in is reduced from ~ 3500 to ~ 3200 non-zero entries; reduction of $< 10\%$
- It is hard to find a good ordering for matrix with a random sparsity structure



(a) re-ordered X matrix



(b) factor L of re-ordered X

- Non-zero entries of re-ordered X are closer to the diagonal than for the original X
- Fill-in is reduced by a roughly 20% from ~ 1000 to ~ 800 non-zero entries

Conclusions/summary

While there are general techniques for dealing with sparse matrices that help greatly, it all depends on the structure of the matrix

Pivoting has a dual, sometimes conflicting goal:

1. Reduce fill-in, i.e., improve memory use: Still active subject of research!
2. Reduce roundoff errors, i.e., improve stability. Typically some threshold pivoting is used only when needed

For many sparse matrices *iterative methods* (later) are required when large fill-in.

Feedback cards

Feedback cards

- Upload slides before class

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board
- Define terms clearly instead of assuming everyone knows

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board
- Define terms clearly instead of assuming everyone knows
- More intuitive explanations and motivations

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board
- Define terms clearly instead of assuming everyone knows
- More intuitive explanations and motivations
- Show examples in Matlab rather than just on slides

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board
- Define terms clearly instead of assuming everyone knows
- More intuitive explanations and motivations
- Show examples in Matlab rather than just on slides
- TA sessions

Feedback cards

- Upload slides before class
- Neater handwriting and going slower when writing on board
- Define terms clearly instead of assuming everyone knows
- More intuitive explanations and motivations
- Show examples in Matlab rather than just on slides
- TA sessions
- Code examples are interesting

Least-squares problems

Least-squares problems

Given data points/measurements

$$(t_i, b_i), \quad i = 1, \ldots, m$$

and a model function ϕ that relates t and b:

$$b = \phi(t; x_1, \ldots, x_n),$$

where $x_1, \ldots, x_n$ are model function parameters. If the model is supposed to describe the data, the deviations/errors

$$\Delta_i = b_i - \phi(t_i, x_1, \ldots, x_n)$$

should be small. Thus, to fit the model to the measurements, one must choose $x_1, \ldots, x_n$ appropriately.

Least-squares problems

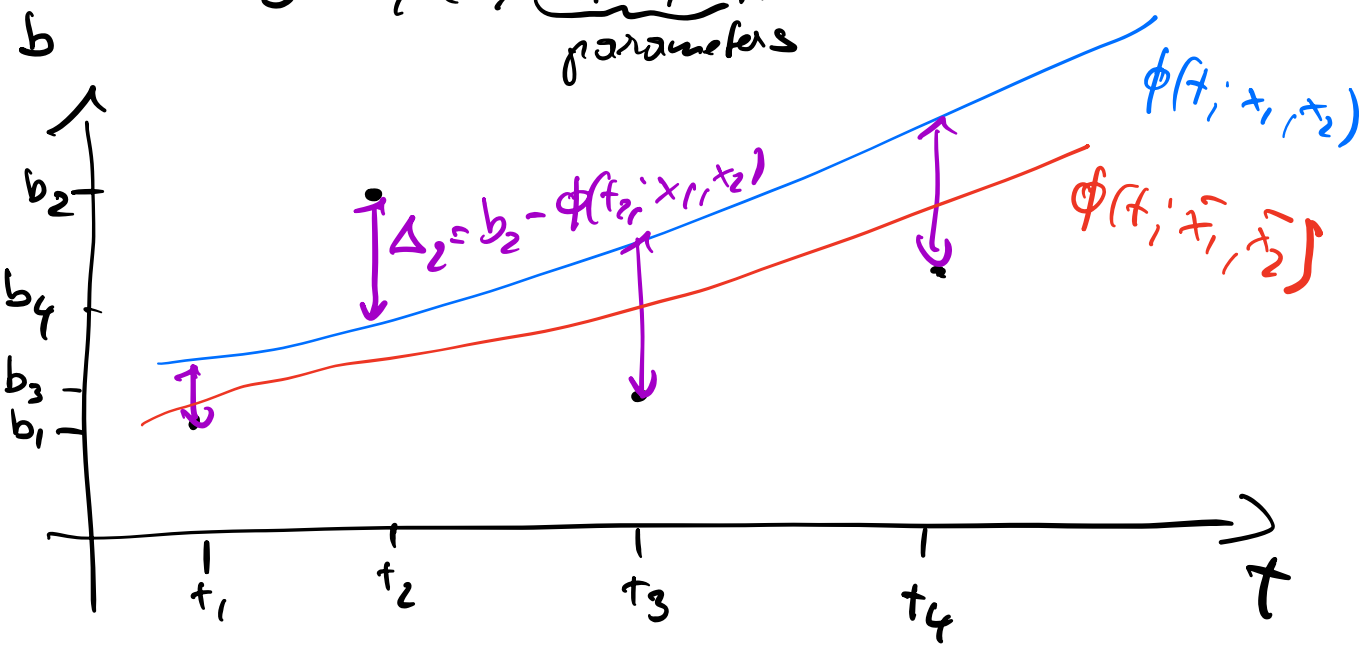
$\rightsquigarrow$ visualization on board

Least squares problem

data $\{(t_i, b_i)\}_{i=1}^{m}$ error $\Delta_i = b_i - \phi(t_i, x_1, \dots, x_n)$

parametrization

$$b = \phi(t; \underbrace{x_1, \dots, x_n}_{\text{parameters}})$$



Example:

1) $\phi(t; x_1, x_2) = x_1 t^2 + x_2 \exp(t)$

2) $\tilde{\phi}(t; x_1, x_2) = x_1^2 t^2 + \log(x_2)\, t$

Least-squares problems

Least squares: Find $x_1, \ldots, x_n$ such that

$$\frac{1}{2}\sum_{i=1}^{m} \Delta_i^2 \to \min$$

Weighted least squares: Find $x_1, \ldots, x_n$ such that

$$\frac{1}{2}\sum_{i=1}^{m} \left(\frac{\Delta_i}{\delta b_i}\right)^2 \to \min,$$

where $\delta b_i > 0$ contain information about how much we trust the ith data point.

Least-squares problems (cont'd)

Alternatives to using squares:

L^1 error: Find $x_1, \ldots, x_n$ such that

$$\sum_{i=1}^{m} |\Delta_i| \to \min$$

Result can be very different, other statistical interpretation, more stable with respect to outliers.

L^∞ error: Find $x_1, \ldots, x_n$ such that

$$\max_{1 \le i \le m} |\Delta_i| \to \min$$

Keeps the worst-case error small (risk averse)

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Cost analysis of LU decomposition
- Solving linear systems with sparse matrices
- Least-squares problems

Today

- Least-squares problems

Announcements

- Homework 2 has been posted; is due next week Mon, Oct 7 *before class*

Recap: Least-squares problems

Given data points/measurements

$$(t_i, b_i), \quad i = 1, \ldots, m$$

and a model function ϕ that relates t and b:

$$b = \phi(t; x_1, \ldots, x_n),$$

where $x_1, \ldots, x_n$ are model function parameters. If the model is supposed to describe the data, the deviations/errors

$$\Delta_i = b_i - \phi(t_i, x_1, \ldots, x_n)$$

should be small. Thus, to fit the model to the measurements, one must choose $x_1, \ldots, x_n$ appropriately.

Recap: Least-squares problems

Least squares: Find $x_1, \ldots, x_n$ such that

$$\frac{1}{2}\sum_{i=1}^{m} \Delta_i^2 \to \min$$

Weighted least squares: Find $x_1, \ldots, x_n$ such that

$$\frac{1}{2}\sum_{i=1}^{m} \left(\frac{\Delta_i}{\delta b_i}\right)^2 \to \min,$$

where $\delta b_i > 0$ contain information about how much we trust the ith data point.

Linear least-squares

We assume (for now) that the model depends linearly on $x_1, \ldots, x_n$, e.g.:

$$\phi(t; x_1, \ldots x_n) = a_1(t)x_1 + \ldots + a_n(t)x_n$$

$\rightsquigarrow$ board

$\phi(t; x_1, x_2) = x_1 t^2 + x_2 \exp(t)$; data points (t_i, b_i), $i = 1 \ldots 3$

$\Delta_1 = b_1 - (x_1 t_1^2 + x_2 \exp(t_1))$

$\Delta_2 = b_2 - (x_1 t_2^2 + x_2 \exp(t_2))$

$\Delta_3 = b_3 - (x_1 t_3^2 + x_2 \exp(t_3))$

$$\min_{x_1, x_2} \sum_{i=1}^{3} \Delta_i^2$$

2 unknows

3 equations

Linear least-squares

Choosing the least square error, this results in

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2,$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^T$, $\boldsymbol{b} = (b_1, \ldots, b_m)^T$, and $a_{ij} = a_j(t_i)$.

In the following, we study the overdetermined case, i.e., $m \geq n \rightsquigarrow$ board

$$A = \begin{bmatrix} t_1^2 & \exp(t_1) \\ t_2^2 & \exp(t_2) \\ t_3^2 & \exp(t_3) \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\in \mathbb{R}^{3\times 2} \qquad \in \mathbb{R}^2 \qquad \in \mathbb{R}^3$$

$$m = 3, \quad n = 2$$

Linear least-squares

Different perspective:

Consider non-square matrices $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and rank(A) $= n$. Then the system

$$A\boldsymbol{x} = \boldsymbol{b}$$

does not necessarily have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}} \Phi(\boldsymbol{x})$$

How can we solve this optimization problem?

Because we consider the Euclidean norm $\|.\|_2$, we obtain

$$\Phi(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \ldots$$

which is quadratic in $\boldsymbol{x}$ if $\boldsymbol{A}$ has full rank $\rightsquigarrow$ convex in $\boldsymbol{x}$

Therefore, the critical point is the global optimum

$$\nabla\Phi(\boldsymbol{x}) = \boldsymbol{A}^T(2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})) = \boldsymbol{0}$$

which satisfies the *normal equations*

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}.$$

If $\boldsymbol{A}$ is full rank, $\text{rank}(\boldsymbol{A}) = n$, then $\boldsymbol{A}^T\boldsymbol{A}$ is positive definite and the normal equations can be solved with the Cholesky factorization

Because we consider the Euclidean norm $\|.\|_2$, we obtain

$$\Phi(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \ldots$$

which is quadratic in $\boldsymbol{x}$ if $\boldsymbol{A}$ has full rank $\rightsquigarrow$ convex in $\boldsymbol{x}$

Therefore, the critical point is the global optimum

$$\nabla\Phi(\boldsymbol{x}) = \boldsymbol{A}^T(2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})) = \boldsymbol{0}$$

which satisfies the *normal equations*

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}.$$

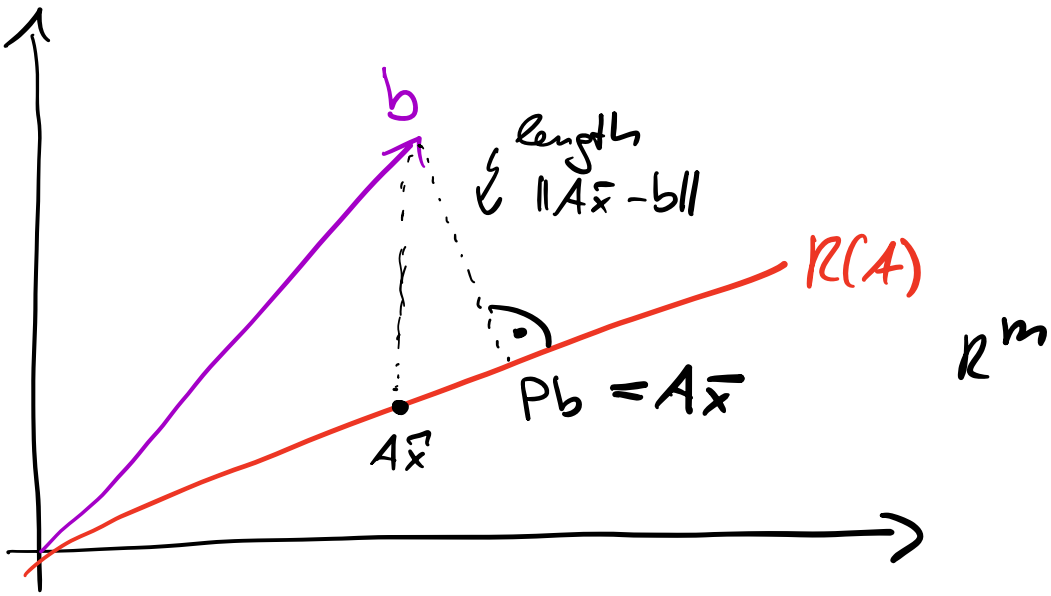
If $\boldsymbol{A}$ is full rank, $\text{rank}(\boldsymbol{A}) = n$, then $\boldsymbol{A}^T\boldsymbol{A}$ is positive definite and the normal equations can be solved with the Cholesky factorization (warning: we shouldn't do this, can you already see why?)

A geometry perspective on the normal equations

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2, \quad A \in \mathbb{R}^{m \times n}, \quad m > n, \quad \operatorname{rank}(A) = n$$
$$b \in \mathbb{R}^m$$

Geometric sketch

$$\bar{x} = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$
$$\Updownarrow$$
$$A\bar{x} - b \perp R(A)$$
$$\| \{Ax \mid x \in \mathbb{R}^n\}$$



Orthogonal projection

Consider finite-dim space V with $\langle \cdot, \cdot \rangle$ and subspace $U \subset V$. Let $U^\perp$ be the orthogonal complement w.r.t. V:

$$U^\perp = \{ v \in V \mid \langle v, u \rangle = 0 \quad \forall u \in U \}$$

Then

$$\underset{u' \in U}{\arg\min} \|u' - v\| = u \iff v - u \in U^\perp$$

Proof:

„$\Leftarrow$": let $u \in U$ such that $v - u \in U^\perp$, then for all $u' \in U$

$$\|v - u'\|^2 = \|v - u + u - u'\|^2 =$$

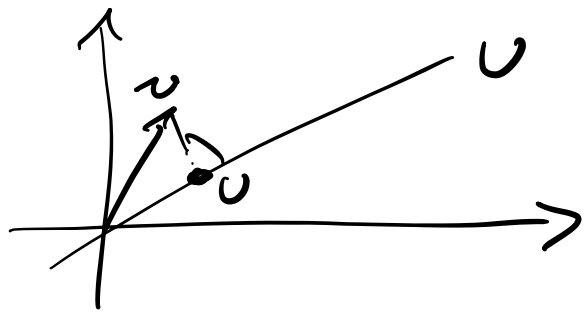
$$= \|v - u\|^2 + \underbrace{2\langle \quad \rangle}_{=0} + \|u - u'\|^2$$

$$= \|v - u\|^2 + \|u - u'\|^2 \geq \|v - u\|^2$$

„$\Rightarrow$"

This unique $u \in U$ is called the orthogonal projection of v onto U

$$u = P(v)$$



P is linear. Thus (finite-dim), there exists matrix P such that

$$u = \underline{P} v$$

Now apply geometric interpretation to find normal equations

$$\bar{x} = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$

$V = \mathbb{R}^m$, $U = R(A) \subset V$

$$\|b - Ax\|^2 = \min \iff \langle b - Ax, Ax' \rangle = 0 \quad \forall x' \in \mathbb{R}^n$$

$$\iff \langle A^T(b - Ax), x' \rangle = 0 \quad \forall x' \in \mathbb{R}^n$$

$$\iff A^T A x = A^T b$$

$$\iff x = (A^T A)^{-1} A^T b$$

$A(A^T A)^{-1} A^T$ ... projector onto column space of A

solve: $Ax = Pb = A(A^T A)^{-1} A^T b \iff x = (A^T A)^{-1} A^T b$

Linear least-squares problems

Solving the normal equations

$$A^T A\bar{\boldsymbol{x}} = A^T \boldsymbol{b}$$

requires:

- computing $A^T A$ (which is $O(mn^2)$)
- condition number of $A^T A$? $\rightsquigarrow$ board

Condition number of A^TA in $\|\cdot\|_2$ norm is

$$k_2(A^TA) = k_2(A)^2$$

Proof:

$$k_2(A)^2 = \|A^{-1}\|_2^2 \|A\|_2^2 = \frac{\max_{\|x\|_2=1} \|Ax\|_2^2}{\min_{\|x\|_2=1} \|Ax\|_2^2}$$

$$= \frac{\max_{\|x\|_2=1} \langle Ax, Ax \rangle_2}{\min_{\|x\|_2=1} \langle Ax, Ax \rangle_2}$$

$$= \frac{\max_{\|x\|_2=1} \langle A^TAx, x \rangle_2}{\min_{\|x\|_2=1} \langle A^TAx, x \rangle_2}$$

$$= \frac{\lambda_{max}(A^TA)}{\lambda_{min}(A^TA)}$$

Because A^TA is spd, have

$$\lambda_{max}(A^TA) = \sqrt{\lambda_{max}(A^TA)^2} = \sqrt{\lambda_{max}((A^TA)^2)}$$

$$= \sqrt{\lambda_{max}((A^TA)^T(A^TA))}$$

$$= \|A^TA\|_2$$

analogous: $\frac{1}{\lambda_{min}(A^TA)} = \|(A^TA)^{-1}\|_2$

$$k_2(A)^2 = \|A^TA\|_2 \|(A^TA)^{-1}\| = k_2(A^TA)$$

Because $\kappa_2(A^TA) = \kappa_2(A)^2$ we should <u>not</u> compute with A^TA even

---

Condition of orthogonal projection

Lemma:

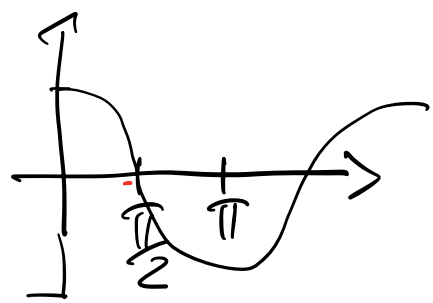
$P: \mathbb{R}^m \to V$ orthogonal projection onto $V \subseteq \mathbb{R}^m$

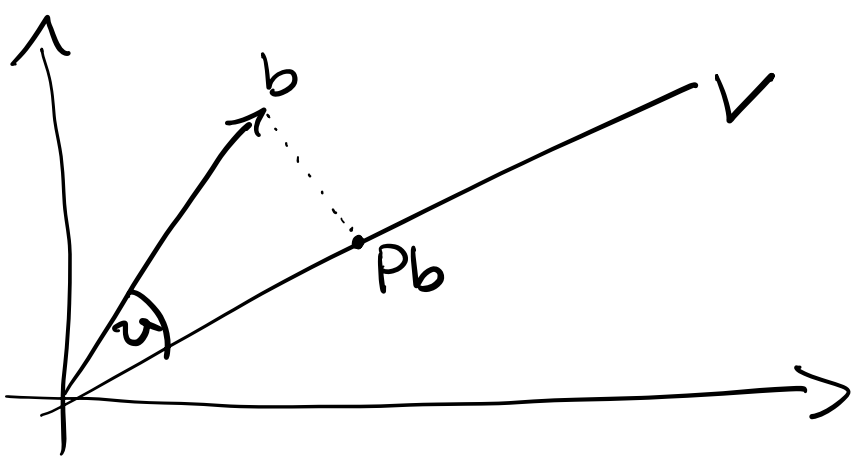
For $b \in \mathbb{R}^m$, denote by v angle between b and V defined by

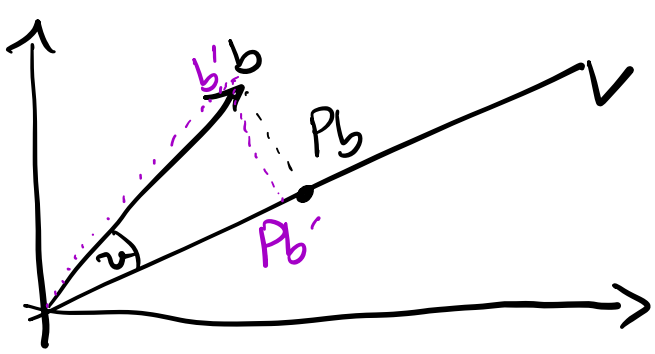
$$\sin(v) = \frac{\|b - Pb\|_2}{\|b\|_2}$$

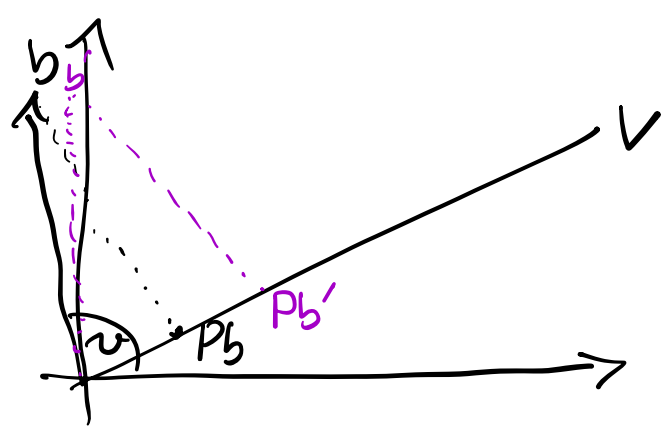
Then the <u>relative condition number of (P, b)</u> w.r.t. 2-norm is

$$\kappa_{rel} = \frac{1}{\cos(v)} \|P\|_2$$









Linear least-squares problems

Solving the normal equations

$$A^T A\bar{\boldsymbol{x}} = A^T \boldsymbol{b}$$

requires:

- computing $A^T A$ (which is $O(mn^2)$)
- condition number of $A^T A$? $\rightsquigarrow$ board is square of condition number of A; (problematic for the Choleski factorization)

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Least-squares problems

Today

- Least-squares problems

Announcements

- Homework 2 has been posted; is due next week Mon, Oct 7 *before class*

Recap: Linear least-squares

Consider non-square matrices $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and rank(A) $= n$. Then the system

$$A\boldsymbol{x} = \boldsymbol{b}$$

does not necessarily have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}} \Phi(\boldsymbol{x})$$

Recap: Linear least-squares

Consider non-square matrices $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and rank(A) $= n$. Then the system

$$A\boldsymbol{x} = \boldsymbol{b}$$

does not necessarily have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}} \Phi(\boldsymbol{x})$$

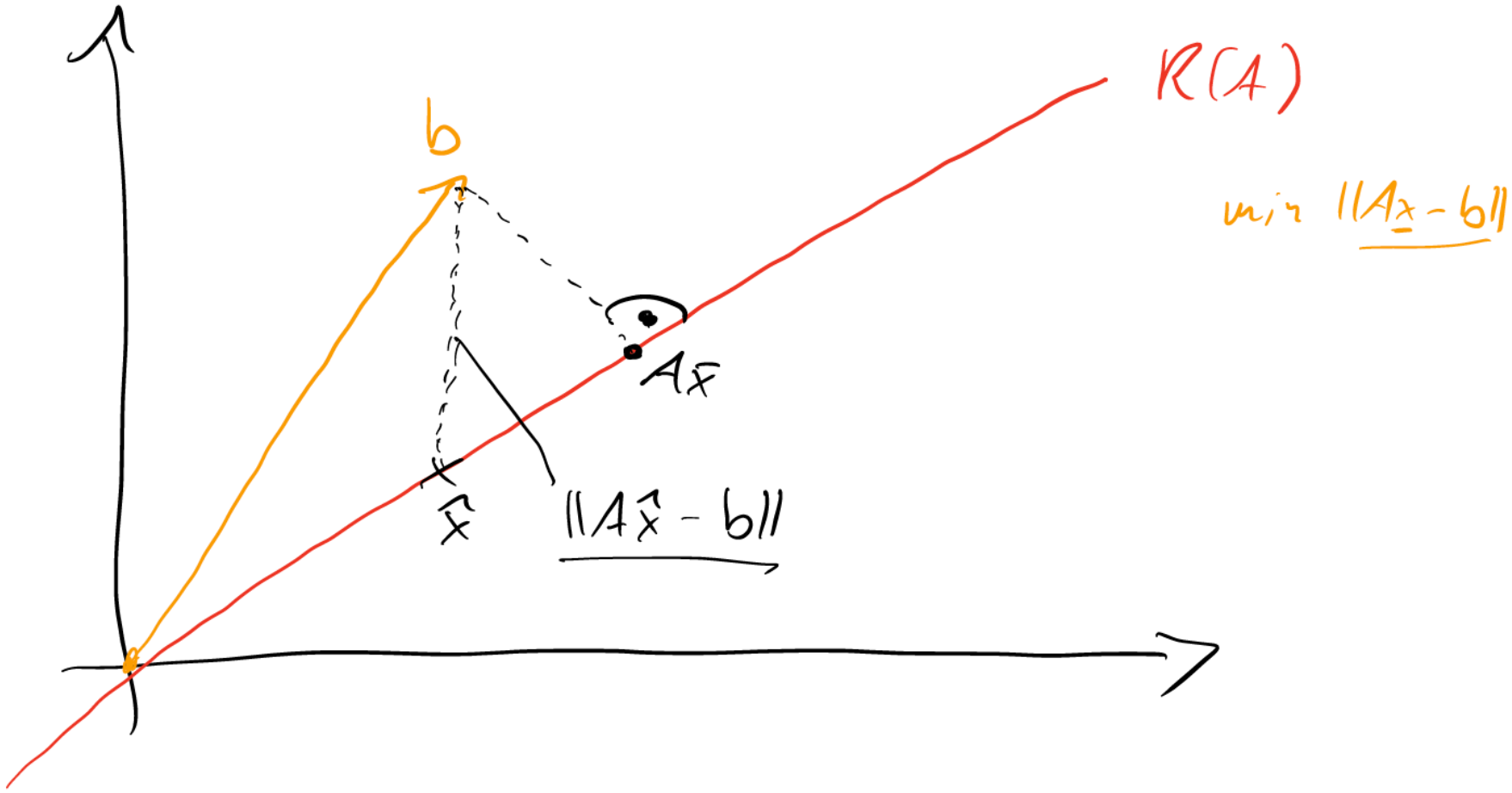
Solving the normal equations

$$A^T A \bar{\boldsymbol{x}} = A^T \boldsymbol{b}$$

requires:

- computing $A^T A$ (which is $O(mn^2)$)
- condition number of $A^T A$? $\rightsquigarrow$ is square of condition number of A; (problematic for the Choleski factorization)

Recap: Least-squares problems



Recap: Linear least-squares problems

Conditioning

Solving the normal equation is equivalent to computing $P\boldsymbol{b}$, the orthogonal projection of $\boldsymbol{b}$ onto the subspace V spanned by columns of A.

Let $P : \mathbb{R}^m \to V$ be an orthogonal projection onto $V \subseteq \mathbb{R}^n$. For $b \in \mathbb{R}^m$, denote by θ the angle between b and V defined by

$$\sin(\theta) = \frac{\|b - Pb\|_2}{\|b\|_2} .$$

The relative condition number of projecting b onto V with P with respect to the 2-norm (b is input) is

$$\kappa_{\text{rel}}(b) = \frac{1}{\cos(\theta(b))} \|P\|_2 .$$

$\rightsquigarrow$ board

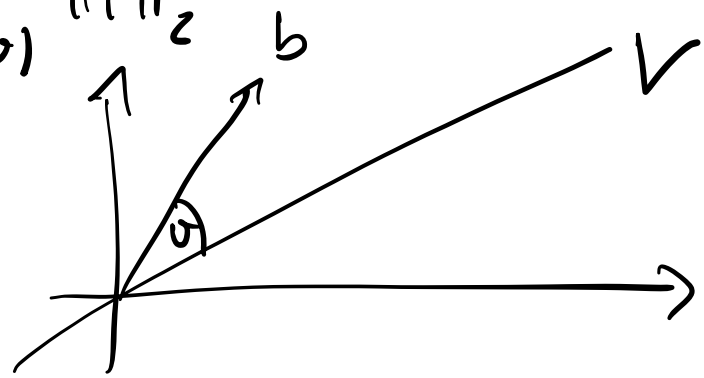
$P: \mathbb{R}^m \to V$ orthogonal proj onto $V \subseteq \mathbb{R}^m$

For $b \in \mathbb{R}^m$, denote angle ϑ between b and V that is defined as

$$\sin(\vartheta) = \frac{\|b - Pb\|_2}{\|b\|_2}$$

Then the relative condition number of (P, b) w.r.t. 2-norm is

$$k_{rel} = \frac{1}{\cos(\vartheta)} \|P\|_2$$



Def: $f: X \to Y$ differentiable

$$k_{rel} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\|$$

Projection: P is linear $\Rightarrow$ differentiable

$$k_{rel}(b) = \frac{\|b\|}{\|Pb\|} \|P'(b)\|$$

(1) $P'(b) = P$

(2) $\dfrac{\|Pb\|^2}{\|b\|^2}$

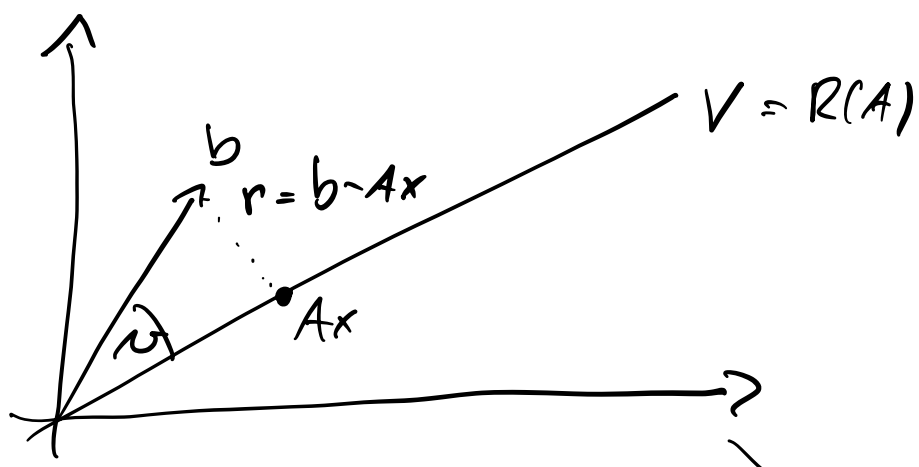
$$b - Pb \perp Pb$$

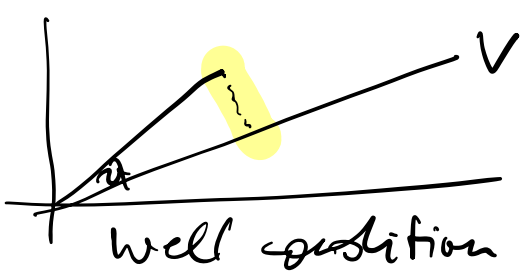
$$\|Pb\|^2 = \|b\|^2 - \|b - Pb\|^2$$

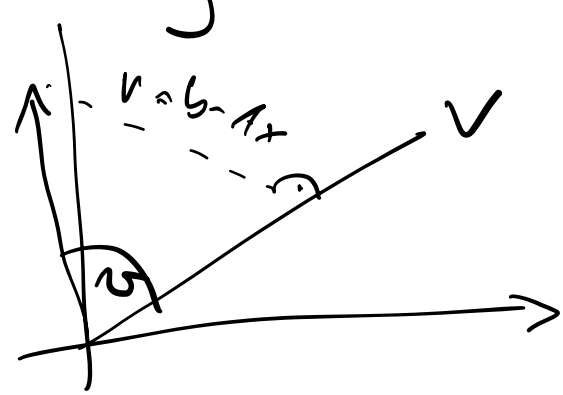
$$\frac{\|Pb\|^2}{\|b\|^2} = \frac{\|b\|^2 - \|b-Pb\|^2}{\|b\|^2}$$

$$= 1 - \sin^2(v) = \cos^2(v)$$

$$k_{rel}(b) = \frac{1}{\cos(v)} \|P\|_2$$







Linear least-squares problems

Now for the least-squares problem $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. The relative condition number κ in the Euclidean norm is bounded by

- With respect to perturbations in $\boldsymbol{b}$:

$$\kappa \leq \frac{\kappa_2(A)}{\cos(\theta)}$$

- With respect to perturbations in $\boldsymbol{A}$:

$$\kappa \leq \kappa_2(A) + \kappa_2(A)^2 \tan(\theta)$$

Proof $\rightsquigarrow$ next week

What are these bounds telling us?

Linear least-squares problems

Now for the least-squares problem $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. The relative condition number κ in the Euclidean norm is bounded by

- With respect to perturbations in $\boldsymbol{b}$:

$$\kappa \leq \frac{\kappa_2(A)}{\cos(\theta)}$$

- With respect to perturbations in $\boldsymbol{A}$:

$$\kappa \leq \kappa_2(A) + \kappa_2(A)^2 \tan(\theta)$$

Proof $\rightsquigarrow$ next week

What are these bounds telling us?

Small residual problems, small angle θ $\cos(\theta) \approx 1$, $\tan(\theta) \approx 0$: behavior similar to linear system.

Large residual problems, large angle θ $\cos(\theta) \ll 1$, $\tan(\theta) \approx 1$: behavior very different from linear system because $\kappa_2(A)^2$ shows up

How should we solve least-squares problems numerically?

We know from the previous slide that if the residual is large, then the condition κ is much larger than $\kappa_2(A)$ (closer to $\kappa_2(A)^2$)

- This is a poorly condition problem; however, do we care?

How should we solve least-squares problems numerically?

We know from the previous slide that if the residual is large, then the condition κ is much larger than $\kappa_2(A)$ (closer to $\kappa_2(A)^2$)

- This is a poorly condition problem; however, do we care?
- If the residual is large, then our $\boldsymbol{Ax}$ won't explain well the right-hand side $\boldsymbol{b}$
- This means that "our curve doesn't fit well the data" and we probably should try to find another space in which to search for a solution (another $\boldsymbol{A}$ with a range that better approximates the projected right-hand side $\boldsymbol{b}$)

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?
- If we choose a numerical method that solves the normal equations

$$\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}$$

then our problem becomes the problem of solving the linear system with matrix $\boldsymbol{A}^T \boldsymbol{A}$, which has condition number

$$\kappa_2(\boldsymbol{A}^T \boldsymbol{A}) = \kappa_2(\boldsymbol{A})^2$$

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?
- If we choose a numerical method that solves the normal equations

$$\boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{b}$$

  then our problem becomes the problem of solving the linear system with matrix $\boldsymbol{A}^T \boldsymbol{A}$, which has condition number

$$\kappa_2(\boldsymbol{A}^T \boldsymbol{A}) = \kappa_2(\boldsymbol{A})^2$$

  $\rightsquigarrow$ we are back in the situation of a poorly condition problem ("solving a linear system with $\boldsymbol{A}^T \boldsymbol{A}$") even though our original problem (least-squares problem) is well condition
- Can we do better and solve the least-squares problem (the problem we are actually interested in) without having to solve a problem with condition that grows with $\kappa_2(\boldsymbol{A})^2$ on the way?

The QR decomposition

Recall that projecting $\boldsymbol{b}$ onto the column span (range) of $\boldsymbol{A}$ was the key step $\rightsquigarrow$ let's try to find a numerical method that computes an orthonormal basis $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$ of the rank-n column span of $\boldsymbol{A}$

$$\boldsymbol{A} = \begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \ldots & \boldsymbol{a}_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad m \geq n$$

$$\Downarrow$$

$$\underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \ldots & \boldsymbol{a}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \ldots & \boldsymbol{q}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \\ & & & r_{nn} \end{bmatrix}}_{\boldsymbol{R}}$$

with an invertible matrix $\boldsymbol{R}$ so that

$$\operatorname{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k) = \operatorname{span}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k), \qquad k = 1, \ldots, n$$

$$\underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \dots & \boldsymbol{a}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & & \dots \\ & & \ddots & \\ & & & r_{nn} \end{bmatrix}}_{\boldsymbol{R}}$$

$\Downarrow$ leads to system of equations $\Downarrow$

$$\begin{aligned}
\boldsymbol{a}_1 &= r_{11}\boldsymbol{q}_1 \\
\boldsymbol{a}_2 &= r_{12}\boldsymbol{q}_1 + r_{22}\boldsymbol{q}_2 \\
\boldsymbol{a}_3 &= r_{13}\boldsymbol{q}_1 + r_{23}\boldsymbol{q}_2 + r_{33}\boldsymbol{q}_3 \\
&\vdots \\
\boldsymbol{a}_n &= r_{1n}\boldsymbol{q}_1 + r_{2n}\boldsymbol{q}_2 + \cdots + r_{nn}\boldsymbol{q}_n
\end{aligned}$$

What process does this motivate?

This motivates a process for computing the basis $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$

- At step j, we have $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$ that span $\text{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1})$
- We want to find $\boldsymbol{q}_j$ orthonormal to $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$ so that $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j$ spans $\text{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j)$
- Thus, set

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

  and normalize

$$\boldsymbol{q}_j = \frac{\boldsymbol{v}_j}{\|\boldsymbol{v}_j\|_2}$$

Notice that at step j, the quantities $\boldsymbol{q}_1^T \boldsymbol{a}_j, \boldsymbol{q}_2^T \boldsymbol{a}_j, \ldots, \boldsymbol{q}_{j-1}^T \boldsymbol{a}_j$ are the values $r_{j,1}, \ldots, r_{j,j-1}$ and r_{jj} is responsible for the normalization and set to

$$r_{jj} = \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|_2$$

This process is the *classical Gram-Schmidt* procedure to compute the QR factorization

This motivates a process for computing the basis $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$

- At step j, we have $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$ that span $\text{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{j-1})$
- We want to find $\boldsymbol{q}_j$ orthonormal to $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{j-1}$ so that $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j$ spans $\text{span}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j)$
- Thus, set

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

  and normalize

$$\boldsymbol{q}_j = \frac{\boldsymbol{v}_j}{\|\boldsymbol{v}_j\|_2}$$

Notice that at step j, the quantities $\boldsymbol{q}_1^T \boldsymbol{a}_j, \boldsymbol{q}_2^T \boldsymbol{a}_j, \ldots, \boldsymbol{q}_{j-1}^T \boldsymbol{a}_j$ are the values $r_{j,1}, \ldots, r_{j,j-1}$ and r_{jj} is responsible for the normalization and set to

$$r_{jj} = \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|_2$$

This process is the *classical Gram-Schmidt* procedure to compute the QR factorization

However, this process is numerically unstable!

Instead of directly computing

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

Instead of directly computing

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

based on $\boldsymbol{a}_j$, the *modified* Gram-Schmidt procedure computes $\boldsymbol{v}_j$ iteratively

$$\begin{aligned}
\boldsymbol{v}_j^{(1)} &= \boldsymbol{a}_j\,, \\
\boldsymbol{v}_j^{(2)} &= \boldsymbol{v}_j^{(1)} - \boldsymbol{q}_1\boldsymbol{q}_1^T \boldsymbol{v}_j^{(1)}\,, && \text{"subtract from } \boldsymbol{v}_j^{(1)} \text{ what is already in } \boldsymbol{q}_1\text{"} \\
\boldsymbol{v}_j^{(3)} &= \boldsymbol{v}_j^{(2)} - \boldsymbol{q}_2\boldsymbol{q}_2^T \boldsymbol{v}_j^{(2)}\,, && \text{"subtract from } \boldsymbol{v}_j^{(2)} \text{ what is already in } \boldsymbol{q}_2\text{"} \\
&\vdots \\
\boldsymbol{v}_j &= \boldsymbol{v}_j^{(j)} = \boldsymbol{v}_j^{(j-1)} - \boldsymbol{q}_{j-1}\boldsymbol{q}_{j-1}^T \boldsymbol{v}_j^{(j-1)}
\end{aligned}$$

Computing a QR factorization with the modified Gram-Schmidt procedure is stabler than with the classical Gram-Schmidt procedure. However, even the modified Gram-Schmidt procedure can lead to vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$ that are far from orthogonal if the condition number of $\boldsymbol{A}$ is large (see, Golub et al., Matrix Computations, Section 5.2.9)

Let's recall what the Gram-Schmidt procedure is doing: It is applying a succession of triangular matrices R_k on the right of A so that the resulting matrix

$$\boldsymbol{A}\underbrace{\boldsymbol{R}_1\boldsymbol{R}_2\ldots\boldsymbol{R}_n}_{\boldsymbol{R}^{-1}} = \boldsymbol{Q}$$

has orthonormal columns and $\boldsymbol{R}$ is upper-triangular.

Let's recall what the Gram-Schmidt procedure is doing: It is applying a succession of triangular matrices R_k on the right of A so that the resulting matrix

$$\boldsymbol{A}\underbrace{\boldsymbol{R}_1\boldsymbol{R}_2\ldots\boldsymbol{R}_n}_{\boldsymbol{R}^{-1}} = \boldsymbol{Q}$$

has orthonormal columns and $\boldsymbol{R}$ is upper-triangular.

Instead, we could try to find orthonormal matrices ($\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{X}\boldsymbol{X}^T = \boldsymbol{I}$) so that

$$\underbrace{\boldsymbol{Q}_n\ldots\boldsymbol{Q}_2\boldsymbol{Q}_1}_{\boldsymbol{Q}}\boldsymbol{A} = \boldsymbol{R}$$

is upper-triangular. The product $\boldsymbol{Q}_n\ldots\boldsymbol{Q}_2\boldsymbol{Q}_1 = \boldsymbol{Q}^T$ is orthonormal too and thus $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ a QR factorization of $\boldsymbol{A}$.

The Householder method judiciously finds the matrices $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \ldots, \boldsymbol{Q}_n$ via so-called Householder reflectors $\rightsquigarrow$ board. The Householder method is backward stable.

The QR factorization

All these three algorithms (classical Gram-Schmidt, modified Gram-Schmidt, Householder triangularization) have roughly the FLOPs of $2mn^2$ for an $m \times n$ matrix

Why would we ever want to use (modified) Gram-Schmidt instead of Householder triangularization?

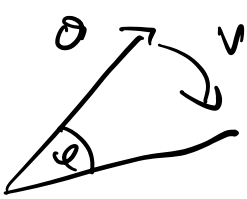
Transform A by multiplications with orthogonal matrices

$$A \rightsquigarrow Q_1 A \rightsquigarrow Q_2 Q A \rightsquigarrow \dots$$

$$k_2(Q) = \|Q\| \, \|Q^{-1}\| = 1 \qquad \Big| \qquad \det(Q)^2 = 1$$

What are basic orthogonal transformations in $\mathbb{R}^{2,2}$?

<u>rotation</u> $(\det = 1)$



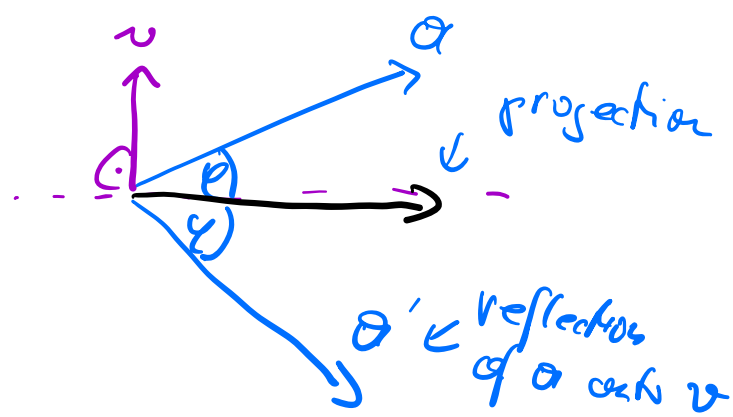
$$\begin{bmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{bmatrix}$$

$\Rightarrow$ Givens rotations to construct Q with $A = QR$

$\Rightarrow$ text books

reflections $(\det = -1)$

reflect at a hyperplane normal to v



projection

$$a \mapsto \left(I - \frac{v v^T}{v^T v}\right) a$$

reflection

$$a \mapsto \left(I - 2 \frac{v v^T}{v^T v}\right) a$$

$\Rightarrow$ Householder reflections

Householder transformation

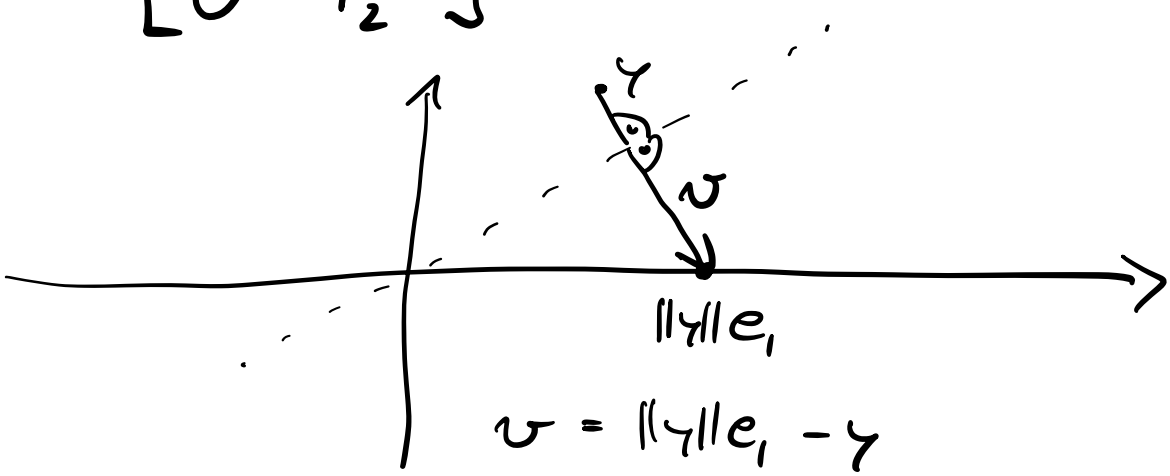
$$\underset{A}{\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix}} \xrightarrow[Q_1]{k=1} \underset{Q_1 A}{\begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix}} \xrightarrow[Q_2]{k=2} \underset{Q_2 Q_1 A}{\begin{bmatrix} x & x & x \\ & x & x \\ & 0 & x \\ & 0 & x \\ & 0 & x \end{bmatrix}}$$

of step $k=2$

$$y = \begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix} \xrightarrow[F]{} \begin{bmatrix} * \\ 0 \\ 0 \\ 0 \end{bmatrix} = * \, \underline{e}_1$$

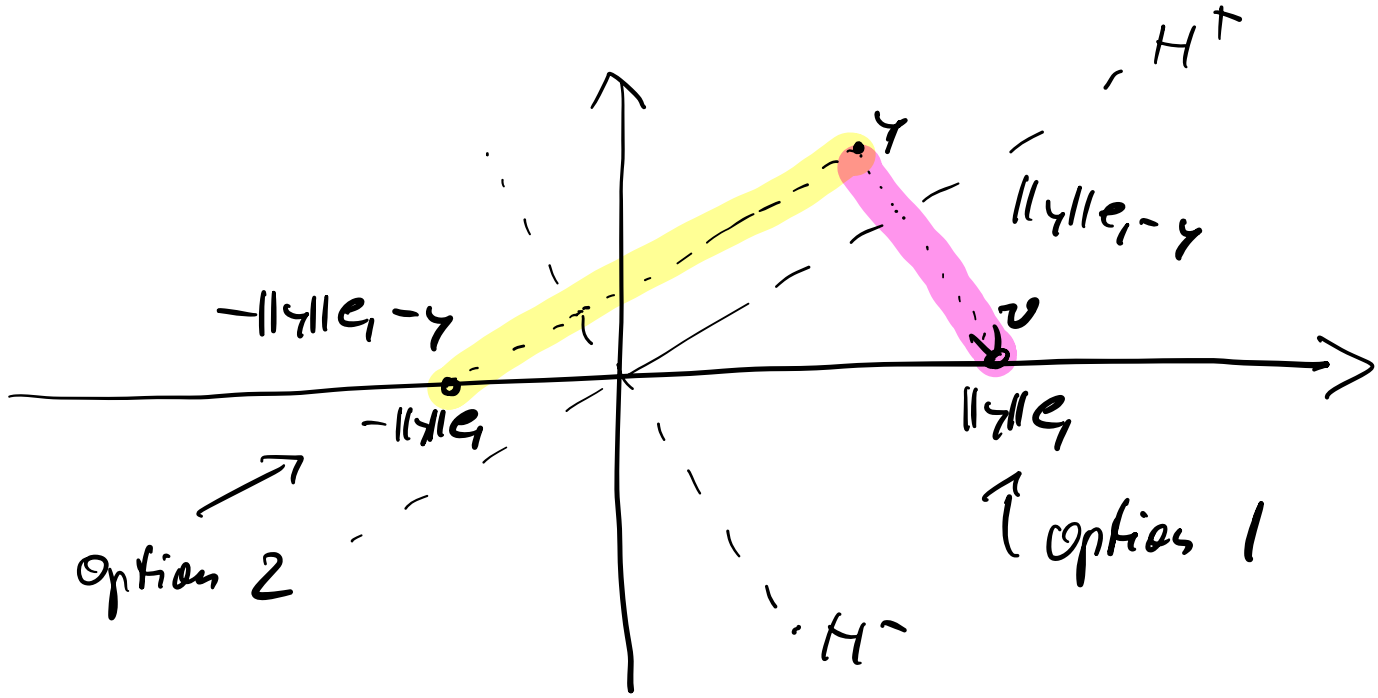
find

$$Q_2 = \begin{bmatrix} I_{k-1} & 0 \\ 0 & F_2 \end{bmatrix}$$



$$v = \|y\| e_1 - y$$

$$F_2 = \left(I - 2 \frac{v v^T}{v^T v} \right)$$



distance of reflection

→ taking the reflection corresponding to the longer distance is numerically more stable.

The QR factorization

All these three algorithms (classical Gram-Schmidt, modified Gram-Schmidt, Householder triangularization) have roughly the FLOPs of $2mn^2$ for an $m \times n$ matrix

Why would we ever want to use (modified) Gram-Schmidt instead of Householder triangularization? Gram-Schmidt can be easier to parallelize, for example (Recall that best algorithm depends also on what hardware we want to implement it on.)

Every matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ has a QR factorization. It is unique if we require the diagonal elements of R to be positive.

If $m > n$ and $\boldsymbol{Q} \in \mathbb{R}^{m \times n}$, then we speak of a reduced QR factorization. Otherwise, we have $\boldsymbol{Q} \in \mathbb{R}^{m \times m}$ and we speak of a full QR factorization.

```
>> A = randn(10, 10); [Q, R] = qr(A);
>> size(Q)
ans =
    10    10
>> size(R)
ans =
    10    10
```

```
>> A = randn(10, 4); [Q, R] = qr(A)
>> size(Q)
ans =
    10    10
>> size(R)
ans =
    10     4
>>
>> [Q, R] = qr(A, 0); % reduced QR
>> size(Q)
ans =
    10     4
>> size(R)
ans =
     4     4
```

Back to our least-squares problem

One would like to avoid the multiplication $A^T A$ and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \left[Q_1, Q_2\right] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

Back to our least-squares problem

One would like to avoid the multiplication $A^T A$ and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \begin{bmatrix} Q_1, Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}} \|Q^T(A\boldsymbol{x} - \boldsymbol{b})\|^2 = \min_{\boldsymbol{x}} \left\| \begin{bmatrix} \boldsymbol{b}_1 - R_1\boldsymbol{x} \\ \boldsymbol{b}_2 \end{bmatrix} \right\|^2,$$

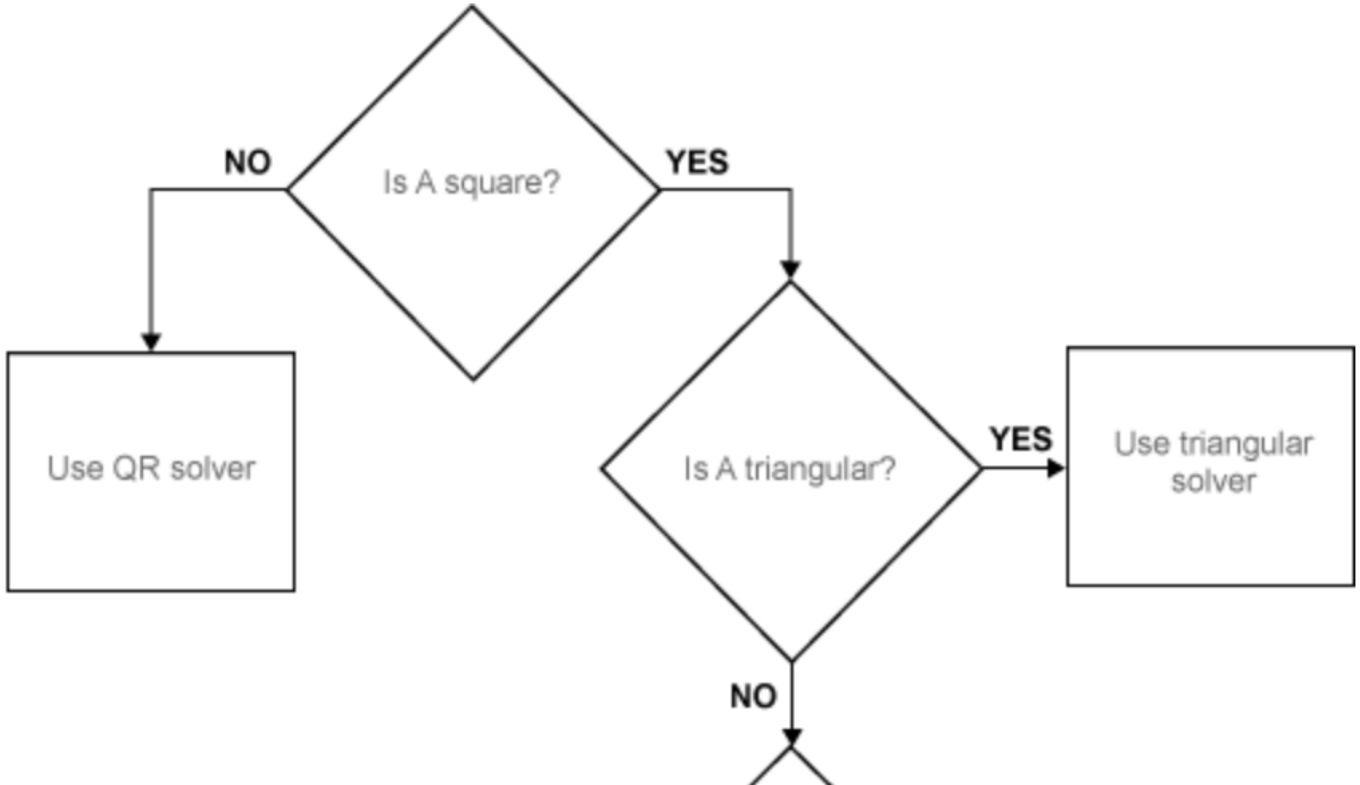
$$= \min_{\boldsymbol{x}} \|\boldsymbol{b}_1 - R_1\boldsymbol{x}\|^2 + \|\boldsymbol{b}_2\|^2$$

where $Q^T\boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix}$.

Thus, the least squares solution is $\boldsymbol{x} = R^{-1}\boldsymbol{b}_1$ and the residual is $\|\boldsymbol{b}_2\|$.

Stability of solving least-squares problem with Householder triangularization

Solving a least-squares problem with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $m \geq n$ and $\text{rank}(\boldsymbol{A}) = n$ via QR factorization computed with Householder triangularization is backward stable.



Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Eigen decomposition

Eigen decomposition

- For a square matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$, there exists at least one λ such that

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x} \implies (\boldsymbol{A} - \lambda \boldsymbol{I})\boldsymbol{x} = 0$$

- Putting the eigenvectors $\boldsymbol{x}_j$ as columns in a matrix $\boldsymbol{X}$, and the eigenvalues λ_j on the diagonal of a diagonal matrix $\boldsymbol{\Lambda}$, we get

$$\boldsymbol{A}\boldsymbol{X} = \boldsymbol{X}\boldsymbol{\Lambda}$$

- A matrix is non-defective or diagonalizable if there exist n linearly independent eigenvectors, which means that $\boldsymbol{X}$ is invertible

$$\begin{aligned} \boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X} &= \boldsymbol{\Lambda} \\ \boldsymbol{A} &= \boldsymbol{X}\boldsymbol{\Lambda}\boldsymbol{X}^{-1} \end{aligned}$$

- The transformation from $\boldsymbol{A}$ to $\boldsymbol{\Lambda} = \boldsymbol{X}^{-1}\boldsymbol{A}\boldsymbol{X}$ is called a similarity transformation and it preserves the eigenvalues.

- A matrix is unitarily diagonalizable if there exist n linearly independent orthogonal eigenvectors, i.e., if the matrix $\boldsymbol{X}$ can be chosen to be unitary (orthonormal), $\boldsymbol{X} = \boldsymbol{U}$, where $\boldsymbol{U}^{-1} = \boldsymbol{U}^H$

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^H$$

  Note that unitary matrices generalize orthogonal matrices to the complex domain, so we use adjoints (conjugate transpose) instead of transpose throughout

- Theorem: A matrix is unitarily diagonlizable iff it is normal, i.e., it commutes with its adjoint:

$$\boldsymbol{A}^H\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^H$$

- Theorem: Hermitian (symmetric) matrices, $\boldsymbol{A}^H = \boldsymbol{A}$, are unitarily diagonalizable and have *real* eigenvalues.

- The usual eigenvectors are more precisely called *right* eigenvectors. There are also *left* eigenvectors corresponding to a given eigenvalue λ

$$\boldsymbol{y}^H \boldsymbol{A} = \lambda \boldsymbol{y}^H \implies \boldsymbol{A}^H \boldsymbol{y} = \bar{\lambda} \boldsymbol{y}\,,$$

$$\boldsymbol{Y}^H \boldsymbol{A} = \boldsymbol{\Lambda} \boldsymbol{Y}^H$$

  with conjugate $\bar{\lambda}$ of λ

- For a matrix that is diagonalizable, observe that

$$\boldsymbol{Y}^H = \boldsymbol{X}^{-1}$$

  and so the left eigenvectors provide no new information

- For unitarily diagonalizable matrices, $\boldsymbol{Y} = (\boldsymbol{X}^{-1})^H = (\boldsymbol{X}^H)^H = \boldsymbol{X} = \boldsymbol{U}$, so that the left and right eigenvectors coincide.

Numerically finding eigenvalues

For a matrix $A \in \mathbb{C}^{n \times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Most relevant problems:

- A symmetric (and large)
- A spd (and large)
- A stochastic matrix, i.e., all entries $0 \leq a_{ij} \leq 1$ are probabilities, and thus $\sum_j a_{ij} = 1$.

How hard are they to find numerically?

How hard are they to find numerically?

- This is a nonlinear problem.

How hard are they to find numerically?

- This is a nonlinear problem.
- How difficult is this?

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

- Must use an iterative algorithm

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

- Must use an iterative algorithm $\rightsquigarrow$ this is fundamentally different from what we have seen previously when solving systems of *linear* equations! These algorithms (LU, QR) give the *exact* solution in *exact* arithmetic in finite number of steps. We *cannot* expect something similar for computing eigenvalues!

Condition of finding eigenvalues of a matrix

The absolute condition number of determining a simple eigenvalue λ_0 of a matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ with respect to the $\|\cdot\|_2$ is

$$\kappa_{\text{abs}} = \frac{1}{|\cos(\angle(\boldsymbol{x},\boldsymbol{y}))|}\,, \qquad \cos(\angle(\boldsymbol{x},\boldsymbol{y})) = \frac{|\langle \boldsymbol{x},\boldsymbol{y}\rangle|}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

and the relative condition number is

$$\kappa_{\text{rel}} = \frac{\|\boldsymbol{A}\|}{|\lambda_0 \cos(\angle(\boldsymbol{x},\boldsymbol{y}))|}\,,$$

where $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ for the eigenvalue λ_0 ($\boldsymbol{A}\boldsymbol{x} = \lambda_0\boldsymbol{x}$) and $\boldsymbol{y}$ an adjoint eigenvector ($\boldsymbol{A}^H\boldsymbol{y} = \bar{\lambda}_0\boldsymbol{y}$).

Sketch of proof $\rightsquigarrow$ board $\rightsquigarrow$ next time

(see also Deuflhard, Theorem 5.2)

Interpretation

Perturbations of order δ in entries of matrix $\boldsymbol{A}$ induce changes of the order $\delta\lambda = \delta / \cos(\angle(\boldsymbol{x}_0, \boldsymbol{y}_0))$

In particular, for normal matrices* ($\boldsymbol{A}\boldsymbol{A}^H = \boldsymbol{A}^H\boldsymbol{A}$), we have $\boldsymbol{x}_0 = \boldsymbol{y}_0$ and thus $\angle(\boldsymbol{x}_0, \boldsymbol{y}_0) = 0$ and thus $\cos(\angle(\boldsymbol{x}_0, \boldsymbol{y}_0)) = 1$, which means $\kappa_{\text{abs}} = 1$, which can be considered well conditioned

Finding non-simple eigenvalues can have very high absolute condition number (but can still be done numerically). For a detailed treatment have a look at textbook by Golub et al. on Matrix Computations.

*Equivalent: Have orthonormal eigenbasis of $\mathbb{C}$; diagonalizable by unitary matrix.

Bounding error in eigenvalue computation

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ be a Hermitian matrix and let $(\hat{\lambda}, \hat{\boldsymbol{x}})$ be a computed approximation of an eigenvalue/eigenvector pair $(\lambda, \boldsymbol{x})$ of $\boldsymbol{A}$. Defining the residual

$$\hat{\boldsymbol{r}} = \boldsymbol{A}\hat{\boldsymbol{x}} - \hat{\lambda}\hat{\boldsymbol{x}}\,, \qquad \hat{\boldsymbol{x}} \neq \boldsymbol{0}\,,$$

it then follows that

$$\min_{\lambda_i \in \sigma(\boldsymbol{A})} |\hat{\lambda} - \lambda_i| \leq \frac{\|\hat{\boldsymbol{r}}\|_2}{\|\hat{\boldsymbol{x}}\|_2}\,,$$

where $\sigma(\boldsymbol{A}) = \{\lambda | \lambda \text{ is an eigenvalue of } \boldsymbol{A}\}$ is the spectrum of $\boldsymbol{A}$.

Proof $\rightsquigarrow$ board

What is special about this bound?

Let $A \in \mathbb{C}^{n\times n}$ Hermite, let $(\hat{\lambda}, \hat{x})$ approximation of eigenpair (λ, x). Define residual

$$\hat{r} = A\hat{x} - \hat{\lambda}\hat{x}, \qquad \hat{x} \neq 0$$

it follows

$$\min_{\lambda_i \in \sigma(A)} |\hat{\lambda} - \lambda_i| \leq \frac{\|\hat{r}\|}{\|\hat{x}\|}$$

A is Hermite, exists an orthogonal eigenbasis of $\mathbb{C}^n$ $\{u_k\}_{k=1}^n$

$$\hat{x} = \sum_{i=1}^n \alpha_i u_i \qquad \text{with} \quad \alpha_i = u_i^H \hat{x}$$

$$\hat{r} = A\hat{x} - \hat{\lambda}\hat{x} = A\left(\sum_{i=1}^n \alpha_i u_i\right) - \hat{\lambda}\sum_{i=1}^n \alpha_i u_i$$

$$= \sum_{i=1}^n \alpha_i \lambda_i u_i - \hat{\lambda}\sum_{i=1}^n \alpha_i u_i$$

$$= \sum_{i=1}^n \alpha_i (\lambda_i - \hat{\lambda}) u_i$$

$$\frac{\|\hat{r}\|^2}{\|\hat{x}\|^2} = \frac{\sum_{i=1}^n |\alpha_i|^2 (\lambda_i - \hat{\lambda})^2 \|u_i\|^2}{\sum_{j=1}^n |\alpha_j|^2 \|u_j\|^2}$$

$$= \sum_{i=1}^n \left(\frac{|\alpha_i|^2}{\sum_{j=1}^n |\alpha_j|^2}\right)(\lambda_i - \hat{\lambda})^2 = \sum_{i=1}^n \beta_i (\lambda_i - \hat{\lambda})^2$$

Now note that $\beta_i \geq 0$

$$\sum_{i=1}^{n} \beta_i = \sum_{i=1}^{n} \frac{|\alpha_i|^2}{\sum_{j=1}^{n} |\alpha_j|^2} = 1$$

Thus

$$\frac{\|\hat{r}\|^2}{\|\hat{x}\|^2} = \sum_{i=1}^{n} \beta_i (\lambda_i - \hat{\lambda})^2 \geq \sum_{i=1}^{n} \beta_i \min_j (\lambda_j - \hat{\lambda})^2$$

$$= \min_j (\lambda_j - \hat{\lambda})^2 \underbrace{\sum_{i=1}^{n} \beta_i}_{=1} = \min_j (\lambda_j - \hat{\lambda})^2$$

$$\min_j |\lambda_j - \hat{\lambda}| \leq \frac{\|r\|}{\|\hat{x}\|}$$

Bounding error in eigenvalue computation

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ be a Hermitian matrix and let $(\hat{\lambda}, \hat{\boldsymbol{x}})$ be a computed approximation of an eigenvalue/eigenvector pair $(\lambda, \boldsymbol{x})$ of $\boldsymbol{A}$. Defining the residual

$$\hat{\boldsymbol{r}} = \boldsymbol{A}\hat{\boldsymbol{x}} - \hat{\lambda}\hat{\boldsymbol{x}}\,, \qquad \hat{\boldsymbol{x}} \neq \boldsymbol{0}\,,$$

it then follows that

$$\min_{\lambda_i \in \sigma(\boldsymbol{A})} |\hat{\lambda} - \lambda_i| \leq \frac{\|\hat{\boldsymbol{r}}\|_2}{\|\hat{\boldsymbol{x}}\|_2}\,,$$

where $\sigma(\boldsymbol{A}) = \{\lambda | \lambda \text{ is an eigenvalue of } \boldsymbol{A}\}$ is the spectrum of $\boldsymbol{A}$.

Proof $\rightsquigarrow$ board

What is special about this bound?

- This is an *a posteriori* bound that bounds the error *after* we have computed the result
- We will see many more residual-based *a posteriori* bounds (broadly speaking: the residual is something we can compute, and if the problem is "well-behaved" then the norm of the residual is a reasonable bound of the norm of the error.)

Condition of computing eigenvectors

- The condition of computing eigenvector $\boldsymbol{x}_i$ for an eigenvalue λ_i depends on the separation between the eigenvalues

$$\kappa = \frac{1}{\min_{i \neq j} |\lambda_i - \lambda_j|}$$

(Quarteroni et al., Section 5)

- Computing $\boldsymbol{x}_i$ can be ill-conditioned if some eigenvalue λ_j is "very close" to the eigenvalue λ_i

- This indicates that multiple eigenvalues require care. Even for Hermitian matrices eigen*vectors* can be hard to compute

The Power Method

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ be diagonalizable matrix and λ_1 be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Let $\boldsymbol{x}_0$ be an initial guess that is not orthogonal to the eigenspace of λ_1, then $\boldsymbol{x}_k$ obtained via the iterations

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \tag{1}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{z}_{k+1} / \|\boldsymbol{z}_{k+1}\|_2 \tag{2}$$

will converge to the normalized eigenvector of $\boldsymbol{A}$ corresponding to λ_1 for $k \to \infty$.

This process is called the power method.

Proof $\rightsquigarrow$ board

Power method

↳ first $$x_k = \frac{A^k x_0}{\|A^k x_0\|_2}$$

↳ Because A is diagonalizable, have vectors

$$\{v_i\}_{i=1}^{n}$$

with

$$x_0 = \sum_{i=1}^{n} \alpha_i v_i, \qquad \alpha_1 \neq 0$$

↳ Further

$$A^k x_0 = V \Lambda^k V^{-1} x_0 = V \Lambda^k \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}$$

$$= \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i$$

$$= \alpha_1 \lambda_1^k \Big(v_1 + \overbrace{\sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i}^{\gamma^{(k)}} \Big)$$

Because $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$ per definition of λ_1, the vector $A^k x_0$ assumes an increasingly significant component in direction v_1

$$x_k = \frac{\alpha_1 \lambda_1^k (v_1 + y^{(k)})}{\|\alpha_1 \lambda_1^k (v_1 + y^{(k)})\|_2} = \eta_k \frac{v_1 + y^{(k)}}{\|v_1 + y^{(k)}\|_2}$$

where η_k is the sign of $\alpha_1 \lambda_1^k$.

with

$$\|y^{(k)}\| \to 0 \quad \text{for } k \to \infty$$

$$x_k \to \frac{v_1}{\|v_1\|} \eta$$

Convergence speed

component in direction v_2 converges slowest with $\mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right)$

$\to$ the bigger the gap $\lambda_1 \leftrightarrow \lambda_2$, the faster

$\to$ again, well separated EV are „easiest" to find

Compute eigenvalue $\lambda_1^{(k)}$ via

$$\frac{y^T A y}{y^T y} = \frac{y^T \lambda y}{y^T y} = \lambda \quad \text{if } y \text{ is EV with EV } \lambda \text{ of } A$$

$$\lambda_1^{(k)} = \frac{x_k^T A x_k}{x_k^T x_k}$$

Power method (cont'd)

Start with initial guess $\boldsymbol{x}_0$ and then iterate

- Compute matrix-vector product and normalize it

$$\boldsymbol{x}_k = \frac{\boldsymbol{A}\boldsymbol{x}_{k-1}}{\|\boldsymbol{A}\boldsymbol{x}_{k-1}\|}$$

- Obtain eigenvalue estimate (note that $\|\boldsymbol{x}_k\| = 1$)

$$\lambda_1^{(k)} = \boldsymbol{x}_k^H \boldsymbol{A}\boldsymbol{x}_k$$

- Test for convergence? How?

Power method (cont'd)

Start with initial guess $\boldsymbol{x}_0$ and then iterate

- Compute matrix-vector product and normalize it

$$\boldsymbol{x}_k = \frac{\boldsymbol{A}\boldsymbol{x}_{k-1}}{\|\boldsymbol{A}\boldsymbol{x}_{k-1}\|}$$

- Obtain eigenvalue estimate (note that $\|\boldsymbol{x}_k\| = 1$)

$$\lambda_1^{(k)} = \boldsymbol{x}_k^H \boldsymbol{A}\boldsymbol{x}_k$$

- Test for convergence? How? Compute residual

$$\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{x}_k - \lambda_1^{(k)}\boldsymbol{x}_k$$

  and terminate if the error estimate is small enough (bound if $\boldsymbol{A}$ Hermitian, heuristic otherwise)

$$\min_i |\lambda_i - \lambda_1^{(k)}| \leq \frac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{x}_k\|} < \epsilon$$

Let us recall the condition number

$$\| f(\tilde{x}) - f(x) \| \overset{\cdot}{\leq} \kappa_{abs} \, \| \tilde{x} - x \|, \quad x \to \tilde{x}$$

We define „$\overset{\cdot}{\leq}$" to mean

$$g(x) \overset{\cdot}{\leq} h(x), \; x \to x_0 \quad \hat{=} \quad g(x) \leq h(x) + \underset{x \to x_0}{\underline{\underline{o}}}(\| h(x) \|)$$

Sometime you see

$$g(x) \leq h(x) + \mathcal{O}(\| h(x) \|^2)$$

Thus, the condition number tells us:

$$\| f(x + \delta x) - f(x) \| \leq \kappa_{abs} \, \| \delta x \| + o(\delta x), \quad \delta x \to 0$$

This is a perturbation bound.

With condition number, have a receipt how to derive perturbation bounds

① formulate problem via function f

② establish f is differentiable

③ take derivative of f and compute the norm

Let us start with „$Ax = b$"

Bound for perturbation in b

$$f : \mathbb{R}^n \to \mathbb{R}^n, \; b \mapsto f(b) = A^{-1} b \qquad ①$$

Function f is linear $\to$ differentiable ②

Derivative $f'(b) = A^{-1}$ with norm $\|A^{-1}\|$ ③

Obtain

$$\|x - \tilde{x}\| \leq \|A^{-1}\| \, \|\delta b\| + o(\|\delta b\|), \quad \|\delta b\| \to 0$$

Now bound for „$Ax = b$" with perturbations in A

$$f: GL(n) \to \mathbb{R}^n, \quad A \mapsto f(A) = A^{-1} b \qquad ①$$

Differentiable
↳ Cramer's rule
↳ Neumann series

} Dennfchard ②

Now show: The mapping $g: GL(n) \to GL(n)$ with $g(A) = A^{-1}$ is differentiable and

$$g'(A)C = -A^{-1} C A^{-1} \qquad C \in \mathbb{R}^{n \times n}$$

Take derivative of $(A + tC)(A + tC)^{-1} = I$

$$0 = C(A + tC)^{-1} + (A + tC) \frac{d}{dt}(A + tC)^{-1}$$

$$\frac{d}{dt}(A + tC)^{-1} = \underline{-(A + tC)^{-1} C (A + tC)^{-1}}$$

$$\underline{g'(A)C} = \frac{d}{dt}(A + tC)^{-1}\Big|_{t=0} = -A^{-1} C A^{-1}$$

Now back to our condition number

$$f(A) = A^{-1}b$$

$$f'(A)C = -A^{-1}C\underline{A^{-1}b} = -A^{-1}Cx$$

$$\kappa_{rel} = \frac{\|A\|}{\|f(A)\|}\|f'(A)\|_{op}$$

$$= \frac{\|A\|}{\|x\|}\sup_{\|C\|=1}\|-A^{-1}Cx\| \leq$$

$$\leq \frac{\|A\|}{\|x\|}\sup_{\|C\|=1}\|A^{-1}\|\,\|C\|\,\|x\|$$

$$= \|A\|\,\|A^{-1}\| = \kappa(A)$$

Perturbation bound

$$\frac{\|x-\tilde{x}\|}{\|x\|} \leq \|A\|\,\|A^{-1}\|\,\frac{\|A-\tilde{A}\|}{\|A\|} + o(\|\delta A\|)$$

$$\|\delta A\| \to 0$$

Now least squares

Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, full rank$(A) = n$, and x the unique solution

$$\|b - Ax\|_2 = \min$$

Assume $x \neq 0$ and define angle v between b and $R(A)$ of A as

$$\sin(v) = \frac{\|b-Ax\|_2}{\|b\|_2} = \frac{\|r\|_2}{\|b\|_2}$$

Then

(a) rel condition number w.r.t. perturbations in b is

$$k_{rel} \leq \frac{k_2(A)}{\cos(\vartheta)}$$

(b) rel cond. number w.r.t. A

$$k_{rel} \leq k_2(A) + k_2(A)^2 \tan(\vartheta)$$

Let's look at (a) first:

Solution

$$x = \phi(b) = (A^T A)^{-1} A^T b \qquad (1)$$

ϕ is differentiable because is linear in b

$$\phi'(b) = (A^T A)^{-1} A^T \qquad (2)$$

Assemble condition number

$$k_{rel} = \frac{\|b\|}{\|x\|} \|\phi'(b)\|$$

$$= \frac{\|b\|}{\|x\|} \|(A^T A)^{-1} A^T\|_2 \frac{\|A\|_2}{\|A\|_2}$$

- For a full column rank matrix A

$$k_2(A) = \|A\|_2 \|(A^T A)^{-1} A^T\|_2$$

- Recall

$$\frac{\|b\|}{\|Pb\|} = \frac{1}{\cos(\vartheta)}$$

where P is proj onto R(A)

$$P = A(A^T A)^{-1} A^T$$

$$k_{rel} = \frac{\|b\|}{\|A\|\,\|x\|}\, k_2(A)$$

$$\leq \frac{\|b\|}{\|Ax\|}\, k_2(A) = \frac{\|b\|}{\|A(A^TA)^{-1}A^Tb\|}\, k_2(A)$$

$$= \frac{\|b\|}{\|Pb\|}\, k_2(A) = \frac{k_2(A)}{\cos(\vartheta)}$$

Perturbation bound

$$\frac{\|x-\tilde{x}\|}{\|x\|} \leq \frac{k_2(A)}{\cos(\vartheta)} \frac{\|\delta b\|}{\|b\|} + o(\|\delta b\|), \quad \|\delta b\| \to 0$$

Now (b): Our function

$$\phi(A) = (A^TA)^{-1}A^Tb \qquad (1)$$

ϕ is differentiable, convinced us already earlier (2)

Construct derivative by using equation that defines ϕ

$$(A+tC)^T(A+tC)\,\underbrace{\phi(A+tC)}_{=x} = (A+tC)^Tb$$

Take derivative

$$C^T(A+tC)\,\phi(A+tC) +$$
$$(A+tC)^TC\,\phi(A+tC) +$$
$$(A+tC)^T(A+tC)\,\underbrace{\frac{d}{dt}\phi(A+tC)}_{\phi'(A)C} = C^Tb$$

Now for $t=0$ obtain

$$C^TA\,\underbrace{\phi(A)}_{=x} + A^TC\,\underbrace{\phi(A)}_{=x} + A^TA\,\phi'(A)C = C^Tb$$

$$c^T A x + A^T C x + A^T A \, \phi'(A) C = c^T b$$

$$\phi'(A) C = (A^T A)^{-1} \left(c^T (b - Ax) - A^T C x \right)$$

$$= (A^T A)^{-1} c^T (b - Ax) - (A^T A)^{-1} A^T C x$$

Estimate norm

$$\|\phi'(A)\| = \sup_{\|c\|=1} \|\phi'(A) c\|$$

$$\leq \|(A^T A)^{-1}\| \, \|b - Ax\| + \|(A^T A)^{-1} A^T\| \, \|x\|$$

Condition number

$$k_{rel} = \frac{\|A\|}{\|\phi(A)\|} \|\phi'(A)\|$$

$$\leq \frac{\|A\|}{\cancel{\|x\|}} \|(A^T A)^{-1} A^T\| \, \cancel{\|x\|} + \frac{\|A\|}{\|x\|} \|(A^T A)^{-1}\| \, \|r\| \frac{\|A\|}{\|A\|}$$

$$= \underbrace{\|A\| \, \|(A^T A)^{-1} A^T\|}_{k_2(A)} + \underbrace{\|A\|^2 \, \|(A^T A)^{-1}\|}_{k_2(A^T A) = k_2(A)^2} \frac{\|r\|}{\|A\| \, \|x\|}$$

$$\frac{\|r\|}{\|A\| \, \|x\|} = \frac{\|r\|}{\|b\|} \frac{\|b\|}{\|A\| \, \|x\|} \leq \sin(v) \frac{1}{\cos(v)} = \tan(v)$$

Again this gives perturbation bound.

Now eigenvalues

Let $\lambda_0 \in \mathbb{C}$ be a simple eigenvalue of $A \in \mathbb{C}^{n\times n}$.

Then there is a cont. diff. mapping

$$\lambda : V \subset \mathbb{C}^{n\times n} \to \mathbb{C}, \quad B \mapsto \lambda(B)$$

from a neighborhood V of A in $\mathbb{C}^{n\times n}$ such that

$$\lambda(A) = \lambda_0$$

and $\lambda(B)$ is a simple eigenvalue of B for all $B \in V$.

If x_0 is an eigenvector of A for λ_0 and y_0 is a (left) eigenvector of $A^* = A^H$ for the eigenvalue

$$\overline{\lambda_0}$$

so that

$$Ax_0 = \lambda_0 x_0 \qquad \text{and} \qquad A^* y_0 = \overline{\lambda_0}\, y_0,$$

then the derivative of λ at A satisfies

$$\lambda'(A)C = \frac{\langle Cx_0, y_0 \rangle}{\langle x_0, y_0 \rangle} \qquad \text{for all } C \in \mathbb{C}^{n\times n}.$$

Similar to before, we consider the matrix

$$A + tC, \qquad C \in \mathbb{C}^{n\times n}$$

and want to understand the solution map $\lambda(t)$.

$\ulcorner$ $A \in \mathbb{C}^{n \times n}$ has n eigenvalues of which $d \leq n$ are distinct, then the char. polynomial is

$$\chi_A(\lambda) = \det(A - \lambda I) = \prod_{i=1}^{n} (\lambda_i - \lambda)$$
$$= \prod_{i=1}^{d} (\lambda_i' - \lambda)^{\mu(\lambda_i')}$$

with $\mu(\lambda_i')$ being the algebraic multiplicity of the eigenvalue λ_i'.

Simple means

$$\mu(\lambda_0) = 1$$

$\lrcorner$

Because we have a simple eigenvalue

$$\chi_A'(\lambda_0) \neq 0$$

Define function

$$F(t, \lambda) = \chi_{A+tC}(\lambda)$$

Apply implicit function theorem to show

$\lambda : (-\varepsilon, \varepsilon) \to \mathbb{C}$ is cont. diff. with $\lambda(0) = \lambda_0$

$\lambda(t)$ simple EV of $A + tC$

$\hookrightarrow$ Consider (t_0, λ_0), $t_0 = 0$

$\hookrightarrow$ $F(t_0, \lambda_0) = \chi_A(\lambda_0) = 0$

$\hookrightarrow \frac{\partial}{\partial \lambda} F(t,\lambda)\Big|_{t_0,\lambda_0} = \chi'_{A+tC}(\lambda)\Big|_{t_0,\lambda_0} = \chi'_A(\lambda) \neq 0$

$\Rightarrow$ there exists $\lambda : (-\varepsilon, \varepsilon) \to \mathbb{C}$ as stated above.

Similarly, use that λ_0 is simple to determine cont. diff

$$x : (-\varepsilon, \varepsilon) \to \mathbb{C}^n, \quad t \mapsto x(t)$$

such that $x(0) = x_0$ and $x(t)$ is eigenvector of $A+tC$ for eigenvalue $\lambda(t)$.

Now we can differentiate w.r.t. t the equation

$$(A+tC)\, x(t) = \lambda(t)\, x(t)$$

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Computing eigenvalues
- Perturbation bounds

Today

- More on computing the eigenvalues

Announcements

- Homework 3 posted, is due Mon, Oct 21 before class

Numerically finding eigenvalues

For a matrix $A \in \mathbb{C}^{n \times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Most relevant problems:

- A symmetric (and large)
- A spd (and large)
- A stochastic matrix, i.e., all entries $0 \leq a_{ij} \leq 1$ are probabilities, and thus $\sum_j a_{ij} = 1$.

How hard are they to find numerically?

How hard are they to find numerically?

- This is a nonlinear problem.

How hard are they to find numerically?

- This is a nonlinear problem.
- How difficult is this?

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

- Must use an iterative algorithm

How hard are they to find numerically?

- This is a nonlinear problem.

- How difficult is this? Eigenvalues are the roots of the characteristic polynomial. Also, any polynomial is the characteristic polynomial of a matrix $\rightsquigarrow$ For matrices larger than 4×4, eigenvalues cannot be computed in closed form (Abel's theorem).

- Must use an iterative algorithm $\rightsquigarrow$ this is fundamentally different from what we have seen previously when solving systems of *linear* equations! These algorithms (LU, QR) give the *exact* solution in *exact* arithmetic in finite number of steps. We *cannot* expect something similar for computing eigenvalues!

Condition of finding eigenvalues of a matrix

The absolute condition number of determining a simple eigenvalue λ_0 of a matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ with respect to the $\|\cdot\|_2$ is

$$\kappa_{\text{abs}} = \frac{1}{|\cos(\angle(\boldsymbol{x},\boldsymbol{y}))|}, \qquad \cos(\angle(\boldsymbol{x},\boldsymbol{y})) = \frac{|\langle \boldsymbol{x},\boldsymbol{y}\rangle|}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}$$

and the relative condition number is

$$\kappa_{\text{rel}} = \frac{\|\boldsymbol{A}\|}{|\lambda_0 \cos(\angle(\boldsymbol{x},\boldsymbol{y}))|},$$

where $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ for the eigenvalue λ_0 ($\boldsymbol{A}\boldsymbol{x} = \lambda_0\boldsymbol{x}$) and $\boldsymbol{y}$ an adjoint eigenvector ($\boldsymbol{A}^H\boldsymbol{y} = \bar{\lambda}_0\boldsymbol{y}$).

Sketch of proof $\rightsquigarrow$ board $\rightsquigarrow$ finish proof sketch

(see also Deuflhard, Theorem 5.2)

Last time

↳ looked $A + tC$, $C \in \mathbb{C}^{n \times n}$

↳ cont. diff

$$\lambda : (-\varepsilon, \varepsilon) \to \mathbb{C}$$

so that $\lambda(0) = \lambda_0$ and $\lambda(t)$ is a simple EV of $A + tC$

↳ cont. diff

$$x : (-\varepsilon, \varepsilon) \to \mathbb{C}^n$$

so that $x(0) = x_0$ and $x(t)$ is EV of $A + tC$

Now we can differentiate w.r.t. t the equation

$$(A + tC)\, x(t) = \lambda(t)\, x(t)$$

to obtain

$$C x(t) + (A + tC) x'(t) = \lambda'(t) x(t) + \lambda(t) x'(t)$$

At $t = 0$

$$C x_0 + A x'(0) = \lambda'(0) x_0 + \lambda_0 x'(0)$$

Multiply from right with y_0

$$\langle C x_0, y_0 \rangle + \underline{\langle A x'(0), y_0 \rangle} = \langle \lambda'(0) x_0, y_0 \rangle + \underline{\langle \lambda_0 x'(0), y_0 \rangle}$$

Now use that

$$\langle \lambda'(0) x_0, y_0 \rangle = \underline{\lambda'(0) \langle x_0, y_0 \rangle}$$

and

$$\langle A x'(0), y_0 \rangle = \langle x'(0), A^H y_0 \rangle$$
$$= \langle x'(0), \bar{\lambda}_0 y_0 \rangle$$
$$= \underline{\langle \lambda_0 x'(0), y_0 \rangle}$$

Obtain derivative

$$\lambda'(0) = \frac{\langle C x_0, y_0 \rangle}{\langle x_0, y_0 \rangle}$$

$$\lambda'(A) C = \lambda'(0) = \frac{\langle C x_0, y_0 \rangle}{\langle x_0, y_0 \rangle} .$$

The absolute condition number of determining a simple eigenvalue λ_0 of a matrix $A \in \mathbb{C}^{n \times n}$ with respect to the 2-norm is

$$\kappa_{abs} = \| \lambda'(A) \| = \frac{\|x\| \, \|y\|}{|\langle x, y \rangle|} = \frac{1}{|\cos(\measuredangle(x,y))|}$$

and the rel. condition number

$$\kappa_{rel} = \frac{\|A\|}{\|\lambda_0\|} \| \lambda'(A) \| = \frac{\|A\|}{|\lambda_0 \cos(\measuredangle(x,y))|}$$

where x is eigenvector of A for λ_0
y is left eigenvector ...

For all $C \in \mathbb{C}^{n \times n}$, we have

$$\underline{|\langle Cx, y\rangle|} \leq \|Cx\| \, \|y\| \leq \|C\| \, \underline{\|x\| \, \|y\|}$$

Equality if $C = yx^H$

$$\|\lambda'(A)C\| = \sup_{C \neq 0} \frac{|\langle Cx, y\rangle / \langle x, y\rangle|}{\|C\|}$$

$$\underset{=}{C = yx^H} \quad \frac{|\langle yx^H x, y\rangle / \langle x, y\rangle|}{\|yx^H\| \, [= \|x\| \, \|y\|]}$$

$$= \frac{|\, \|yx^H\|^2 / \langle x, y\rangle |}{\|y\| \, \|x\|}$$

$$= \frac{\|y\|^2 \, \|x\|^2}{\|y\| \, \|x\|} \, \frac{1}{|\langle x, y\rangle|}$$

$$= \frac{\|x\| \, \|y\|}{|\langle x, y\rangle|} = \frac{1}{\cos(\measuredangle(x, y))}$$

The Power Method

Recap: Power method

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ be diagonalizable matrix and λ_1 be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Let $\boldsymbol{x}_0$ be an initial guess that is not orthogonal to the eigenspace of λ_1, then $\boldsymbol{x}_k$ obtained via the iterations

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \tag{3}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{z}_{k+1} / \|\boldsymbol{z}_{k+1}\|_2 \tag{4}$$

will converge to the normalized eigenvector of $\boldsymbol{A}$ corresponding to λ_1 for $k \to \infty$.

This process is called the power method.

Proof $\rightsquigarrow$ last week

Recap Power method (cont'd)

Start with initial guess $\boldsymbol{x}_0$ and then iterate

- Compute matrix-vector product and normalize it

$$\boldsymbol{x}_k = \frac{\boldsymbol{A}\boldsymbol{x}_{k-1}}{\|\boldsymbol{A}\boldsymbol{x}_{k-1}\|}$$

- Obtain eigenvalue estimate (note that $\|\boldsymbol{x}_k\| = 1$)

$$\lambda_1^{(k)} = \boldsymbol{x}_k^H \boldsymbol{A}\boldsymbol{x}_k$$

- Test for convergence? How? Compute residual

$$\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{x}_k - \lambda_1^{(k)}\boldsymbol{x}_k$$

  and terminate if the error estimate is small enough (bound if $\boldsymbol{A}$ Hermitian, heuristic otherwise)

$$\min_i |\lambda_i - \lambda_1^{(k)}| \leq \frac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{x}_k\|} < \epsilon$$

- The power method converges linearly

$$\|\boldsymbol{x}_k - (\pm \boldsymbol{v}_1)\| \in \mathcal{O}((|\lambda_2|/|\lambda_1|)^k)$$

- If $\boldsymbol{A}$ is normal, then the eigenvalue estimate converges a bit faster but still linearly

$$|\lambda_1^{(k)} - \lambda_1| \in \mathcal{O}((|\lambda_2|/|\lambda_1|)^{2k})$$

- The power method is fast when the dominant eigenvalue is well separated from the rest

- This conclusion is rather general for all iterative methods, convergence is often good if eigenvalues are well separated and bad otherwise

- The power method is typically too slow to be used in practice

we have that eigenvector converges as

$$\boxed{\|x_k - (\pm v_1)\| \in \mathcal{O}\left(\left|\lambda_2/\lambda_1\right|^k\right)}$$

linear convergence: $\exists c > 0$

$$\lim_{k\to\infty} \frac{\|x - x_{k+1}\|}{\|x - x_k\|} < c, \quad c < 1$$

quadratic convergence

$$\lim_{k\to\infty} \frac{\|x - x_{k+1}\|}{\|x - x_k\|^2} < c$$

Try linear convergence

$$\lim_{k\to\infty} \frac{\|x_{k+1} - (\pm v_1)\|}{\|x_k - (\pm v_1)\|} = \lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|^{k+1}}{\left|\lambda_2/\lambda_1\right|^k}$$

$$= \lim_{k\to\infty} \left|\lambda_2/\lambda_1\right| < c \leq 1$$

Try quadratic

$$\lim_{k\to\infty} \frac{\|x_{k+1} - (\pm v_1)\|}{\|x_k - (\pm v_1)\|^2} = \lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|^{k+1}}{\left|\lambda_2/\lambda_1\right|^{2k}}$$

$$= \lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|}{\left|\lambda_2/\lambda_1\right|^k} \text{ diverges}$$

Now eigenvalue

$$\|\lambda_1^{(k)} - \lambda_1\| \in \mathcal{O}\left(\left|\lambda_2/\lambda_1\right|^{2k}\right)$$

Quadratic

$$\lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|^{2(k+1)}}{\left|\lambda_2/\lambda_1\right|^{2k\times 2}} = \lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|^{2}}{\left|\lambda_2/\lambda_1\right|^{2k}} \text{ diverges}$$

linear

$$\lim_{k\to\infty} \frac{\left|\lambda_2/\lambda_1\right|^{2(k+1)}}{\left|\lambda_2/\lambda_1\right|^{2k}} = \lim_{k\to\infty} \left|\lambda_2/\lambda_1\right|^{2} < 1$$

The inverse power method

For any μ not an eigenvalue of $\boldsymbol{A}$:

- The eigenvectors of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are the same as the eigenvectors of $\boldsymbol{A}$
- The eigenvalues of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are $\{(\lambda_j - \mu)^{-1}\} \rightsquigarrow$ why useful?

The inverse power method

For any μ not an eigenvalue of $\boldsymbol{A}$:

- The eigenvectors of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are the same as the eigenvectors of $\boldsymbol{A}$
- The eigenvalues of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are $\{(\lambda_j - \mu)^{-1}\} \rightsquigarrow$ why useful?

Thus, if we have a good estimate μ of an eigenvalue λ_J of matrix $\boldsymbol{A}$, then

$$\frac{|\lambda_j - \mu|^{-1}}{|\lambda_J - \mu|^{-1}} \ll 1\,, \qquad j \neq J$$

and thus the power method applied to $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ converges rapidly to $\boldsymbol{v}_J$:

$$(\boldsymbol{A} - \mu \boldsymbol{I})\boldsymbol{y}_{k+1} = \boldsymbol{x}_k \tag{5}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1} / \|\boldsymbol{y}_{k+1}\| \tag{6}$$

- What do we need to keep in mind?

The inverse power method

For any μ not an eigenvalue of $\boldsymbol{A}$:

- The eigenvectors of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are the same as the eigenvectors of $\boldsymbol{A}$
- The eigenvalues of $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ are $\{(\lambda_j - \mu)^{-1}\} \rightsquigarrow$ why useful?

Thus, if we have a good estimate μ of an eigenvalue λ_J of matrix $\boldsymbol{A}$, then

$$\frac{|\lambda_j - \mu|^{-1}}{|\lambda_J - \mu|^{-1}} \ll 1\,, \qquad j \neq J$$

and thus the power method applied to $(\boldsymbol{A} - \mu \boldsymbol{I})^{-1}$ converges rapidly to $\boldsymbol{v}_J$:

$$(\boldsymbol{A} - \mu \boldsymbol{I})\boldsymbol{y}_{k+1} = \boldsymbol{x}_k \tag{5}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{y}_{k+1} / \|\boldsymbol{y}_{k+1}\| \tag{6}$$

- What do we need to keep in mind? Costs: Requires matrix solve in every iteration; same matrix, different right-hand sides ($\rightsquigarrow$ LU and Cholesky decompositions)
- This algorithm is used in practice to find eigen*vectors* if the eigen*values* are already known

Rayleigh quotient iterations

- The convergence speed of the inverse power method increases with a better eigenvalue estimate $\rightsquigarrow$ what could we do?

Rayleigh quotient iterations

- The convergence speed of the inverse power method increases with a better eigenvalue estimate $\rightsquigarrow$ what could we do?
- Combine estimating eigenvalue and eigenvector $\rightsquigarrow$ Rayleigh quotient iteration

Accelerated version of the inverse power method using changing shifts:

- Choose starting vector $\boldsymbol{x}^0$ with $\|\boldsymbol{x}^0\| = 1$. Compute $\lambda^{(0)} = (\boldsymbol{x}^0)^T A\boldsymbol{x}^0$.
- For $i = 0, 1, \ldots$ do

$$(A - \lambda^{(k)} I)\boldsymbol{x}^{k+1} = \boldsymbol{x}^k, \quad \boldsymbol{y}^{k+1} = \boldsymbol{x}^{k+1}/\|\boldsymbol{x}^{k+1}\|.$$

- Compute $\lambda^{(k+1)} = (\boldsymbol{y}^{k+1})^T A\boldsymbol{y}^{k+1}$, and go back.

If it converges (depends on starting point), then it converges *cubically* $\rightsquigarrow$ details in Trefethen & Bau
(This is the only method we will see that converges so quickly!)

The QR algorithm

The QR algorithm

The power method is not well suited for finding all eigenvalues of a matrix $\boldsymbol{A}$

Idea of the QR method: Build a matrix A' that shares the eigenvalues of $\boldsymbol{A}$ via similarity transformations

$$\boldsymbol{A}' = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}\,, \qquad \boldsymbol{A}, \boldsymbol{A}' \text{ have the same eigenvalues}$$

and for which we know the eigenvalues. $\rightsquigarrow$ What matrix would we like $\boldsymbol{A}'$ to be?

The QR algorithm

The power method is not well suited for finding all eigenvalues of a matrix $\boldsymbol{A}$

Idea of the QR method: Build a matrix A' that shares the eigenvalues of $\boldsymbol{A}$ via similarity transformations

$$\boldsymbol{A}' = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}\,, \qquad \boldsymbol{A}, \boldsymbol{A}' \text{ have the same eigenvalues}$$

and for which we know the eigenvalues. $\rightsquigarrow$ What matrix would we like $\boldsymbol{A}'$ to be?

The QR algorithm for finding eigenvalues is as follows ($A_0 := A$), and for $k = 0, 1, \ldots$:

- Compute QR decomposition of A_k, i.e., $A_k = Q_k R_k$.
- $A_{k+1} := R_k Q_k$, $k := k + 1$ and go back.

All iterates $A_1, A_2, \ldots$ have the same eigenvalues because

$$Q_k A_{k+1} Q_k^T = Q_k R_k Q_k Q_k^T = Q_k R_k = A_k$$

and A_k converges to a *diagonal* matrix if A is Hermitian and eigenvalues well separated

Intuition why QR method converges

Think of it as the power method applied to many linearly independent vectors $z_1^{(0)}, \ldots, z_n^{(0)}$ at once

Define

$$Z^{(0)} = \begin{bmatrix} | & & | \\ z_1^{(0)} & \ldots & z_n^{(0)} \\ | & & | \end{bmatrix}$$

and define

$$Z^{(k)} = A^k Z^{(0)} = \begin{bmatrix} | & & | \\ z_1^{(k)} & \ldots & z_n^{(k)} \\ | & & | \end{bmatrix}$$

Recall that in the power method we had to re-normalize after each step $\rightsquigarrow$ now we have multiple vectors and therefore also need to orthogonalize $\rightsquigarrow$ QR

With orthogonalization after each iteration, we obtain the algorithm

1. $Z^{(k)} = A\bar{Q}^{(k-1)}$
2. $\bar{Q}^{(k)}R^{(k)} = Z^{(k)}$
3. $A^{(k)} = (\bar{Q}^{(k)})^T A\bar{Q}^{(k)}$

$\rightsquigarrow$ equivalent to the QR method

Summary: Let the QR algorithm be applied to a symmetric real matrix A with well separated eigenvalues

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$$

and eigenvectors matrix V that has nonsingular leading principal submatrices (all upper-left 1×1, 2×2, ... submatrices). Then, for $k \to \infty$, the iterates $A^{(k)}$ converge linearly in

$$\mathcal{O}\left(\max_j \frac{|\lambda_{j+1}|^k}{|\lambda_j|^k}\right)$$

to $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ and $\bar{Q}^{(k)}$ to V (up to $\pm$)

- The convergence of the QR algorithm is closely related to that of the power method: It is only fast if all eigenvalues are well separated

- For more general (e.g., non-symmetric) matrices in complex arithmetic, the algorithm converges to the Schur decomposition $A = UTU^H$ with triangular matrix T and unitary matrix $U \rightsquigarrow$ read eigenvalues from diagonal of T

- The work *per iteration* of the basic QR algorithm that we discussed is in $\mathcal{O}(n^3)$ because of the QR factorization in each step; the power method has cost $\mathcal{O}(n^2)$ (mat-vec) per iteration

- There are several key improvements to the basic QR algorithm that bring down the cost per iteration to $\mathcal{O}(n^2)$ (Hessenberg matrices)

- There also can be shifts (compare power method) to accelerate the convergence

- As always with linear algebra routines, the "best" are implemented in LAPACK and can be called via Matlab, numpy, etc

Eigenvalues in Matlab

- In MATLAB, sophisticated variants of the **QR algorithm** (LAPACK library) are implemented in the function *eig*:

$$\Lambda = eig(A)$$

$$[X, \Lambda] = eig(A)$$

- For large or sparse matrices, iterative methods based on the **Arnoldi iteration** (ARPACK library), can be used to obtain a few of the largest/smallest/closest-to-μ eigenvalues:

$$\Lambda = eigs(A, n_{eigs})$$

$$[X, \Lambda] = eigs(A, n_{eigs})$$

- The **Schur decomposition** is provided by $[U, T] = schur(A)$.

Conclusions/summary

- Eigenvalues are **well-conditioned** for **unitarily diagonalizable matrices** (includes Hermitian matrices), but ill-conditioned for nearly non-diagonalizable matrices.
- Eigenvectors are **well-conditioned** only when **eigenvalues are well-separated**.
- Eigenvalue algorithms are **always iterative**.
- The **power method** and its variants can be used to find the **largest or smallest eigenvalue**, and they converge fast if there is a **large separation** between the target eigenvalue and nearby ones.
- Estimating **all eigenvalues and/or eigenvectors** can be done by combining the power method with QR factorizations.
- MATLAB has high-quality implementations of sophisticated variants of these algorithms.

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Computing eigenvalues

Today

- Singular value decomposition

Announcements

- Homework 3 posted, is due Mon, Oct 21 before class

Recap: Power method for computing eigenvectors and eigenvalues

Let $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ be diagonalizable matrix and λ_1 be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Let $\boldsymbol{x}_0$ be an initial guess that is not orthogonal to the eigenspace of λ_1, then $\boldsymbol{x}_k$ obtained via the iterations

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \tag{7}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{z}_{k+1} / \|\boldsymbol{z}_{k+1}\|_2 \tag{8}$$

will converge to the normalized eigenvector of $\boldsymbol{A}$ corresponding to λ_1 for $k \to \infty$.

This process is called the power method.

Recap: The QR algorithm

The power method is not well suited for finding all eigenvalues of a matrix $\boldsymbol{A}$

Idea of the QR method: Build a matrix A' that shares the eigenvalues of $\boldsymbol{A}$ via similarity transformations

$$\boldsymbol{A}' = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}\,, \qquad \boldsymbol{A}, \boldsymbol{A}' \text{ have the same eigenvalues}$$

and for which we know the eigenvalues. $\rightsquigarrow$ What matrix would we like $\boldsymbol{A}'$ to be?

Recap: The QR algorithm

The power method is not well suited for finding all eigenvalues of a matrix $\boldsymbol{A}$

Idea of the QR method: Build a matrix A' that shares the eigenvalues of $\boldsymbol{A}$ via similarity transformations

$$\boldsymbol{A}' = \boldsymbol{P}^{-1}\boldsymbol{A}\boldsymbol{P}\,, \qquad \boldsymbol{A}, \boldsymbol{A}' \text{ have the same eigenvalues}$$

and for which we know the eigenvalues. $\rightsquigarrow$ What matrix would we like $\boldsymbol{A}'$ to be?

The QR algorithm for finding eigenvalues is as follows ($A_0 := A$), and for $k = 0, 1, \ldots$:

- Compute QR decomposition of A_k, i.e., $A_k = Q_k R_k$.
- $A_{k+1} := R_k Q_k$, $k := k + 1$ and go back.

All iterates $A_1, A_2, \ldots$ have the same eigenvalues because

$$Q_k A_{k+1} Q_k^T = Q_k R_k Q_k Q_k^T = Q_k R_k = A_k$$

and A_k converges to a *diagonal* matrix if A is Hermitian and eigenvalues well separated

Singular Value Decomposition (SVD)

Let $A \in \mathbb{C}^{m \times n}$. A singular value decomposition of A is a factorization

$$A = U \Sigma V^H ,$$

where

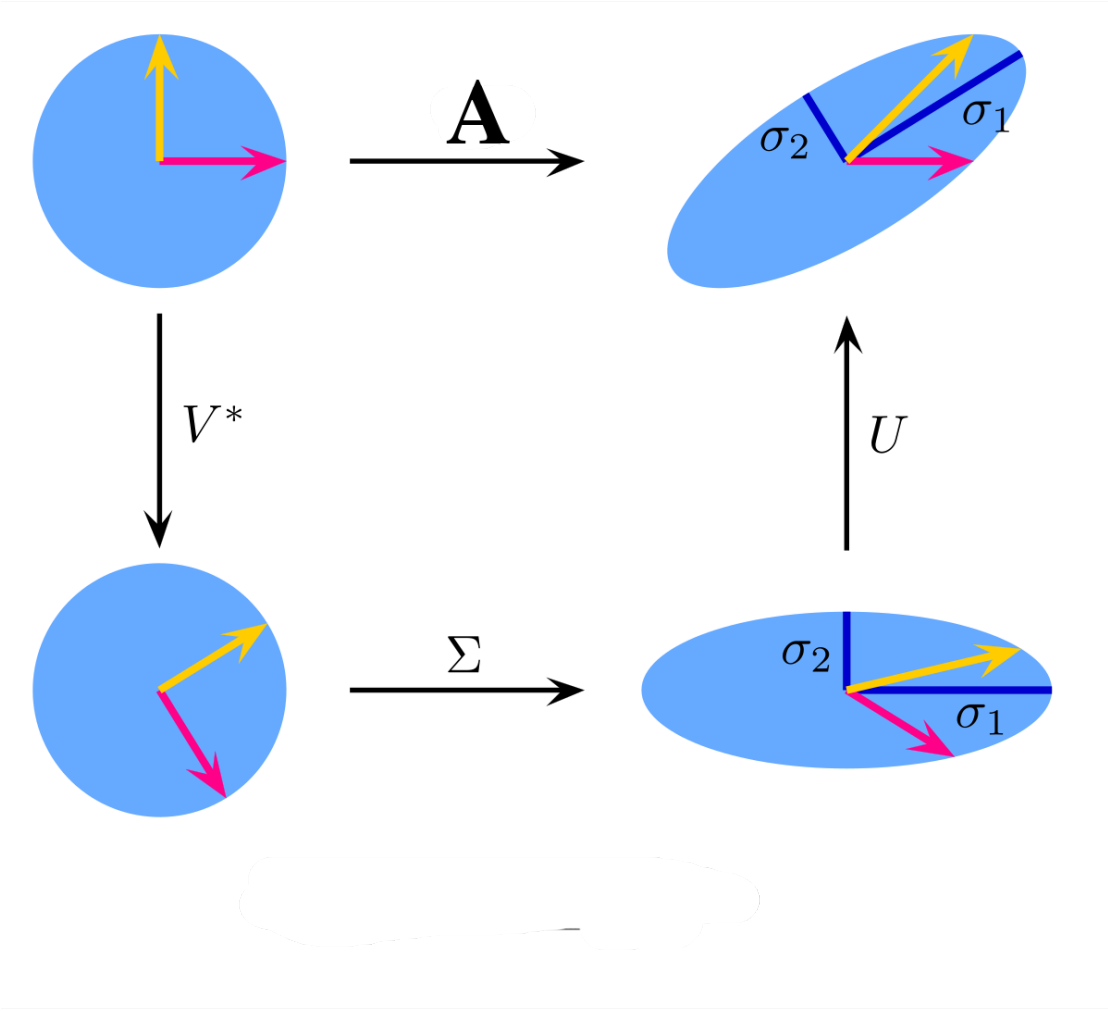
$$U \in \mathbb{C}^{m \times m} \text{ is unitary} \tag{9}$$

$$V \in \mathbb{C}^{n \times n} \text{ is unitary} \tag{10}$$

$$\Sigma \in \mathbb{R}^{m \times n} \text{ is diagonal.} \tag{11}$$

Additionally, the diagonal entries σ_j of Σ are non-negative and in non-decreasing order so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p \in \min(m, n)$.

- The diagonal matrix Σ is real and has the same shape as A even when A is not square
- The matrices U and V are always square



[Edited from figure by Georg-Johann, Wikipedia]

The image of the unit sphere under a map A is a hyperellipse (in $\mathbb{R}^m$). Thus, with $A = U\Sigma V^H$, have

- The unitary map V^H preserves the sphere (rotating a sphere is a sphere)
- The diagonal matrix Σ stretches the sphere into a hyperellipse aligned with the canonical basis
- The unitary map U rotates or reflects the hyperellipse without changing shape

Existence and uniqueness of SVD

Theorem: Every matrix $A \in \mathbb{C}^{m \times n}$ has a singular value decomposition. Furthermore, the singular values $\{\sigma_j\}$ are uniquely determined, and, if A is square and the σ_j are distinct, the left and right singular vectors $\{u_j\}$ and $\{v_j\}$ are uniquely determined up to complex signs (i.e., complex scalar factors of absolute value 1.) Proof in Trefethen & Bau.

Reduced SVD

$$\underbrace{\begin{bmatrix} \\ \\ A \\ \\ \\ \end{bmatrix}}_{m \times n} = \underbrace{\begin{bmatrix} \\ \\ \hat{U} \\ \\ \\ \end{bmatrix}}_{m \times n} \underbrace{\begin{bmatrix} \\ \\ \hat{\Sigma} \\ \\ \\ \end{bmatrix}}_{n \times n} \underbrace{\begin{bmatrix} \\ \\ \hat{V}^H \\ \\ \\ \end{bmatrix}}_{n \times n}$$

SVD vs. eigenvalue decomposition

SVD expresses a matrix in proper bases for domain and range space to represent it as a diagonal matrix

$$A = U\Sigma V^H$$

We have seen something similar with eigenvectors: A non-defective square matrix A can be expressed as a diagonal matrix of eigenvalues Λ if the range and domain are presented in a basis of eigenvectors

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

There are fundamental differences between the SVD and eigenvalue decomposition

- SVD uses two different bases (left and right singular vectors); eigenvalue decomposition uses just one (eigenvectors)
- SVD uses orthonormal bases, whereas eigenvalue basis generally is not orthogonal
- Not all matrices (even square ones) have an eigendecomposition, but all matrices (even rectangular ones) have a singular value decomposition

Typically, eigenvalues tell us something about the behavior of iterative processes that involve the matrix A such as A^k and e^{tA}

Singular values tend to tell us something about A itself

The SVD and matrix properties

In the following:

- The matrix A is of dimension $m \times n$
- $p = \min(m, n)$
- $r \leq p$ is the number of non-zero singular values of A

We now list how the SVD is related to fundamental properties of the matrix A

- The rank of A is r, the number of non-zero singular values.
- The range (column span) of A is $\mathrm{span}(u_1, \ldots, u_r)$, the kernel is $\mathrm{span}(v_{r+1}, \ldots, v_n)$
- $\|A\|_2 = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$
- For square A, $|\det(A)| = \prod_{i=1}^{m} \sigma_i$
- The non-zero singular values of A are the square roots of the non-zero eigenvalues of $A^H A$ and AA^H $\rightsquigarrow$ proof

The non-zero SV of A are the square roots of the non-zero EV of $A^H A$ and AA^H

$$A^H A = (U \Sigma V^H)^H (U \Sigma V^H)$$
$$= V \Sigma^H \underbrace{U^H U} \Sigma V^H$$
$$= V \Sigma^H \Sigma V^H$$

Need that $\Sigma^H \Sigma$ is diagonal, then we have eigendecomposition of $A^H A$

$\Sigma^H \Sigma$ has size $n \times n$ and the diagonal $\sigma_1^2, \dots, \sigma_p^2$

$$\Lambda = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_p^2 \end{bmatrix} = \Sigma^H \Sigma$$

Full SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times m}\underbrace{\boldsymbol{\Sigma}}_{m\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Reduced SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times n}\underbrace{\boldsymbol{\Sigma}}_{n\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Rank-revealing SVD of a rank r matrix with dimension $m \times n$

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times r}\underbrace{\boldsymbol{\Sigma}}_{r\times r}\underbrace{\boldsymbol{V}^H}_{r\times n}$$

Representing matrices via sums of rank-one matrices

Represent A as a sum of M rank-one matrices

$$A = \sum_{j=1}^{M} A^{(j)}$$

There are many possibilities of choosing $A^{(j)}$

- Let $A^{(j)}$ contain the j-th of the m rows of A
- Let $A^{(j)}$ contain the j-th of the n columns of A
- Let $A^{(j)}$ contain one of the mn entries of A
- ...

What is a property of a "sum representation" that we like to see in numerical analysis?

Representing matrices via sums of rank-one matrices

Represent A as a sum of M rank-one matrices

$$A = \sum_{j=1}^{M} A^{(j)}$$

There are many possibilities of choosing $A^{(j)}$

- Let $A^{(j)}$ contain the j-th of the m rows of A
- Let $A^{(j)}$ contain the j-th of the n columns of A
- Let $A^{(j)}$ contain one of the mn entries of A
- ...

What is a property of a "sum representation" that we like to see in numerical analysis?
$\rightsquigarrow$ we can truncate it after a few terms and get a good approximation

$$A \approx \sum_{j=1}^{M'} A^{(j)}$$

with $M' \ll M$

SVD for low-rank approximation

Consider the SVD $A = U\Sigma V^H$, then

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^H$$

Let us truncate the sum after $1 \leq q \leq r$ terms and define

$$A_q = \sum_{j=1}^{q} \sigma_j u_j v_j^H.$$

Then, A_q is a best rank q approximation of A in the $\|\cdot\|_2$ norm

$$\|A - A_q\|_2 = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ \mathrm{rank}(B) \leq q}} \|A - B\|_2$$

$\rightsquigarrow$ proof

Best rank q approximation:

Consider $A = U \Sigma V^H$

$$A_r = \sum_{j=1}^{r} \sigma_j u_j v_j^H$$

want show
$$\begin{cases} (1) & \|A - A_q\|_2 = \sigma_{q+1} \\ (2) & \|A - A_q\|_2 = \inf\limits_{\substack{B \in \mathbb{C}^{m \times n} \\ \operatorname{rank}(B) \le q}} \|A - B\|_2 \end{cases}$$

(1) $$A - A_q = \sum_{i=q+1}^{r} \sigma_i u_i v_i^H$$

$$A - A_q = \begin{bmatrix} | & & | \\ u_{q+1} & \cdots & u_m \\ | & & | \end{bmatrix} \begin{bmatrix} \sigma_{q+1} & & \\ & \ddots & \\ & & \sigma_{r_0} \end{bmatrix} \begin{bmatrix} - & v_{q+1}^H & - \\ & \vdots & \\ - & v_n^H & - \end{bmatrix}$$

we found the SVD of $A - A_q$ with SV

$$\sigma_{q+1} \ge \cdots \ge \sigma_r$$

Because $\|Z\|_2 = \sigma_{max}(Z)$

$$\|A - A_q\|_2 = \sigma_{q+1}$$

Suppose there is B with $\operatorname{rank}(B) \le q$ and

$$\|A - B\|_2 < \|A - A_q\|_2 = \sigma_{q+1} \qquad (*)$$

Because $B \in \mathbb{C}^{n \times n}$ has rank $\le q$, there is an $\underline{n-q}$ dimensional null space $W \subseteq \mathbb{C}^n$ such that

$$w \in W \Rightarrow Bw = 0$$

for any $w \in W$ have

$$Aw = (A - B)w$$

and

$$\|A\underline{w}\|_2 = \|(A-B)w\|_2 \le \|A - B\|_2 \|w\|_2$$

$$\underline{<} \ \underline{\sigma_{q+1}} \ \|w\|_2$$

Now take v_i, then

$$Av_i = U \Sigma V^H v_i =$$

$$= U \Sigma \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = U \begin{bmatrix} 0 \\ \vdots \\ \sigma_i \\ \vdots \\ 0 \end{bmatrix} = u_i \sigma_i$$

The vectors $v_1, \dots, v_{q+1}$ span a $\underline{q+1}$ dim space $Z \subset \mathbb{C}^n$

$$\forall z \in Z: \quad z = \sum_{i=1}^{q+1} \alpha_i v_i \quad , \quad \|z\|_2 = \sqrt{\sum_{i=1}^{q+1} |\alpha_i|^2}$$

$$Az = \sum_{i=1}^{q+1} \sigma_i u_i \alpha_i$$

with

$$\|Az\|_2 = \sqrt{\sum_{i=1}^{q+1} \sigma_i^2 \alpha_i^2 \underbrace{\|u_i\|^2}_{=1}} \geq \sigma_{q+1} \sqrt{\sum \alpha_i^2} = \sigma_{q+1} \|z\|_2$$

However, the subspace Z has dim $q+1$ and the subspace W has dim $n-q$, thus there must be a non-zero intersection, which contradicts that $\forall w \in W: \|Aw\| < \sigma_{q+1} \|w\|_2$

SVD for low-rank approximation

Consider the SVD $A = U\Sigma V^H$, then

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^H$$

Let us truncate the sum after $1 \leq q \leq r$ terms and define

$$A_q = \sum_{j=1}^{q} \sigma_j u_j v_j^H.$$

Then, A_q is a best rank q approximation of A in the $\|\cdot\|_2$ norm

$$\|A - A_q\|_2 = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ \text{rank}(B) \leq q}} \|A - B\|_2 = \sigma_{q+1}$$

The error is the first left out singular value σ_{q+1}! $\rightsquigarrow$ proof

Furthermore, in the Frobenius norm $\|\cdot\|_F$, for any $0 \leq q \leq r$, the matrix A_q from the previous slide also satisfies

$$\|A - A_q\|_F = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ \mathrm{rank}(B) \leq q}} \|A - B\|_F = \sqrt{\sigma_{q+1}^2 + \cdots + \sigma_r^2}$$

SVD and pseudo inverse

The (Moore-Penrose) pseudo inverse of an $m \times n$ matrix that is regular and square is $A^+ = A^{-1}$.

Otherwise, the pseudo inverse is given by

$$A^+ = V\Sigma^+ U^H\,,$$

where

$$\Sigma^+ = \mathrm{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$$

Thus, we can solve least-squares problems $\min_x \|b - Ax\|_2$ by taking the SVD of A, computing A^+, and setting $x = A^+ b$ for the minimal norm solution w.r.t. $\|\cdot\|_2$

The approach we discussed via the QR decomposition is cheaper but the approach via the SVD is sometimes preferred because it allows to easily regularize the problem by truncating small singular values.

How to compute the SVD?

†

How to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

†

How to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea?

†

How **not** to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea? $\rightsquigarrow$ as we have seen before, it is typically dangerous[†] from a stability perspective to compute something of the matrix A via the matrix $A^H A$ (think of least-squares regression) $\rightsquigarrow$ board

[†] https://nhigham.com/2022/10/11/seven-sins-of-numerical-linear-algebra/

Motivate why computing SV via EV is not good idea

↳ perturbation of EV of Hermitian $A^H A$

condition number

$$|\lambda_k(A^H A + \delta B) - \lambda_k(A^H A)| \leq \|\delta B\|_2$$

↳ similar perturbation bound hold for SV:

$$|\sigma_k(A + \delta A) - \sigma_k(A)| \leq \|\delta A\|$$

Backward stable algo for SV computes $\tilde{\sigma}_k$ that satisfies

$$\tilde{\sigma}_k = \sigma_k(A + \delta A) \qquad \frac{\|\delta A\|}{\|A\|} \in \mathcal{O}(\varepsilon)$$

implies

$$|\tilde{\sigma}_k - \sigma_k| \leq \|\delta A\| \quad \text{with} \quad \|\delta A\| \in \mathcal{O}(\varepsilon \|A\|)$$

Now let us compute $\lambda_k(A^H A)$. With backward stable algo

$$\tilde{\lambda}_k = \lambda_k(A^H A + \delta B), \qquad \frac{\|\delta B\|}{\|A^H A\|} \in \mathcal{O}(\varepsilon)$$

$$|\tilde{\lambda}_k - \lambda_k| \leq \|\delta B\| \in \mathcal{O}(\varepsilon \|A^H A\|) = \mathcal{O}(\varepsilon \|A\|^2)$$

Compute $\tilde{\sigma}_k = \sqrt{\tilde{\lambda}_k}$ and obtain

$$|\tilde{\sigma}_k - \sigma_k| = |\sqrt{\tilde{\lambda}_k} - \sqrt{\lambda_k}| \leq \frac{|\tilde{\lambda}_k - \lambda_k|}{\underbrace{\sqrt{\tilde{\lambda}_k}}_{\geq 0} + \underbrace{\sqrt{\lambda_k}}_{\sigma_k}}$$

$$\leq \frac{|\tilde{\lambda}_k - \lambda_k|}{\sigma_k}$$

$|\tilde{\sigma}_k - \sigma_k|$ behaves as $\mathcal{O}\left(\varepsilon \frac{\|A\|^2}{\sigma_k}\right)$

worse by factor $\frac{\|A\|_2}{\sigma_k}$ than previous result

$\Rightarrow$ „no" problem for large σ_k with $\sigma_k \approx \|A\|_2$
but big problem for $\sigma_k \ll \|A\|_2$

How we actually compute the SVD

For the sake of the argument, assume that A is a square $m \times m$ (following is applicable to rectangular matrices too) $\rightsquigarrow$ board

Instead of $A^H A$, let us use

$$H = \begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

larger matrix
$\rightarrow$ ignore for now

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Singular value decomposition

Today

- Singular value decomposition
- Iterative methods for solving linear systems

Announcements

- Homework 3 posted, is due Mon, Oct 21 before class

Recap

Let $A \in \mathbb{C}^{m \times n}$. A singular value decomposition of A is a factorization

$$A = U \Sigma V^H ,$$

where

$$U \in \mathbb{C}^{m \times m} \text{ is unitary} \tag{12}$$

$$V \in \mathbb{C}^{n \times n} \text{ is unitary} \tag{13}$$

$$\Sigma \in \mathbb{R}^{m \times n} \text{ is diagonal.} \tag{14}$$

Additionally, the diagonal entries σ_j of Σ are non-negative and in non-decreasing order so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p \in \min(m, n)$.

- The diagonal matrix Σ is real and has the same shape as A even when A is not square
- The matrices U and V are always square

Recap

Full SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times m}\underbrace{\boldsymbol{\Sigma}}_{m\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Reduced SVD

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times n}\underbrace{\boldsymbol{\Sigma}}_{n\times n}\underbrace{\boldsymbol{V}^H}_{n\times n}$$

Rank-revealing SVD of a rank r matrix with dimension $m \times n$

$$\underbrace{\boldsymbol{A}}_{m\times n} = \underbrace{\boldsymbol{U}}_{m\times r}\underbrace{\boldsymbol{\Sigma}}_{r\times r}\underbrace{\boldsymbol{V}^H}_{r\times n}$$

Recap: The SVD and matrix properties

In the following:

- The matrix A is of dimension $m \times n$
- $p = \min(m, n)$
- $r \leq p$ is the number of non-zero singular values of A

We now list how the SVD is related to fundamental properties of the matrix A

- The rank of A is r, the number of non-zero singular values.
- The range (column span) of A is $\text{span}(u_1, \ldots, u_r)$, the kernel is $\text{span}(v_{r+1}, \ldots, v_n)$
- $\|A\|_2 = \sigma_1$ and $\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$
- For square A, $|\det(A)| = \prod_{i=1}^{m} \sigma_i$
- The non-zero singular values of A are the square roots of the non-zero eigenvalues of $A^H A$ and AA^H

Recap: SVD for low-rank approximation

Consider the SVD $A = U\Sigma V^H$, then

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^H$$

Let us truncate the sum after $1 \leq q \leq r$ terms and define

$$A_q = \sum_{j=1}^{q} \sigma_j u_j v_j^H.$$

Then, A_q is a best rank q approximation of A in the $\|\cdot\|_2$ norm

$$\|A - A_q\|_2 = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ \text{rank}(B) \leq q}} \|A - B\|_2 = \sigma_{q+1}$$

The error is the first left out singular value σ_{q+1}!

Recap

Furthermore, in the Frobenius norm $\|\cdot\|_F$, for any $0 \leq q \leq r$, the matrix A_q from the previous slide also satisfies

$$\|A - A_q\|_F = \inf_{\substack{B \in \mathbb{C}^{m \times n} \\ \operatorname{rank}(B) \leq q}} \|A - B\|_F = \sqrt{\sigma_{q+1}^2 + \cdots + \sigma_r^2}$$

Recap: SVD and pseudo inverse

The (Moore-Penrose) pseudo inverse of an $m \times n$ matrix that is regular and square is $A^+ = A^{-1}$.

Otherwise, the pseudo inverse is given by

$$A^+ = V\Sigma^+ U^H\,,$$

where

$$\Sigma^+ = \mathrm{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$$

Thus, we can solve least-squares problems $\min_x \|b - Ax\|_2$ by taking the SVD of A, computing A^+, and setting $x = A^+ b$ for the minimal norm solution w.r.t. $\|\cdot\|_2$

The approach we discussed via the QR decomposition is cheaper but the approach via the SVD is sometimes preferred because it allows to easily regularize the problem by truncating small singular values.

How to compute the SVD?

†

How to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

†

How to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea?

†

How **not** to compute the SVD?

The SVD of an $m \times n$ ($m \geq n$) matrix A is related to the eigenvalue decomposition of $A^H A$

$$A^H A = V \Sigma^H \Sigma V^H ,$$

Since we know how to numerically compute the eigendecomposition, we could compute the SVD of A as follows

1. Form $A^H A$
2. Compute the eigendecomposition $A^H A = V \Lambda V^H$ (Notice that $Z = A^H A$ is normal because $Z^H Z = Z Z^H$)
3. Let Σ be the $m \times n$ non-negative diagonal square root of Λ
4. Solve the system $U\Sigma = AV$ for unitary U

Is this a good idea? $\rightsquigarrow$ as we have seen before, it is typically dangerous[†] from a stability perspective to compute something of the matrix A via the matrix $A^H A$ (think of least-squares regression) $\rightsquigarrow$ board

[†] https://nhigham.com/2022/10/11/seven-sins-of-numerical-linear-algebra/

How we actually compute the SVD

For the sake of the argument, assume that A is a square $m \times m$ (following is applicable to rectangular matrices too) $\rightsquigarrow$ board

Instead of $A^H A$, let us use

$$H = \begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

Because $A = U\Sigma V^H$ have

$$AV = U\Sigma .$$

$$A^H U = V\Sigma^H U^H U = V\Sigma^H = V\Sigma .$$

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix} \underbrace{\begin{bmatrix} V & V \\ U & -U \end{bmatrix}}_{X} = \begin{bmatrix} A^H U & -A^H U \\ AV & AV \end{bmatrix}$$

$$= \begin{bmatrix} V\Sigma & -V\Sigma \\ U\Sigma & U\Sigma \end{bmatrix} = \underbrace{\begin{bmatrix} V & V \\ U & -U \end{bmatrix}} \underbrace{\begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}}_{\text{diagonal}}$$

$$\begin{bmatrix} V & V \\ U & -U \end{bmatrix}^H \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} 2I & 0 \\ 0 & 2I \end{bmatrix}$$

Would like to find the eigendecomposition of

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

Algorithm that uses this idea but avoids assembling H matrix

How we actually compute the SVD

For the sake of the argument, assume that A is a square $m \times m$ (following is applicable to rectangular matrices too) $\rightsquigarrow$ board

Consider the $2m \times 2m$ Hermitian matrix

$$H = \begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix}$$

Since $A = U\Sigma V^H$, we have

$$AV = U\Sigma \text{ and } A^H U = V\Sigma^H = V\Sigma \quad \text{(recall that singular values are real)}$$

which we write in matrix form as

$$\begin{bmatrix} 0 & A^H \\ A & 0 \end{bmatrix} \begin{bmatrix} V & V \\ U & -U \end{bmatrix} = \begin{bmatrix} V & V \\ U & -U \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{bmatrix}$$

which is the eigendecomposition of H $\rightsquigarrow$ avoids using $A^H A$ and AA^H and is stable

Computing the SVD is typically a two-phase procedure:

First, reduce the matrix A to bidiagonal form, then diagonalize the bidiagonal matrix

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \rightarrow \begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & & * \\ & & & \\ & & & \end{bmatrix} \rightarrow \begin{bmatrix} * & & & \\ & * & & \\ & & * & \\ & & & * \\ & & & \\ & & & \end{bmatrix}$$

- Phase 1 involves a finite number of operations that scale as $\mathcal{O}(mn^2)$
- Phase 2 (recall eigenvalue problems) is iterative but converges very quickly; in practice achieves convergence in $\mathcal{O}(n)$ to machine precision
- Thus, state-of-the-art computation of the SVD has costs that scale as $\mathcal{O}(mn^2)$

This is the celebrated Golub-Kahan approach from the 1960s

How can we bidiagonalize a matrix? Recall that we already know how to triangualize a matrix via (unitary) Householder reflection

$$\underset{A}{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}} \rightarrow \underset{Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}} \rightarrow \underset{Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & * & * \\ & & * & * \\ & & * & * \end{bmatrix}} \rightarrow \underset{Q_3 Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & & * \\ & & & * \end{bmatrix}} \rightarrow \underset{Q_4 Q_3 Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & & \\ & & & \end{bmatrix}}$$

How can we bidiagonalize a matrix? Recall that we already know how to triangualize a matrix via (unitary) Householder reflection

$$\underset{A}{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}} \to \underset{Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}} \to \underset{Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & * & * \\ & & * & * \\ & & * & * \end{bmatrix}} \to \underset{Q_3 Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & & * \\ & & & * \end{bmatrix}} \to \underset{Q_4 Q_3 Q_2 Q_1 A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & & * & * \\ & & & * \\ & & & \\ & & & \end{bmatrix}}$$

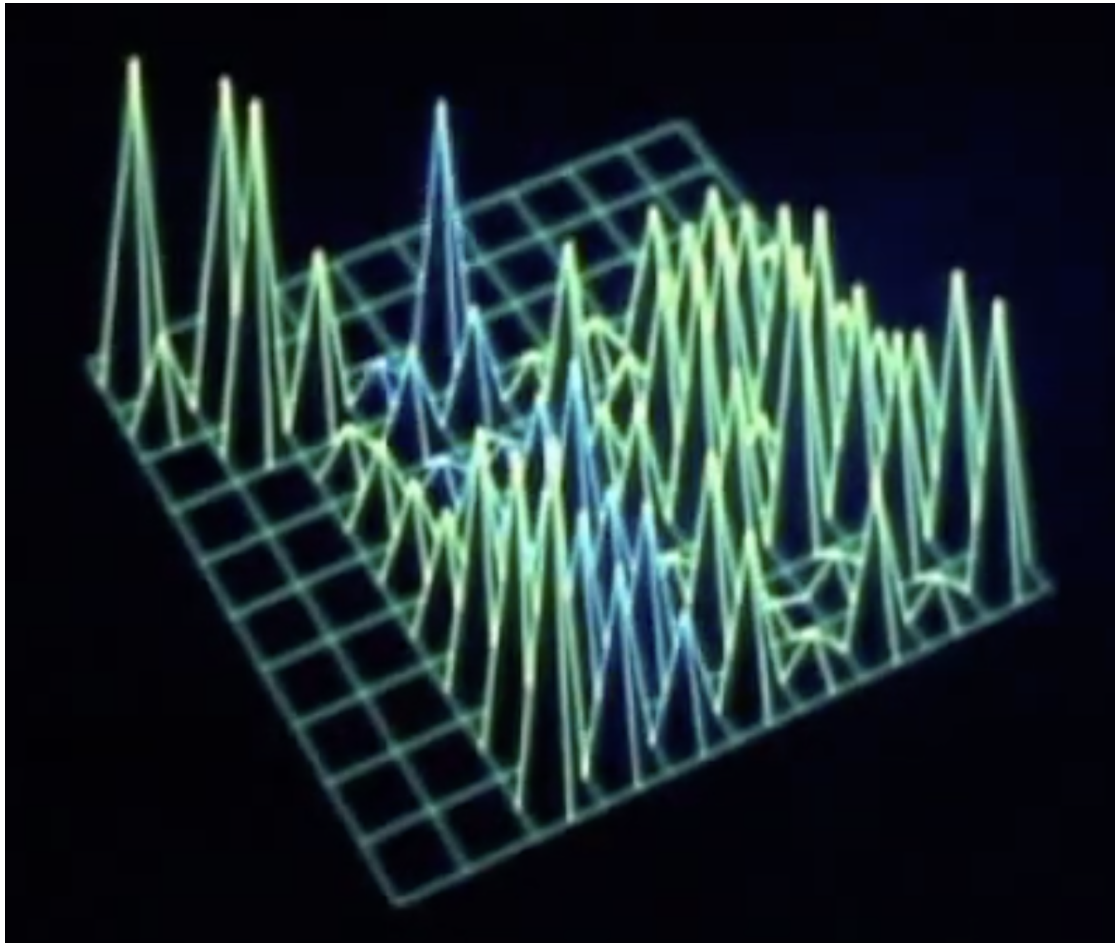
Now, we apply interleaved Householder reflection from left and right

$$\underset{A}{\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}} \to \underset{U_1^H A}{\begin{bmatrix} * & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}} \to \underset{U_1^H A V_1}{\begin{bmatrix} * & * & & \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{bmatrix}} \to \underset{U_2^H U_1^H A V_1}{\begin{bmatrix} * & * & & \\ & * & * & * \\ & & * & * \\ & & * & * \\ & & * & * \\ & & * & * \end{bmatrix}} \to \underset{U_2^H U_1^H A V_1 V_2}{\begin{bmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & * & * \\ & & * & * \\ & & * & * \end{bmatrix}} \to \dots$$

At the end, n reflectors are applied from the left and $n-2$ from the right

A variant of the QR algorithm is then applied to the bidiagonal matrix (or other eigendecomposition algorithms) $\rightsquigarrow$ details in Golub et al., Matrix Computations.

Visualizing SVD computation



Video: http://youtu.be/R9UoFyqJca8

SVD saves the universe

The first Star Trek movie came out in 1979. The producers had asked Los Alamos for computer graphics to run on the displays on the bridge of the Enterprise:

[Figure: Paramount Pictures]

Link: https://blogs.mathworks.com/cleve/2012/12/10/1976-matrix-singular-value-decomposition-film/

Matlab

```
>> X = randn(10, 4);
>> [U, S, V] = svd(X); % full SVD
>> [Ur, Sr, Vr] = svd(X, 0); % reduced (economic) SVD
>> size(S)
>> size(Sr)
ans =
    10     4
ans =
     4     4
```

In most cases, we want the reduced SVD and then we should explicitly compute it:

```
>> X = randn(10000, 50);
>> tic; [U, S, V] = svd(X); toc
Elapsed time is 6.746243 seconds.
>> tic; [U, S, V] = svd(X, 0); toc
Elapsed time is 0.064572 seconds.
```

For very large and sparse matrices, or if the matrix is unavailable and we only have a procedure that returns the matrix-vector product, we can iteratively compute the first few singular vectors via `svds`

```
>> X = gallery('poisson', 100);
>> whos X
Name          Size                    Bytes  Class
    Attributes

  X          10000x10000              1033608  double
      sparse

>> tic; [U, S, V] = svds(X, 1); toc

Elapsed time is 0.307015 seconds.
```

SVD is great because its computation is (very) stable $\rightsquigarrow$ building block for computing many basic linear algebra quantities

type 'edit rank.m' in Matlab

```
function r = rank(A,tol)
%RANK   Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%
%   RANK(A,TOL) is the number of singular values of A
%   that are larger than TOL. By default, TOL = max(size(A)) * eps(norm(A)).
%
%   Class support for input A:
%      float: double, single

%   Copyright 1984-2015 The MathWorks, Inc.

s = svd(A);
if nargin==1
    tol = max(size(A)) * eps(max(s));
end
r = sum(s > tol);
```

type 'edit pinv.m' in Matlab

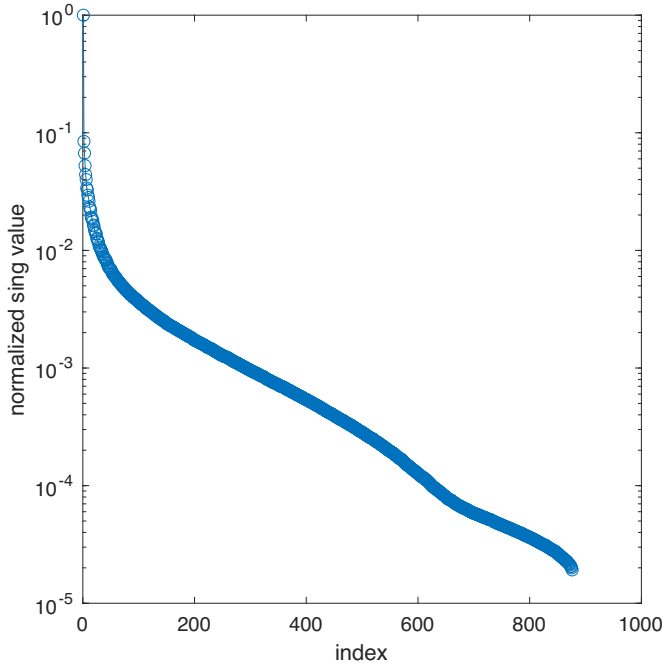
```
function X = pinv(A,tol)
%PINV   Pseudoinverse.
%   X = PINV(A) produces a matrix X of the same dimensions
%   as A' so that A*X*A = A, X*A*X = X and A*X and X*A
%   are Hermitian. The computation is based on SVD(A) and any
%   singular values less than a tolerance are treated as zero.
%
%   PINV(A,TOL) treats all singular values of A that are less than TOL as
%   zero. By default, TOL = max(size(A)) * eps(norm(A)).
%
%   Class support for input A:
%      float: double, single
%
%   See also RANK.

%   Copyright 1984-2015 The MathWorks, Inc.

[U,S,V] = svd(A,'econ');
s = diag(S);
if nargin < 2
    tol = max(size(A)) * eps(norm(s,inf));
end
r1 = sum(s > tol)+1;
V(:,r1:end) = [];
U(:,r1:end) = [];
s(r1:end) = [];
s = 1./s(:);
X = (V.*s.')*U';
```

type 'edit orth.m' in Matlab

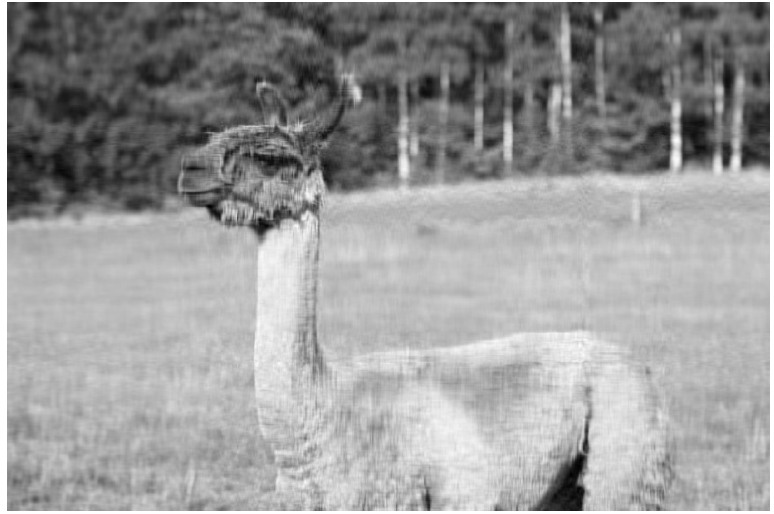
```
function Q = orth(A)
%ORTH   Orthogonalization.
%   Q = ORTH(A) is an orthonormal basis for the range of A.
%   That is, Q'*Q = I, the columns of Q span the same space as
%   the columns of A, and the number of columns of Q is the
%   rank of A.
%
%   Class support for input A:
%      float: double, single
%
%   See also SVD, RANK, NULL.

%   Copyright 1984-2015 The MathWorks, Inc.

[Q,S] = svd(A,'econ'); %S is always square.
s = diag(S);
tol = max(size(A)) * eps(max(s));
r = sum(s > tol);
Q(:, r+1:end) = [];
```

Application of SVD for image compression

```
>> A = rgb2gray(imread('llama.jpg'));
>> figure; imshow(A);
>> [U, S, V] = svd(double(A));
>> figure; semilogy(diag(S)/S(1, 1), '-o');
>> xlabel('index');ylabel('normalized sing value');

>> r = 50;
>> Aapprx = U(:, 1:r)*S(1:r, 1:r)*V(:, 1:r)';
>> figure; imshow(uint8(Aapprx));
```

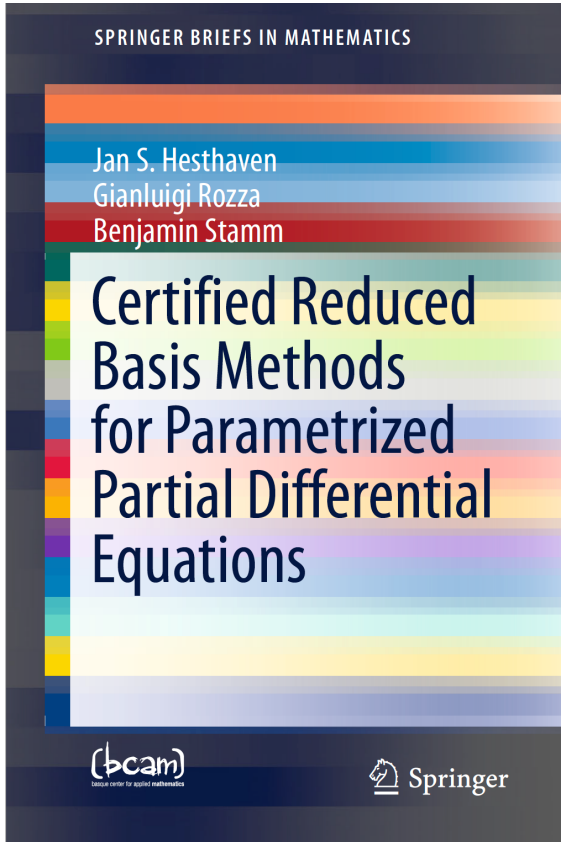


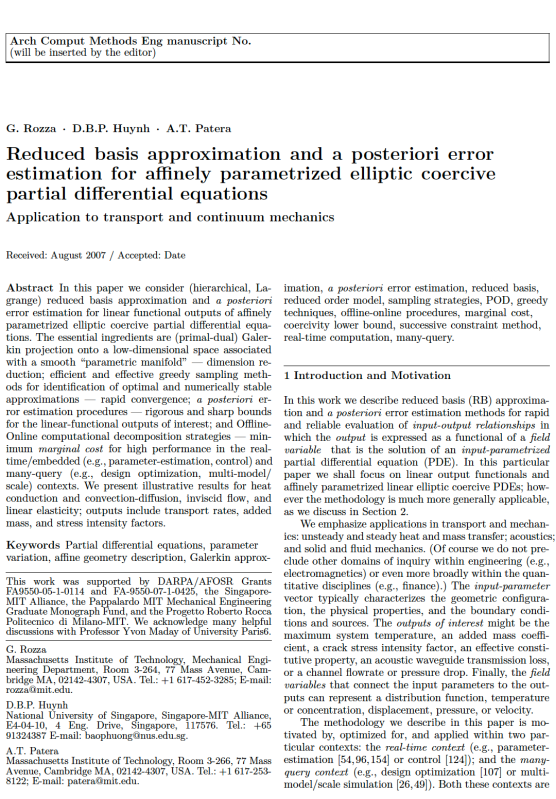
Original picture and singular values



Original picture (left) and reduced picture (right)

Outlook: Model reduction and latent dynamics





Arch Comput Methods Eng manuscript No.
(will be inserted by the editor)

G. Rozza · D.B.P. Huynh · A.T. Patera

Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations

Application to transport and continuum mechanics

Received: August 2007 / Accepted: Date

Abstract In this paper we consider (hierarchical, Lagrange) reduced basis approximation and *a posteriori* error estimation for linear functional outputs of affinely parametrized elliptic coercive partial differential equations. The essential ingredients are (primal-dual) Galerkin projection onto a low-dimensional space associated with a smooth "parametric manifold" — dimension reduction; efficient and effective greedy sampling methods for identification of optimal and numerically stable approximations — rapid convergence; *a posteriori* error estimation procedures — rigorous and sharp bounds for the linear-functional outputs of interest; and Offline-Online computational decomposition strategies — minimum *marginal cost* for high performance in the real-time/embedded (e.g., parameter-estimation, control) and many-query (e.g., design optimization, multi-model/scale) contexts. We present illustrative results for heat conduction and convection-diffusion, inviscid flow, and linear elasticity; outputs include transport rates, added mass, and stress intensity factors.

Keywords Partial differential equations, parameter variation, affine geometry description, Galerkin approximation, *a posteriori* error estimation, reduced basis, reduced order model, sampling strategies, POD, greedy techniques, offline-online procedures, marginal cost, coercivity lower bound, successive constraint method, real-time computation, many-query.

This work was supported by DARPA/AFOSR Grants FA9550-05-1-0114 and FA-9550-07-1-0425, the Singapore-MIT Alliance, the Pappalardo MIT Mechanical Engineering Graduate Monograph Fund, and the Progetto Roberto Rocca Politecnico di Milano-MIT. We acknowledge many helpful discussions with Professor Yvon Maday of University Paris6.

G. Rozza
Massachusetts Institute of Technology, Mechanical Engineering Department, Room 3-264, 77 Mass Avenue, Cambridge MA, 02142-4307, USA. Tel.: +1 617-452-3285; E-mail: rozza@mit.edu.

D.B.P. Huynh
National University of Singapore, Singapore-MIT Alliance, E4-04-10, 4 Eng. Drive, Singapore, 117576. Tel.: +65 91324387 E-mail: baophuong@nus.edu.sg.

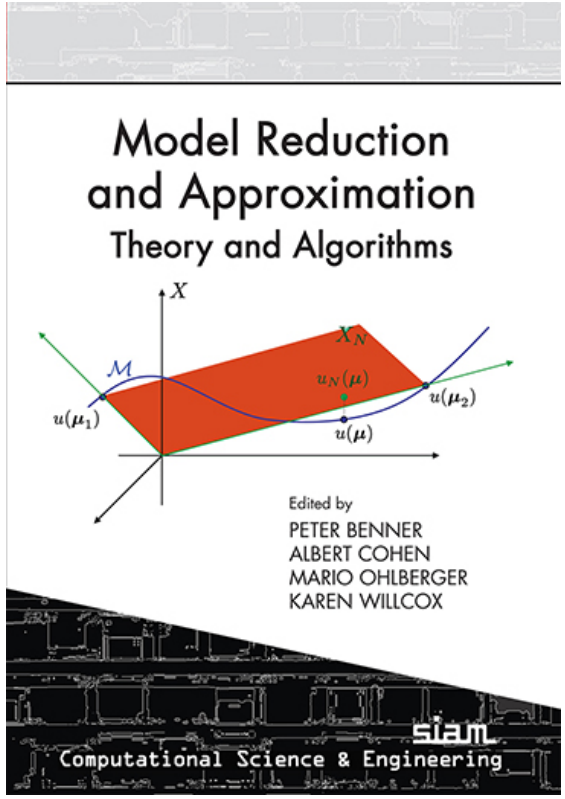
A.T. Patera
Massachusetts Institute of Technology, Room 3-266, 77 Mass Avenue, Cambridge MA, 02142-4307, USA. Tel.: +1 617-253-8122; E-mail: patera@mit.edu.

1 Introduction and Motivation

In this work we describe reduced basis (RB) approximation and *a posteriori* error estimation methods for rapid and reliable evaluation of *input-output relationships* in which the *output* is expressed as a functional of a *field variable* that is the solution of an *input-parametrized* partial differential equation (PDE). In this particular paper we shall focus on linear output functionals and affinely parametrized linear elliptic coercive PDEs; however the methodology is much more generally applicable, as we discuss in Section 2.

We emphasize applications in transport and mechanics: unsteady and steady heat and mass transfer; acoustics; and solid and fluid mechanics. (Of course we do not preclude other domains of inquiry within engineering (e.g., electromagnetics) or even more broadly within the quantitative disciplines (e.g., finance).) The *input-parameter* vector typically characterizes the geometric configuration, the physical properties, and the boundary conditions and sources. The *outputs of interest* might be the maximum system temperature, an added mass coefficient, a crack stress intensity factor, an effective constitutive property, an acoustic waveguide transmission loss, or a channel flowrate or pressure drop. Finally, the *field variables* that connect the input parameters to the outputs can represent a distribution function, temperature or concentration, displacement, pressure, or velocity.

The methodology we describe in this paper is motivated by, optimized for, and applied within two particular contexts: the *real-time context* (e.g., parameter-estimation [54,96,154] or control [124]); and the *many-query context* (e.g., design optimization [107] or multi-model/scale simulation [26,49]). Both these contexts are



Iterative methods for solving systems of linear equations

Why iterative methods for linear systems?

- Costs of solving a linear system with direct method are $\mathcal{O}(n^3)$. This is too much! If n gets large, then n^3 is huge and costs become intractable.

- History of matrix computations over the years (according to Trefethen & Bau)
  1950: $m = 20$
  1965: $m = 200$
  1980: $m = 2000$
  1995: $m = 20000$
  This is an increase of a factor 10^3. However, computing power (FLOP/sec) increased by about 10^9. Notice that $(10^3)^3 = 10^9$, which reflects the $\mathcal{O}(n^3)$ bottleneck

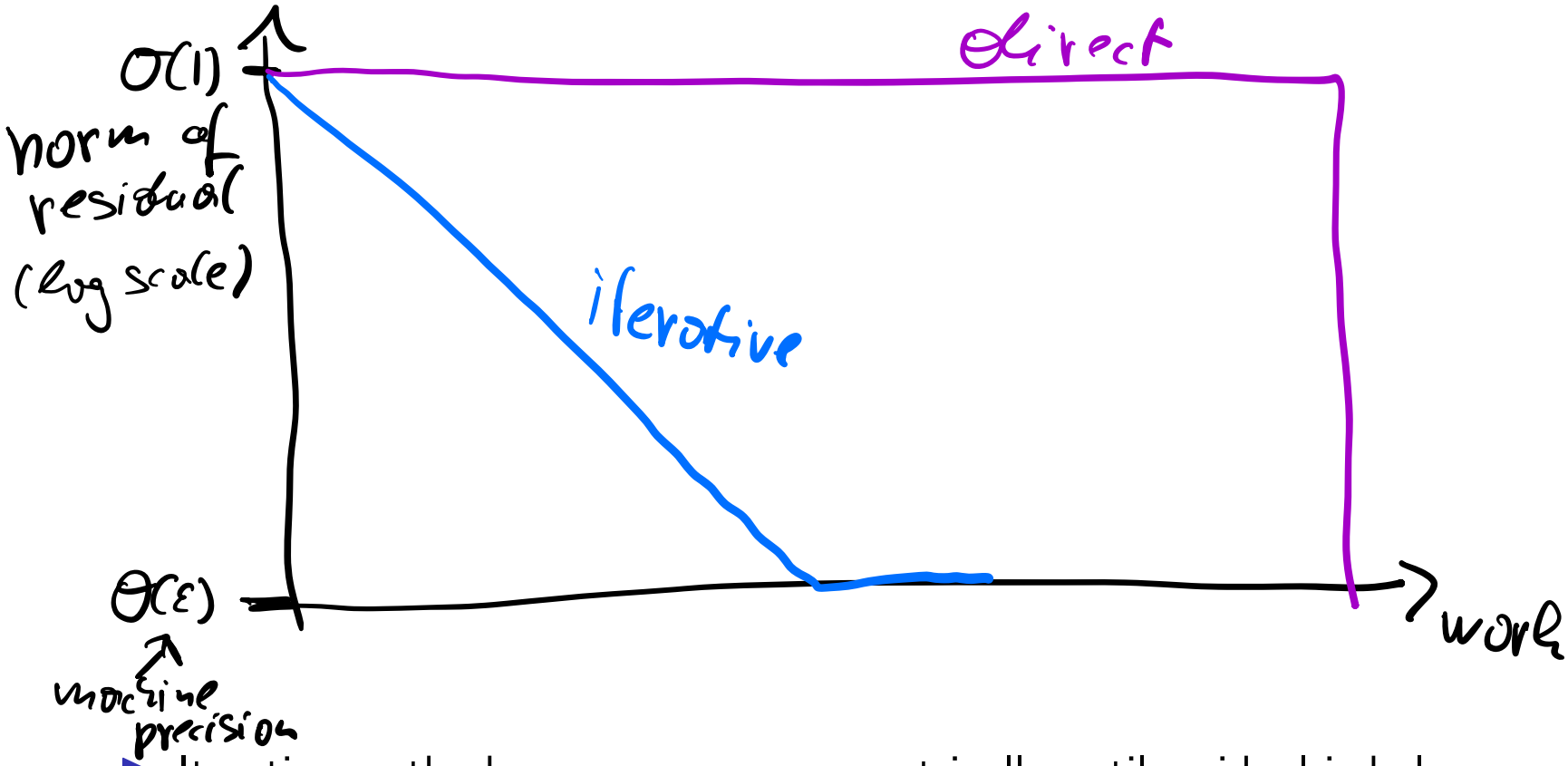
- (Side remark: There are direct methods that beat the complexity $\mathcal{O}(n^3)$ (think of Strassen's algorithm); however,

Why iterative methods for linear systems?

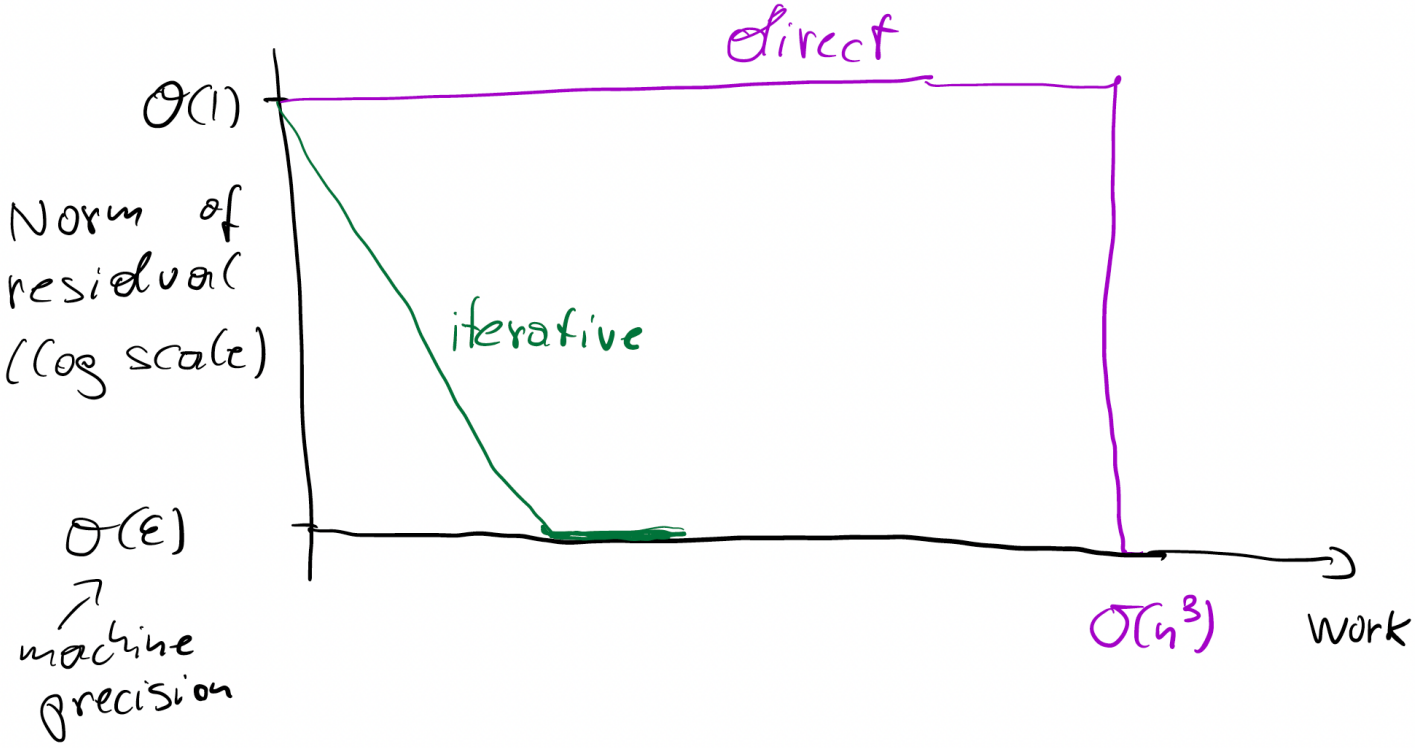
- Costs of solving a linear system with direct method are $\mathcal{O}(n^3)$. This is too much! If n gets large, then n^3 is huge and costs become intractable.

- History of matrix computations over the years (according to Trefethen & Bau)
  1950: $m = 20$
  1965: $m = 200$
  1980: $m = 2000$
  1995: $m = 20000$
  This is an increase of a factor 10^3. However, computing power (FLOP/sec) increased by about 10^9. Notice that $(10^3)^3 = 10^9$, which reflects the $\mathcal{O}(n^3)$ bottleneck

- (Side remark: There are direct methods that beat the complexity $\mathcal{O}(n^3)$ (think of Strassen's algorithm); however, the numerical stability of these algorithms is not well understood and the constants hidden in the complexity results are huge so that we "never" see the improved rate in practice.)



- Iterative methods *can* converge geometrically until residual is below machine precision
- Direct methods make no progress at all until $\mathcal{O}(n^3)$ work is done, and then lead to residual on the order of machine precision



- Iterative methods *can* converge geometrically until residual is below machine precision
- Direct methods make no progress at all until $\mathcal{O}(n^3)$ work is done, and then lead to residual on the order of machine precision

Why iterative methods for linear systems? (cont'd)

- Classical direct methods (e.g., Gauss elimination) follow the pattern of taking $\mathcal{O}(n)$ steps and each step costs $\mathcal{O}(n^2)$ but don't exploit properties of the matrix
- Iterative methods reduce the number of steps and the costs of each step, depending on properties of the problem at hand (e.g., spectral properties of A in $Ax = b$)
- The ideal iterative method requires $\mathcal{O}(1)$ steps and $\mathcal{O}(n)$ costs per step to reach machine precision (think of multigrid and pre-conditioned CG)
- Iterative methods, even in the absence of rounding errors, do not deliver the exact answer (recall that LU, QR give us exact answer in finite number of steps if we are in exact arithmetic).
- However, we only can be as accurate as machine precision anyway if we do calculations on a computer. Furthermore, often matrices are stemming from discretizations of PDEs and then we need the solution only up to the discretization error.

Iterative solution of linear systems

Target problems: very large ($n = 10^5, 10^6, \ldots$), A is usually sparse and has specific properties.

To solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

we construct a sequence

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

of iterates that converges fast to the solution $\boldsymbol{x}$, where $\boldsymbol{x}_{k+1}$ can be cheaply computed from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ (e.g., one matrix-vector multiplication).

Thought experiment: If we can compute one iteration with cost $\mathcal{O}(n)$ (e.g., one matrix-vector multiplication with a sparse matrix) and need a constant $\mathcal{O}(1)$ number of iterations to reach desired precision, then we solve $Ax = b$ with costs $\mathcal{O}(n)$. Intuitively, we cannot do better than that because we solve for n quantities and thus need to touch each at least once.

Prototype of iterative method

Let's start by trying to write $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ more generally as

$$\boldsymbol{x} = f(\boldsymbol{x})$$

For example set $f(\boldsymbol{x}) = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}$ to obtain

$$\boldsymbol{x} = f(\boldsymbol{x}) = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x} + \boldsymbol{b}$$

We can now try to solve this via a fixed-point iteration

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k)$$

which is in our case

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

Instead of picking a specific f, let's look at the prototype

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

where $\boldsymbol{G}$ is an iteration matrix somehow related to $\boldsymbol{A}$ and $\boldsymbol{c}$ is related to $\boldsymbol{b}$.

We must have the actual solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ as a unique fixed point of the iteration:

$$\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c}$$

And we need that the sequence $(\boldsymbol{x}_k)$ converges

Theorem: The fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with an invertible G converges for each starting point $\boldsymbol{x}_o$ if and only if

$$\rho(G) < 1,$$

where $\rho(G) = \max_j |\lambda_j|$ is the largest magnitude of all eigenvalue of G (i.e., the spectral radius).

$\rightsquigarrow$ proof

Symmetric G, real

There is an orthogonal Q

$$QGQ^T = D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}$$

with eigenvalues

Because $|\lambda_i| \leq \rho(G) < 1$ for all $i = 1, \dots, n$ we have

$$\lim_{k \to \infty} \underbrace{\begin{bmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_n^k \end{bmatrix}}_{D^k} = 0$$

Thus

$$\lim_{k \to \infty} G^k = \lim_{k \to \infty} Q D^k Q^T = 0$$

$$\begin{aligned} \|x_{k+m} - x_k\| &= \|G x_{k+m-1} + c - G x_{k-1} - c\| \\ &= \|G(x_{k+m-1} - x_{k-1})\| \\ &= \dots = \\ &= \|G^k (x_m - x_0)\| \end{aligned}$$

$$\lim_{k \to \infty} \|x_{k+m} - x_k\| = 0 \quad , \quad m \geq 1$$

$\to$ Cauchy sequence

We often ask for $\|G\| < 1$, which implies

$$\rho(G) < 1$$

because $\rho(G) \leq \|G\|$ for $\|\cdot\|$ induced by a vector norm.

$$\|x_k - x\| \leq \underline{\underline{\|G\|}}^k \|x_0 - x\|$$

estimates how quickly error decays

Theorem: The fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with an invertible G converges for each starting point $\boldsymbol{x}_o$ if and only if

$$\rho(G) < 1,$$

where $\rho(G) = \max_j |\lambda_j|$ is the largest magnitude of all eigenvalue of G (i.e., the spectral radius).

$\rightsquigarrow$ proof

In particular, for any induced matrix norm $\|\cdot\|$ we have $\rho(G) \leq \|G\|$ and thus $\|G\|$ is a sufficient criterion for convergence. We then obtain

$$\|x_k - x\| \leq \|G\|^k \|x_0 - x\|$$

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration with $\boldsymbol{G} = (I - Q^{-1}A)$ and $c = Q^{-1}b$

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is stationary point

$$\boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x} - \boldsymbol{Q}^{-1}\boldsymbol{b} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x}$$

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (recall: $\boldsymbol{Q}$ needs to be invertible)?

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (recall: $\boldsymbol{Q}$ needs to be invertible)?

Choose $Q = A$, then our iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}\,, \qquad \boldsymbol{G} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}\,, \quad \boldsymbol{c} = \boldsymbol{Q}^{-1}\boldsymbol{b}$$

becomes

$$(I - A^{-1}A)\boldsymbol{x}_k + \underbrace{A^{-1}\boldsymbol{b}}_{\boldsymbol{x}} = \boldsymbol{x}_{k+1}$$

$$\boldsymbol{0} + \boldsymbol{x} = \boldsymbol{x}_{k+1}$$

and we are done in just a single step

$$x_{k+1} = x$$

Thus, if we "know the solution" (in form of having the inverse A^{-1}) then no further work is needed here because we already did all the work when finding A^{-1}

The other extreme is setting $Q = I$, this leads to the Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

We have invested zero costs in finding Q (and Q^{-1}) and so we can expect that $Q = I$ will require high costs in terms of number of iterations to converge in general, if it converges at all

When does Richardson method converge? $\rightsquigarrow$ board

Let A be spd, then

$$\rho(G) = \rho(I - A) = \max\{|1 - \lambda_{max}|, |1 - \lambda_{min}|\}$$

thus need

$$\lambda_{max}(A) < 2$$

$\Rightarrow$ rarely usable, but important building block

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Iterative methods for systems of linear equations

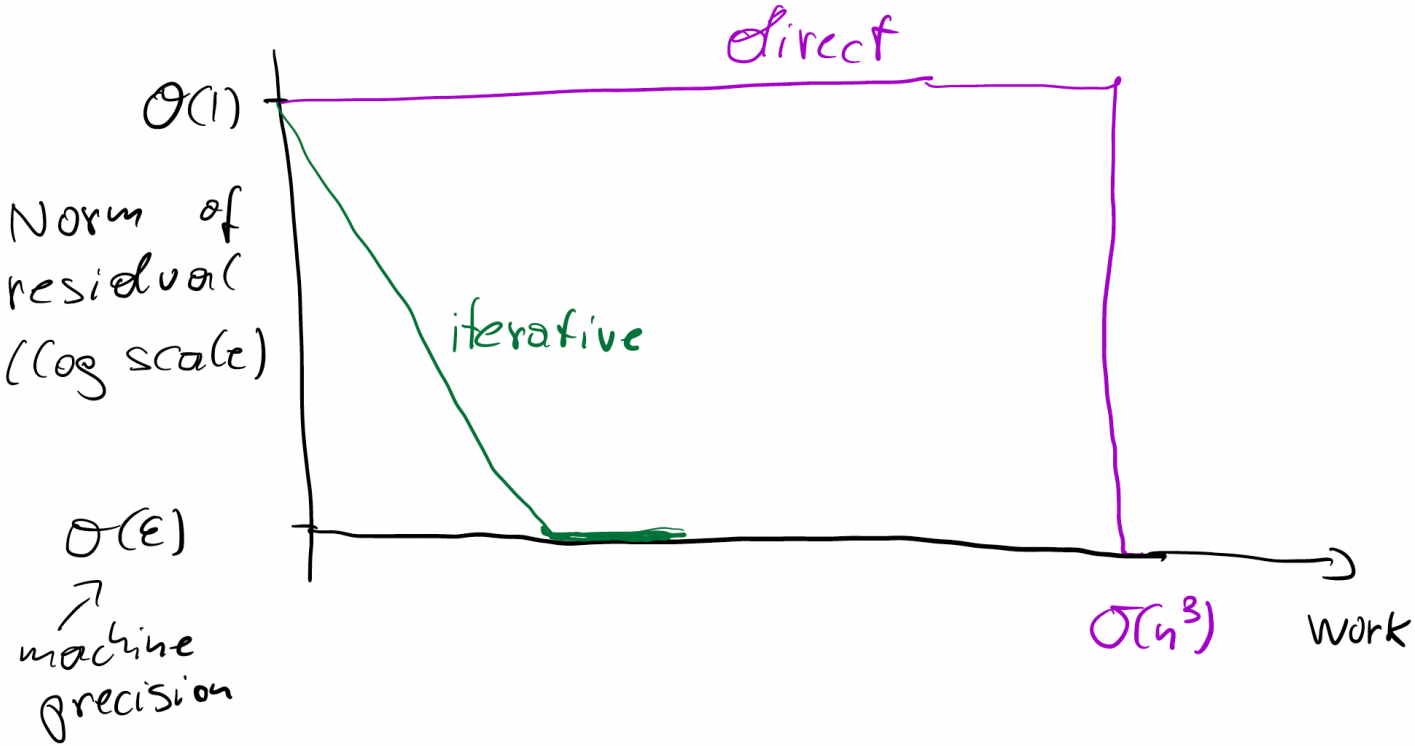
Today

- Iterative methods for systems of linear equations

Announcements

- Homework 4 is due Mon, Nov 4, 2024 before class

Recap



- Iterative methods *can* converge geometrically until residual is below machine precision
- Direct methods make no progress at all until $\mathcal{O}(n^3)$ work is done, and then lead to residual on the order of machine precision

Recap: Iterative solution of linear systems

Target problems: very large ($n = 10^5, 10^6, \ldots$), A is usually sparse and has specific properties.

To solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

we construct a sequence

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

of iterates that converges fast to the solution $\boldsymbol{x}$, where $\boldsymbol{x}_{k+1}$ can be cheaply computed from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ (e.g., one matrix-vector multiplication).

Thought experiment: If we can compute one iteration with cost $\mathcal{O}(n)$ (e.g., one matrix-vector multiplication with a sparse matrix) and need a constant $\mathcal{O}(1)$ number of iterations to reach desired precision, then we solve $Ax = b$ with costs $\mathcal{O}(n)$. Intuitively, we cannot do better than that because we solve for n quantities and thus need to touch each at least once.

Recap

Instead of picking a specific f, let's look at the prototype

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

where $\boldsymbol{G}$ is an iteration matrix somehow related to $\boldsymbol{A}$ and $\boldsymbol{c}$ is related to $\boldsymbol{b}$.

We must have the actual solution $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ as a unique fixed point of the iteration:

$$\boldsymbol{x} = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{c}$$

And we need that the sequence $(\boldsymbol{x}_k)$ converges

Recap

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration with $\boldsymbol{G} = (I - Q^{-1}A)$ and $c = Q^{-1}b$

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is stationary point

$$\boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x} - \boldsymbol{Q}^{-1}\boldsymbol{b} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x}$$

Recap: Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (recall: $\boldsymbol{Q}$ needs to be invertible)?

Choose $Q = A$, then our iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}\,, \qquad \boldsymbol{G} = \boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A}\,, \quad \boldsymbol{c} = \boldsymbol{Q}^{-1}\boldsymbol{b}$$

becomes

$$(I - A^{-1}A)\boldsymbol{x}_k + \underbrace{A^{-1}\boldsymbol{b}}_{\boldsymbol{x}} = \boldsymbol{x}_{k+1}$$

$$\boldsymbol{0} + \boldsymbol{x} = \boldsymbol{x}_{k+1}$$

and we are done in just a single step

$$x_{k+1} = x$$

Thus, if we "know the solution" (in form of having the inverse A^{-1}) then no further work is needed here because we already did all the work when finding A^{-1}

Recap

The other extreme is setting $Q = I$, this leads to the Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

We have invested zero costs in finding Q (and Q^{-1}) and so we can expect that $Q = I$ will require high costs in terms of number of iterations to converge in general, if it converges at all

When does Richardson method converge? $\rightsquigarrow$ if $\lambda_{\max} < 2$, very restrictive

The Richardson method is consistent (solution is a stationary point) but it may not converge or converge very slowly

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

What could we do instead of Richardson method?

The Richardson method is consistent (solution is a stationary point) but it may not converge or converge very slowly

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

What could we do instead of Richardson method?
Let us think of $\boldsymbol{Q}$ as a preconditioner of the Richardson method:

$$\boldsymbol{Q}^{-1} \approx \boldsymbol{A}^{-1}$$

and transform

$$Q^{-1}Ax = Q^{-1}b$$

Applying the Richardson iteration to this modified system gives

$$x_{k+1} = (I - Q^{-1}A)x_k + Q^{-1}b = Gx_k + c$$

Thus, with an iteration matrix $I - Q^{-1}A \approx 0$, expect more rapid convergence

Common choices for Q

Requirements on a good Q are

Common choices for Q

Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Common choices for Q

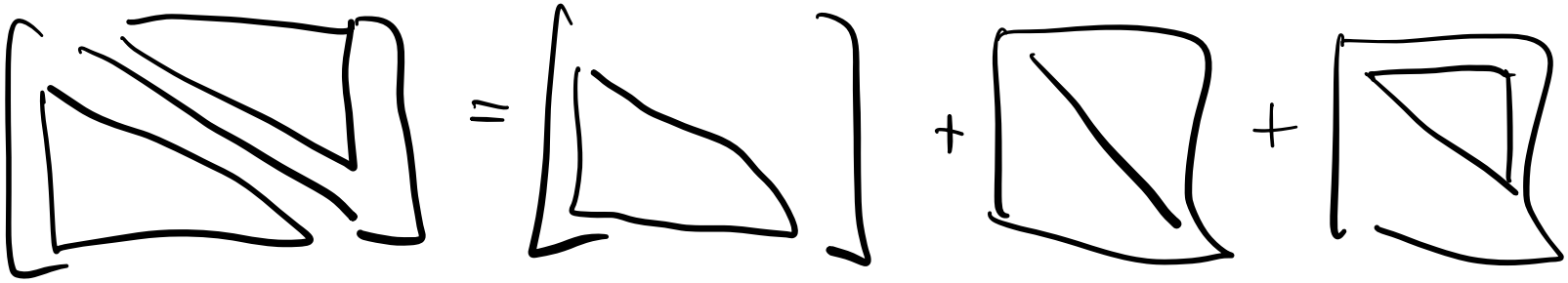
Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Split the matrix A as follows

$$A \quad = \quad L \quad + \quad D \quad + \quad U$$



Jacobi method

Theorem: Select now $Q = D \ldots$ Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

$\rightsquigarrow$ board

Jacobi method

Theorem: Select now $Q = D$ ... Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

$\rightsquigarrow$ board

Notice that $Q = D$ is a good choice in terms of computational costs because we can very quickly solve $Dy = z$ for the diagonal matrix D

Jocobi method

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

$$= (I - D^{-1}(L+D+R))x_k + D^{-1}b$$

$$= \underbrace{-D^{-1}(L+R)}_{G}x_k + D^{-1}b$$

max $|\lambda_i|$

$$\rho(-D^{-1}(L+R)) = \rho(D^{-1}(L+R))$$

$\rho(A) \le \|A\|$ if $\|\cdot\|$ is induced by vector norm

$$\le \|D^{-1}(L+R)\|_\infty$$

$$= \max_i \sum_{i \ne j} \left|\frac{a_{ij}}{a_{ii}}\right| < 1$$

$L+R$

D^{-1}

diagonal dominant

Gauss-Seidel method

Theorem: Choose $Q = D + L \ldots$ Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).
$\rightsquigarrow$ board

Gauss-Seidel convergence

$$x_{k+1} = (I - (D+L)^{-1}A)x_k + (D+L)^{-1}b$$
$$= ((D+L)^{-1}(D+L) - (D+L)^{-1}(D+L+R))x_k + (D+L)^{-1}b$$
$$= -(D+L)^{-1}R x_k + (D+L)^{-1}b$$

For any spd matrix A, we have a scalar product

$$\langle x, y\rangle_A = \langle x, Ay\rangle, \qquad \text{on } \mathbb{R}^n$$

and for any matrix $B \in \mathbb{R}^{n\times n}$, we have the adjoint w.r.t. $\langle \cdot, \cdot\rangle_A$ given by

$$B^* = A^{-1}B^T A$$

so that

$$\underbrace{\langle Bx, y\rangle_A}_{x^T B^T A y} = \langle x, B^* y\rangle_A \qquad \forall x, y \in \mathbb{R}^n$$

$$x^T B^T A y = x^T A A^{-1} B^T A y = x^T A B^* y$$

A self-adjoint matrix $B = B^*$ is positive w.r.t. $\langle \cdot, \cdot\rangle_A$ if $\langle Bx, x\rangle_A > 0 \quad \forall x \neq 0$

First we show: Let $G \in \mathbb{R}^{n \times n}$ with adjoint G^* w.r.t. $\langle \cdot, \cdot \rangle$. Then, if $B = I - G^* G$ is positive w.r.t. $\langle \cdot, \cdot \rangle$, it follows $\rho(G) < 1$.

↳ B positive $\Rightarrow$

$\forall x \neq 0$

$$0 < \langle Bx, x \rangle = \langle x, x \rangle - \langle G^* G x, x \rangle$$
$$= \langle x, x \rangle - \langle Gx, Gx \rangle$$
$$\|x\| > \|Gx\| \quad \forall x \neq 0$$

$$\|G\| = \sup_{\|x\|=1} \frac{\|Gx\|}{\|x\|} < 1$$

implies $\rho(G) \leq \|G\| < 1$.

Now, we can go back to our original problem of showing convergence of Gauss-Seidel for any spd matrix

↳ We show that $B = I - G^* G$ with

$$G = I - (D+L)^{-1} A$$

is a positive matrix w.r.t. $\langle \cdot, \cdot \rangle_A$ with A spd.

↳ Because A spd, have

↳ Because A spd, $A = L + D + R$, we have

$$L^T = R$$

$$G^* = A^{-1} G^T A =$$

$$= I - \underbrace{A^{-1} A^T}\, \underbrace{(D+R)^{-1}}_{(D+L)^{-T}} A$$

$$= I - (D+R)^{-1} A$$

$$\underline{B = I - G^* G} = \dots = (D+R)^{-1} D (D+L)^{-1} A$$

$\Rightarrow$ now show that B pos. wrt. $\langle \cdot, \cdot \rangle_A$

$$\langle Bx, x \rangle_A = \langle \underbrace{(D+R)^{-1}}_{(D+L)^{-T}} D (D+L)^{-1} Ax, Ax \rangle_2$$

$$= \langle \underline{D^{1/2} (D+L)^{-1} Ax}, \underline{D^{1/2} (D+L)^{-1} Ax} \rangle_2$$

$$= \| D^{1/2} (D+L)^{-1} Ax \|_2^2 > 0 \quad \forall x \neq 0$$

$\Rightarrow$ B is positive and thus

$$\rho(G) < 1$$

Gauss-Seidel method

Theorem: Choose $Q = D + L \ldots$ Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).
$\rightsquigarrow$ board

Notice that $Q = D + L$ is a good choice in terms of computational costs because we can very quickly solve $(D + L)y = z$ for the lower triangular matrix $D + L$ (forward substitution)

Relaxation methods:

Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

where $\omega > 0$ is a damping/relaxation parameter (sometimes, $\omega > 1$ is used, leading to overrelaxation). Target is to choose ω such that $\rho(G_\omega)$ is minimal.

Def: A fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with $G = G(A)$ is called *symmetrizable* if for any symmetric positive definite (spd) matrix A, the matrix $I - G$ is similar to an spd matrix, i.e., there is a regular W such that $W(I-G)W^{-1}$ spd.

board

Symmetrizable: need that $I-G$ is similar to an spd matrix:

↳ Richardson: $G = I - A$

$\Rightarrow I - G = A$ is spd if A is spd

↳ Jacobi: $G = I - D^{-1}A$, set $\underline{W} = D^{1/2}$

($W(I-G)W^{-1}$ spd)

$$D^{1/2}(I-G)D^{-1/2} = I - I + D^{1/2}D^{-1}AD^{-1/2}$$
$$= D^{-1/2}AD^{-1/2}$$

which is spd because D has non-zero diagonal entries if A spd

We have the following result for sym. schemes:

$x_{k+1} = G x_k + c$, $G = G(A)$, A spd

$\Rightarrow \sigma(G) \subset (-\infty, 1)$ $\quad (\rho(G) = \max\{|\lambda|\})$

$\sigma(G) = \{\lambda \mid \lambda \text{ eigenvalue of } A\}$

Because $I-G$ is similar to spd matrix, all eigenvalues are real and positive

$\Rightarrow$ eigenvalues of G are

$$1 - \lambda_q(G) > 0$$

$$1 > \lambda_q(G)$$

Finding the optimal damping parameter: $\rightsquigarrow$ board

θ

Finding the optimal damping parameter: $\rightsquigarrow$ board

We obtain that

$$\bar{\omega} = \frac{2}{2 - \lambda_{\max}(G) - \lambda_{\min}(G)}$$

is the optimal damping parameter for symmetrizable iteration methods that minimizes the spectral radius. The spectral radius is

$$\rho(G_{\bar{\omega}}) < 1$$

This means that for a suitable choice $\bar{\omega}$ we can make *any* symmetrizable iteration method convergent for an spd A!

Finding the optimal damping parameter: $\rightsquigarrow$ board

We obtain that

$$\bar{\omega} = \frac{2}{2 - \lambda_{\max}(G) - \lambda_{\min}(G)}$$

is the optimal damping parameter for symmetrizable iteration methods that minimizes the spectral radius. The spectral radius is

$$\rho(G_{\bar{\omega}}) < 1$$

This means that for a suitable choice $\bar{\omega}$ we can make *any* symmetrizable iteration method convergent for an spd A!

Optimal damping parameter for Richardson iteration $\rightsquigarrow$ board

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Iterative methods for systems of linear equations

Today

- Iterative methods for systems of linear equations
- Conjugate gradient method

Announcements

- Homework 4 is due Mon, Nov 4, 2024 before class

Recap: Iterative methods

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration with $\boldsymbol{G} = (I - Q^{-1}A)$ and $c = Q^{-1}b$

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is stationary point

$$\boldsymbol{G}\boldsymbol{x} + \boldsymbol{c} = (\boldsymbol{I} - \boldsymbol{Q}^{-1}\boldsymbol{A})\boldsymbol{x} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x} - \boldsymbol{Q}^{-1}\boldsymbol{b} + \boldsymbol{Q}^{-1}\boldsymbol{b} = \boldsymbol{x}$$

Recap: Common choices for Q

Requirements on a good Q are

Recap: Common choices for Q

Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Recap: Common choices for Q

Requirements on a good Q are

- Q^{-1} is representative of A^{-1}
- We can numerically solve $Qy = z$ much quicker than $Ax = b$ because in each step we solve

$$\boldsymbol{Q}\boldsymbol{x}_{k+1} = (\boldsymbol{Q} - \boldsymbol{A})\boldsymbol{x}_k + b$$

Split the matrix A as follows

$$A \quad = \quad L \quad + \quad D \quad + \quad U$$

Recap: Jacobi method

Theorem: Select now $Q = D \ldots$ Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

Recap: Jacobi method

Theorem: Select now $Q = D$ ... Jacobi method. The Jacobi method converges for any starting point $\boldsymbol{x}_o$ to the solution of $A\boldsymbol{x} = \boldsymbol{b}$ if A is strictly diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \qquad \text{for } i = 1, \ldots, n.$$

Notice that $Q = D$ is a good choice in terms of computational costs because we can very quickly solve $Dy = z$ for the diagonal matrix D

Recap: Gauss-Seidel method

Theorem: Choose $Q = D + L \ldots$ Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).

Recap: Gauss-Seidel method

Theorem: Choose $Q = D + L \ldots$ Gauss-Seidel method. The Gauss-Seidel method converges for any starting point $\boldsymbol{x}_o$ if A is symmetric positive definite (spd).

Notice that $Q = D + L$ is a good choice in terms of computational costs because we can very quickly solve $(D + L)y = z$ for the lower triangular matrix $D + L$ (forward substitution)

Recap: Relaxation methods

Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

where $\omega > 0$ is a damping/relaxation parameter (sometimes, $\omega > 1$ is used, leading to overrelaxation). Target is to choose ω such that $\rho(G_\omega)$ is minimal.

Def: A fixed point method $\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + \boldsymbol{c}$ with $G = G(A)$ is called *symmetrizable* if for any symmetric positive definite (spd) matrix A, the matrix $I - G$ is similar to an spd matrix, i.e., there is a regular W such that $W(I-G)W^{-1}$ spd.

board

Finding the optimal damping parameter: $\rightsquigarrow$ board

$$x_{k+1} = w(Gx_k + c) + (1-w)x_k$$

$$= G_w x_k + wc \quad , \quad G_w = wG + (1-w)I$$

Goal: find $0 < w \leq 1$ with minimal $\rho(G_w)$

The EV of G_w are

$$\lambda_i(G_w) = w\,\lambda_i(G) + 1 - w$$

$$= 1 - w(1 - \lambda_i(G))$$

Because $\lambda_{min}(G) \leq \lambda_{max}(G) < 1$

$$\lambda_i(G_w) < 1$$

and

$$\rho(G_w) = \max\{|1 - w(1-\lambda_{min}(G))|, |1 - w(1-\lambda_{max}(G))|\}$$

Let's look at $w_{min} = \dfrac{1}{1-\lambda_{min}(G)} > 0$

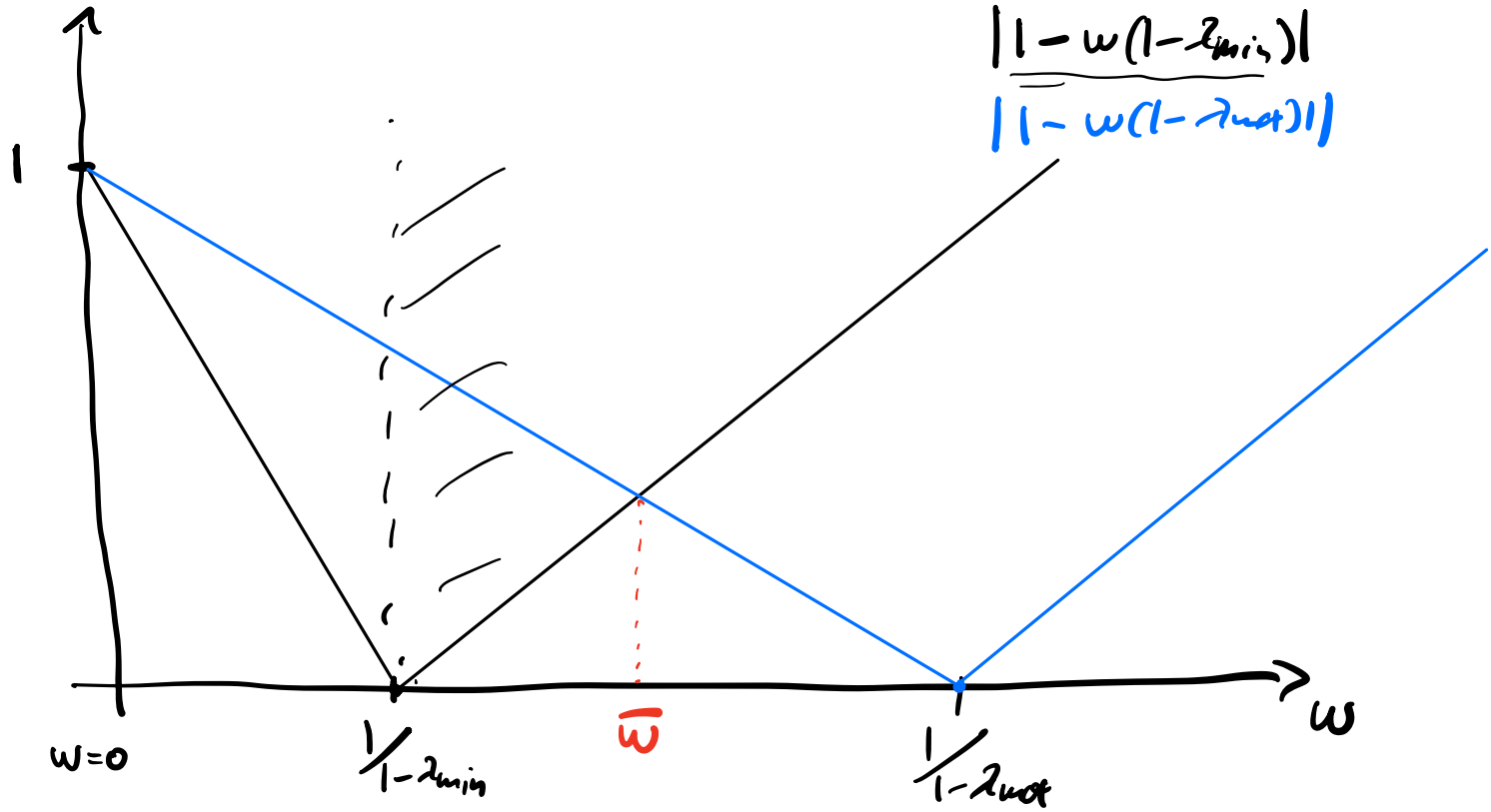
because $0 < 1 - \lambda_{max}(G) \leq 1 - \lambda_{min}(G)$

with w_{min} obtain

$$|1 - w_{min}(1-\lambda_{min}(G))| = 0$$

Similarly, for $w_{max} = \dfrac{1}{1-\lambda_{max}(G)}$ so that

$$|1 - w_{max}(1-\lambda_{max}(G))| = 0$$



optimal $\bar{\omega}$ satisfies

$$-\left[1-\bar{\omega}(1-\lambda_{min}(G))\right] = 1-\bar{\omega}(1-\lambda_{max}(G))$$

solve for $\bar{\omega}$

$$\bar{\omega} = \frac{2}{2-\lambda_{max}(G)-\lambda_{min}(G)}$$

Example: Richardson iteration with spd matrix

↳ for spd matrix A have $\qquad G = I - A$

$$\lambda_{min}(G) = 1-\lambda_{max}(A)$$
$$\lambda_{max}(G) = 1-\lambda_{min}(A)$$

$$\bar{\omega} = \frac{2}{\lambda_{max}(A)+\lambda_{min}(A)}$$

Spectral radius

$$\rho(G_\omega) = \rho(I - \bar{\omega} A) =$$

$$= \frac{\lambda_{max}(A) - \lambda_{min}(A)}{\lambda_{max}(A) + \lambda_{min}(A)}$$

$$= \frac{1/\lambda_{min}}{1/\lambda_{min}} \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \frac{k_2(A) - 1}{k_2(A) + 1} < 1$$

$\Rightarrow$ notice that the condition number of A enters

$\Rightarrow$ different from direct methods

$\Rightarrow$ if $k_2(A)$ is high, then

$$\frac{k_2(A) - 1}{k_2(A) + 1} \approx 1$$

$\Rightarrow$ very slow convergence

Finding the optimal damping parameter: $\rightsquigarrow$ board

We obtain that

$$\bar{\omega} = \frac{2}{2 - \lambda_{\max}(G) - \lambda_{\min}(G)}$$

is the optimal damping parameter for symmetrizable iteration methods that minimizes the spectral radius. The spectral radius is

$$\rho(G_{\bar{\omega}}) < 1$$

This means that for a suitable choice $\bar{\omega}$ we can make *any* symmetrizable iteration method convergent for an spd A!

Finding the optimal damping parameter: $\rightsquigarrow$ board

We obtain that

$$\bar{\omega} = \frac{2}{2 - \lambda_{\max}(G) - \lambda_{\min}(G)}$$

is the optimal damping parameter for symmetrizable iteration methods that minimizes the spectral radius. The spectral radius is

$$\rho(G_{\bar{\omega}}) < 1$$

This means that for a suitable choice $\bar{\omega}$ we can make *any* symmetrizable iteration method convergent for an spd A!

Optimal damping parameter for Richardson iteration $\rightsquigarrow$ board

Algorithmic perspective on iterative methods

Another interpretation of Richardson iterations?

$$\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + b$$

Algorithmic perspective on iterative methods

Another interpretation of Richardson iterations?

$$\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + b = \boldsymbol{x}_k + (b - A\boldsymbol{x}_k)$$

Algorithmic perspective on iterative methods

Another interpretation of Richardson iterations?

$$\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + b = \boldsymbol{x}_k + (b - A\boldsymbol{x}_k)$$

```
for k = 0, 1, ...
 for i = 0, 1, ..., n-1:
```
$x_{k+1}[i] = x_k[i] + r_k[i]$

where the residual $\boldsymbol{r}_k$ at iteration k is given by

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$$

What would we like to update $\boldsymbol{x}_k$ with?

Algorithmic perspective on iterative methods

Another interpretation of Richardson iterations?

$$\boldsymbol{x}_{k+1} = (I - A)\boldsymbol{x}_k + b = \boldsymbol{x}_k + (b - A\boldsymbol{x}_k)$$

```
for k = 0, 1, ...
 for i = 0, 1, ..., n-1:
```
$x_{k+1}[i] = x_k[i] + r_k[i]$

where the residual $\boldsymbol{r}_k$ at iteration k is given by

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$$

What would we like to update $\boldsymbol{x}_k$ with? We would like to update $\boldsymbol{x}_k$ in the direction of the error $\boldsymbol{e}_k = \boldsymbol{x} - \boldsymbol{x}_k$ because then

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{e}_k = \boldsymbol{x}$$

However, we don't have the error $\boldsymbol{e}_k$ and therefore it is reasonable to use the next best thing which is the residual in many situations

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}\boldsymbol{x}_k = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{A}\boldsymbol{e}_k$$

$\rightsquigarrow$ Richardson iteration updates $\boldsymbol{x}_k$ in the direction of the residual $\boldsymbol{r}_k$

Jacobi iterations

$$\boldsymbol{x}_{k+1} = (I - D^{-1}A)\boldsymbol{x}_k + D^{-1}b$$

```
for k = 0, 1, ...
 for i = 0, 1, ..., n-1:
 for i = 0, 1, ..., n-1:
```

$y[i] = 1/a_{ii} r_k[i]$

$x_{k+1}[i] = x_k[i] + y[i]$

- In every substep i of iteration k, an update $y[i]$ is computed and stored
- Applied immediately, this would lead to the (momentary) disappearance of the i-th component of the residual r_k
- Thus, with this current approximation, equation i would be solved exactly—an improvement that would be lost immediately in the following substep for the equation $i + 1$
- However, the updates of a component are not applied immediately but only at the end of an iteration step (second i-loop)

Gauss-Seidel iteration

$$\boldsymbol{x}_{k+1} = (I - (L+D)^{-1}A)\boldsymbol{x}_k + (L+D)^{-1}b$$

```
for k = 0, 1, ...
 for i = 0, 1, ..., n-1:
```

$$r_k[i] = b[i] - \sum_{j=1}^{i-1} a_{ij}x_{k+1}[j] - \sum_{j=i}^{n} a_{ij}x_k[j]$$
$$y[i] = 1/a_{ii}r_k[i], \quad x_{k+1}[i] = x_k[i] + y[i]$$

- In contrast to Jacobi method, the update is performed immediately
- Therefore the new modified values for components $1, ..., i-1$ are already available for updating component i

Damping (mostly Jacobi) or over-relaxation (mostly Gauss-Seidel) means to take step lengths different from 1 in the direction of the residual

The spectral radius of typical iterative matrices

- The spectral radius ρ determines convergence and speed; the smaller ρ, the faster the error decays. In practice, ρ is often very close to 1 so that even though $\rho < 1$ it takes an unreasonable amount of iterations to get a reasonable answer

- An important sample scenario is the discretization of PDEs: It is typical that ρ depends on the dimension n of the matrix A, and thus in terms of PDE discretization it depends on the mesh width h of the underlying grid. For example

$$\rho \in \mathcal{O}(1 - h_l^2) = \mathcal{O}(1 - \frac{1}{4^l})$$

  with mesh width $h_l = 2^{-l}$ in one dimension ($\mathcal{O}(1 - 16^{-l})$ in two spatial dimensions)

- This is a huge disadvantage: Why?

The spectral radius of typical iterative matrices

- The spectral radius ρ determines convergence and speed; the smaller ρ, the faster the error decays. In practice, ρ is often very close to 1 so that even though $\rho < 1$ it takes an unreasonable amount of iterations to get a reasonable answer

- An important sample scenario is the discretization of PDEs: It is typical that ρ depends on the dimension n of the matrix A, and thus in terms of PDE discretization it depends on the mesh width h of the underlying grid. For example

$$\rho \in \mathcal{O}(1 - h_l^2) = \mathcal{O}(1 - \frac{1}{4^l})$$

  with mesh width $h_l = 2^{-l}$ in one dimension ($\mathcal{O}(1 - 16^{-l})$ in two spatial dimensions)

- This is a huge disadvantage: Why? the finer the grid is (and therefore the more accurate our approximation of the PDE solution should be), the slower these iterative methods get.

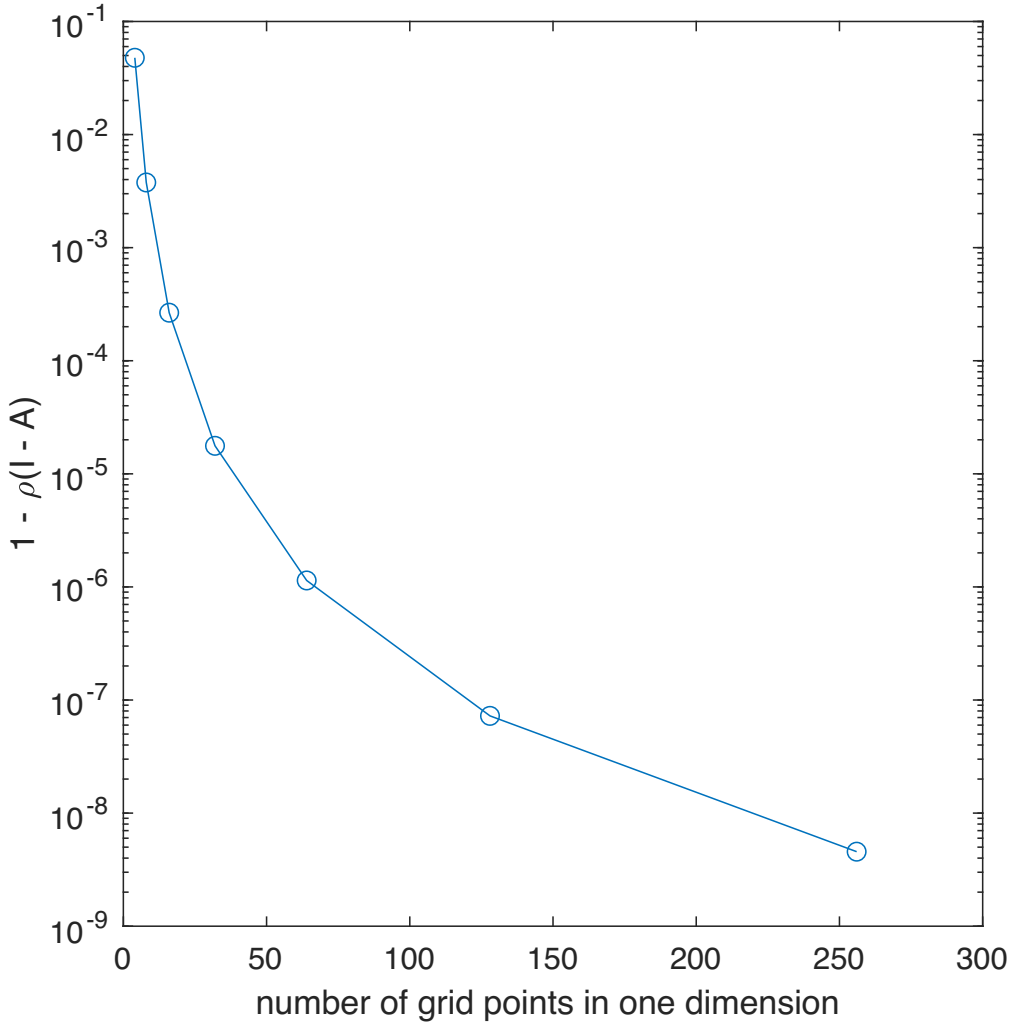
Consider the Laplace equation

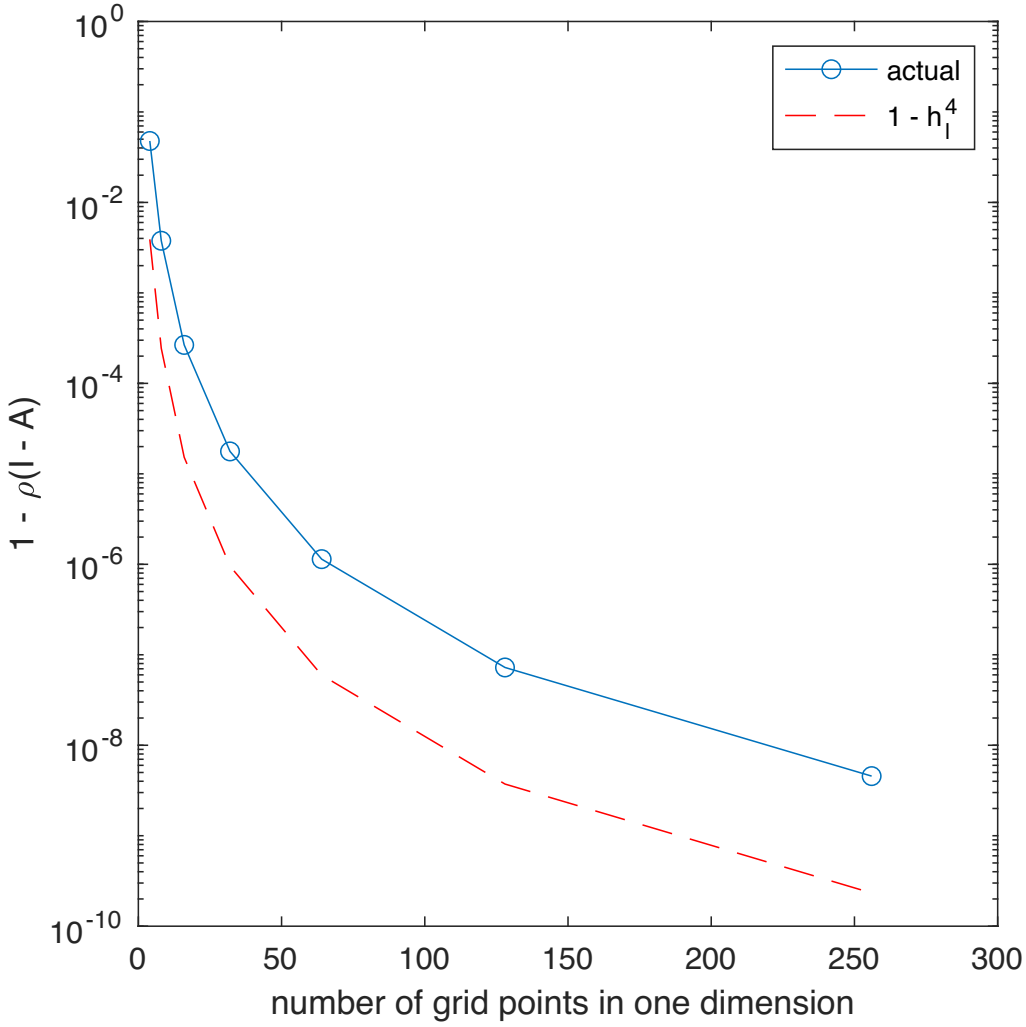
$$\Delta u(x_1, x_2) = 0\,,$$

in two dimensions and discretize with five-point second-order finite-difference stencil on $N \times N$ grid points ($\rightsquigarrow$ NM2)

Plot the spectral radius $\rho(I - A)$ (Richardson interpolation) w.r.t. number of grid points

```
Nlist = 2.^(2:8);
eList = [];
for N=Nlist
    X = gallery('poisson', N)*(1/N^2);
    eList(end + 1) = eigs(speye(N^2) - X, 1);
end
```

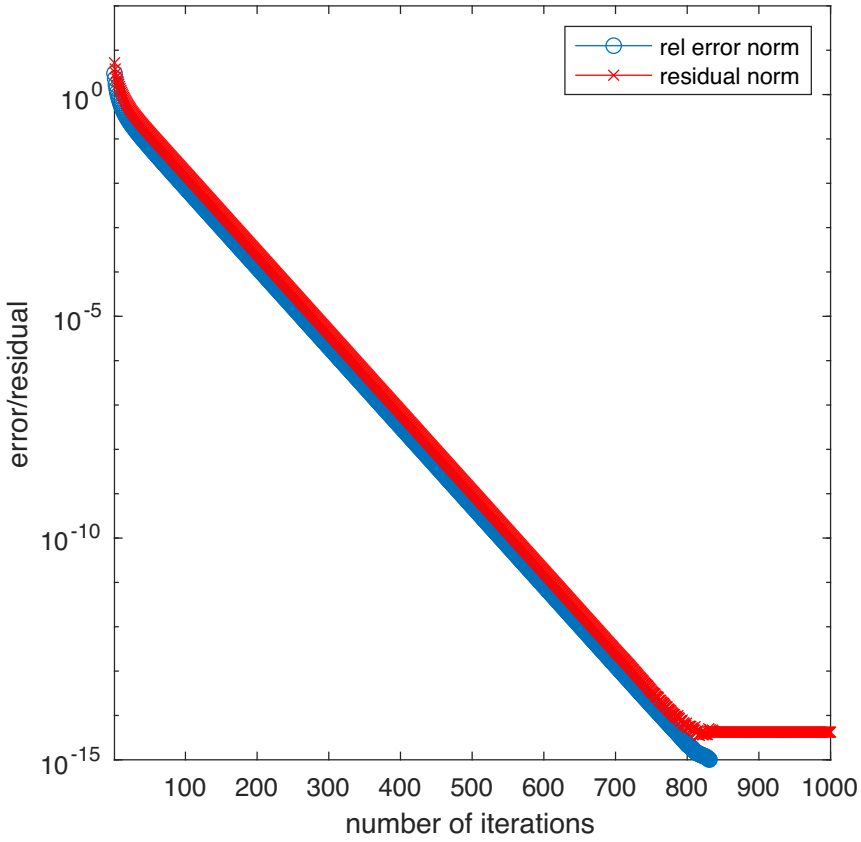

10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
1 - ρ(I - A)
0
50
100
150
200
250
300
number of grid points in one dimension


actual
1 - h_l^4
1 - ρ(I - A)
10^0
10^-2
10^-4
10^-6
10^-8
10^-10
0
50
100
150
200
250
300
number of grid points in one dimension

Running Jacobi on a Poisson matrix with $N = 10$ and $N = 100$ grid points in each dimension:

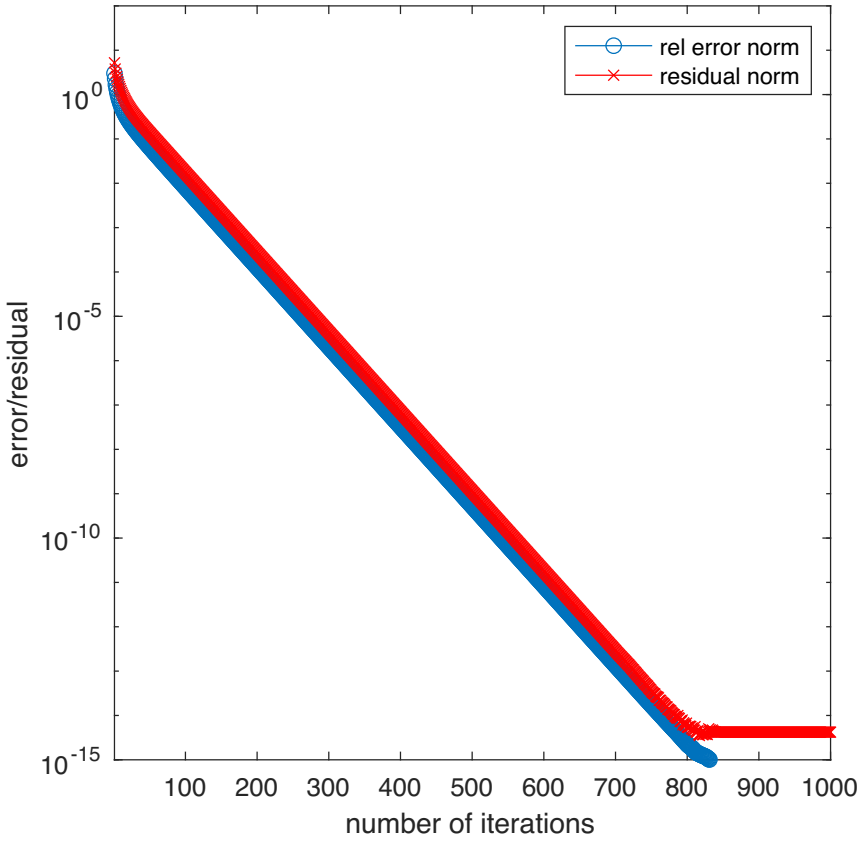
```
A = gallery('poisson', N)*(1/N^2);
b = randn(N^2, 1);
x = randn(N^2, 1);
xTrue = A\b;
M = speye(N^2) - spdiags(1./diag(A), 0, N^2, N^2)*A;
c = spdiags(1./diag(A), 0, N^2, N^2)*b;

hist = [];
for iter=1:1000
    x = M*x + c;
    hist(iter, 1) = norm(x - xTrue)/norm(x);
    hist(iter, 2) = norm(A*x - b);
end
```

$N = 10$

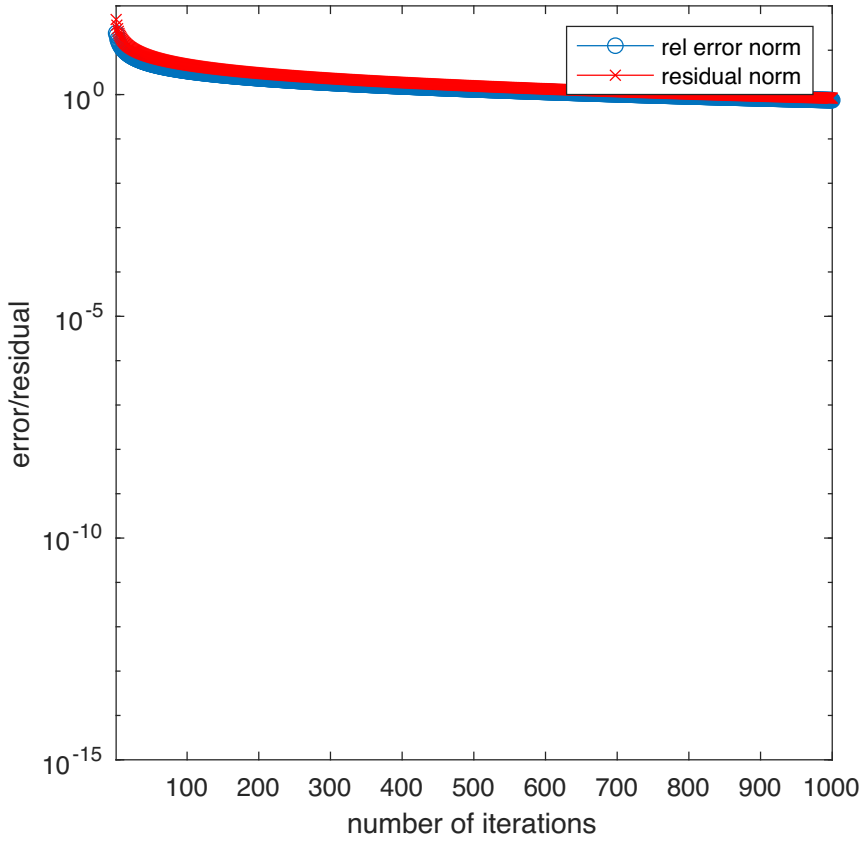

rel error norm
residual norm
error/residual
number of iterations
10^{0}
10^{-5}
10^{-10}
10^{-15}
100
200
300
400
500
600
700
800
900
1000

$N = 100$

$N = 10$

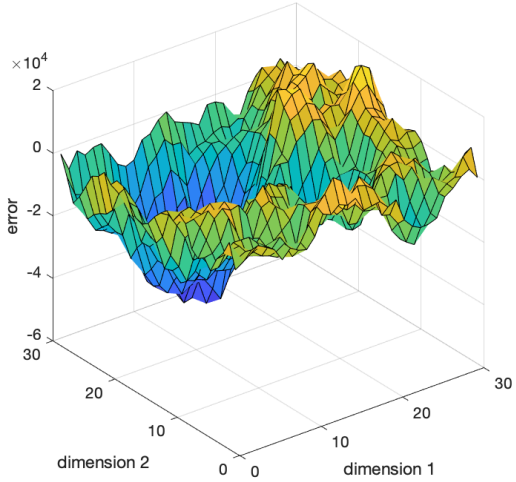

rel error norm
residual norm
error/residual
number of iterations
10^{0}
10^{-5}
10^{-10}
10^{-15}
100 200 300 400 500 600 700 800 900 1000

$N = 100$

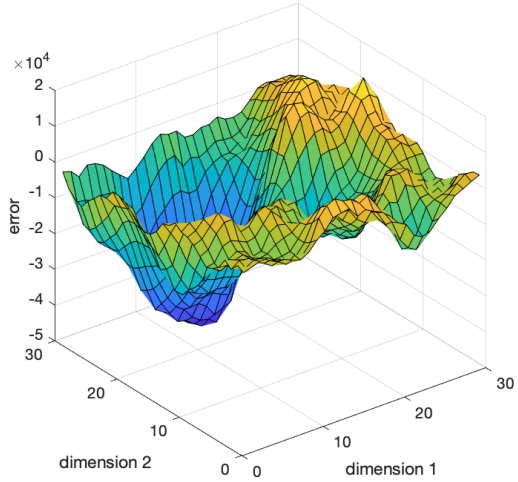

rel error norm
residual norm
error/residual
number of iterations
10^{0}
10^{-5}
10^{-10}
10^{-15}
100 200 300 400 500 600 700 800 900 1000

First few iterations of Jacobi relaxation

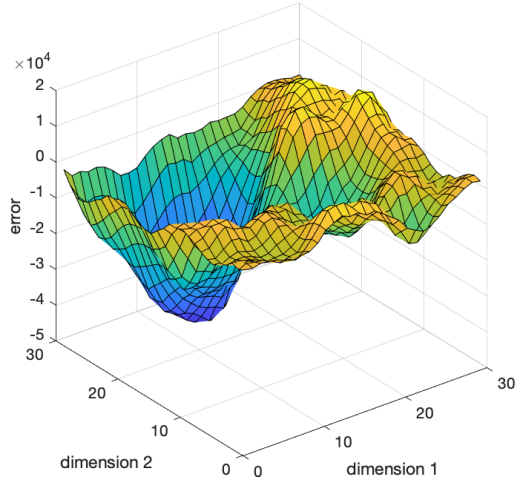
```
N = 30; A = gallery('poisson', N)*(1/N^2);
[X, Y] = meshgrid(linspace(1, N, N), linspace(1, N, N));
b = 10*randn(N^2, 1);
x = randn(N^2, 1);
xTrue = A\b;

for i=1:5
    x = (speye(N^2) - spdiags(1./diag(A), 0, N^2, N^2)*A)*x + ...
                spdiags(1./diag(A), 0, N^2, N^2)*b;
end
```



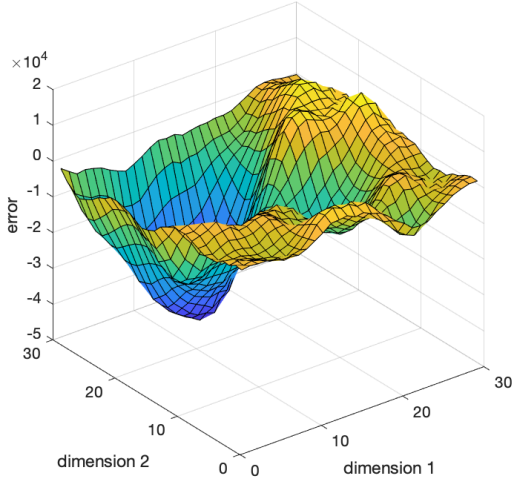
iteration 0



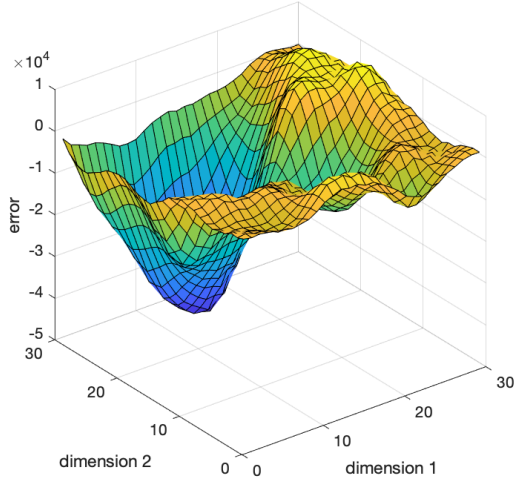
iteration 1



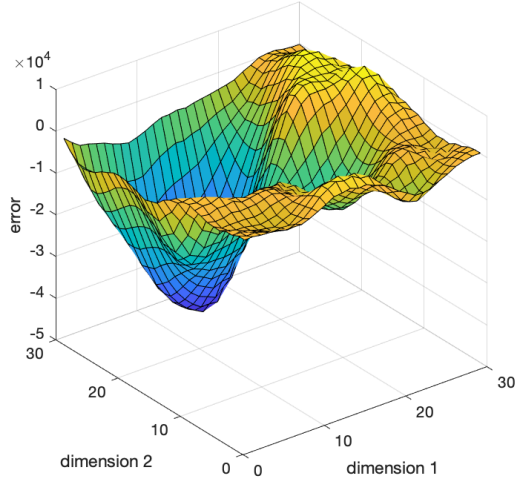
iteration 2



iteration 3



iteration 4

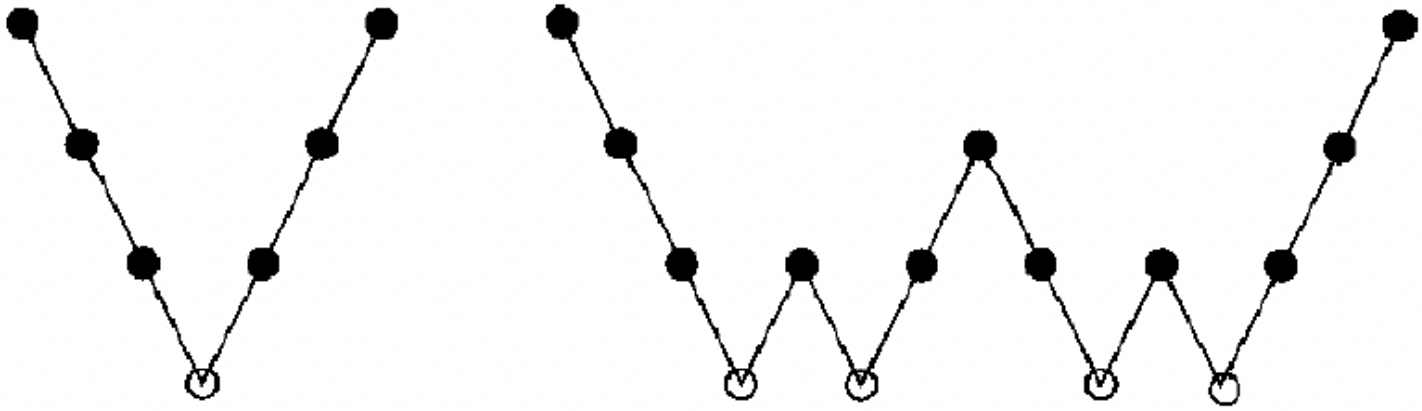


iteration 5

- Relaxation methods such as Jacobi and Gauss Seidel have a smoothing effect on the error

- Even if $\rho \approx 1$, only a few iterations are necessary to obtain a smooth error; this means that high-frequency error is reduced very quickly whereas the low-frequency error is reduced slowly

- Multigrid methods exploit this effect and represent the smoothed error on a coarse grid, where it becomes high-frequency error again, which can be smoothed quickly



$\rightsquigarrow$ Multigrid methods are among the most efficient solvers for PDE problems $\rightsquigarrow$ NM 2 in Spring

Conjugate gradient method

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

$$f(x) = \frac{1}{2}x^T Ax - b^T x\,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, the function f is convex. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b = -r(x) = 0 \iff Ax = b$$

What is the benefit of this point of view?

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

$$f(x) = \frac{1}{2}x^T Ax - b^T x\,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, the function f is convex. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b = -r(x) = 0 \iff Ax = b$$

What is the benefit of this point of view? We now can let loose all what we know about optimization to solve $Ax = b$

Our first try is applying the method of steepest descent in the direction of the negative gradient

$$-\nabla f = r$$

which happens to be the residual

$$r_k = b - Ax_k$$
$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$$
$$x_{k+1} = x_k + \alpha_k r_k$$

The step length α_k minimizes $f(x_k + \alpha_k r_k)$ as a function of α_k $\rightsquigarrow$ board

(This is the same as Richardson iterations with damping α_k; what is the difference between α_k and $\bar{\omega}$? $\rightsquigarrow$
)

Step length

$$f(x) = \frac{1}{2}x^T A x - b^T x, \qquad \underline{\nabla f(x) = Ax - b}$$

$$\min_{\alpha_k} \; f(x_k + \alpha_k r_k) = \frac{1}{2}(x_k + \alpha_k r_k)^T A (x_k + \alpha_k r_k) - b^T (x_k + \alpha_k r_k)$$

$$\frac{\partial}{\partial \alpha_k} f(x_k + \alpha_k r_k) = r_k^T \left[\nabla f(x_k + \alpha_k r_k)\right]$$

$$= r_k^T \left[A x_k + A \alpha_k r_k - b\right] \overset{!}{=} 0$$

$$r_k^T A x_k + \alpha_k r_k^T A r_k - r_k^T b = 0$$

$$r_k^T (\underbrace{A x_k - b}_{r_k}) + \alpha_k r_k^T A r_k = 0$$

$$\Rightarrow \alpha_k^* = \frac{r_k^T r_k}{r_k^T A r_k}$$

Our first try is applying the method of steepest descent in the direction of the negative gradient

$$-\nabla f = r$$

which happens to be the residual

$$r_k = b - Ax_k$$
$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$$
$$x_{k+1} = x_k + \alpha_k r_k$$

The step length α_k minimizes $f(x_k + \alpha_k r_k)$ as a function of α_k $\rightsquigarrow$ board

(This is the same as Richardson iterations with damping α_k; what is the difference between α_k and $\bar{\omega}$? $\rightsquigarrow$ α_k is computable with just mat-vec with A, whereas $\bar{\omega}$ depends on eigenvalues of A)

For steepest descent, if A is spd, we obtain

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y\rangle_A = x^T A y$ and $\|\cdot\|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

Proof $\rightsquigarrow$ board

$$\|x_k - x^*\|_A \leq \left(\frac{k_2(A)-1}{k_2(A)+1}\right)^k \|x^* - x_0\|_A$$

Consider

$$r_k = b - Ax_k = Ax^* - Ax_k$$

$$x_{k+1}(\alpha) = x_k + \alpha r_k$$

$$= x_k + \alpha A(x^* - x_k)$$

$$\underbrace{x^* - x_{k+1}(\alpha)} = x^* - x_k - \alpha A(x^* - x_k)$$

$$= (I - \alpha A)\underbrace{(x^* - x_k)}$$

$$e_{k+1}(\alpha) = (I - \alpha A)\, e_k$$

$$\|e_{k+1}(\alpha)\|_A^2 = e_k^T (I - \alpha A)^T A (I - \alpha A)\, e_k$$

Because spd matrix, have orthogonal eigenbasis w.r.t. $\langle \cdot, \cdot \rangle_2$

$$\{z_1, \dots, z_N\} \text{ and } \{\lambda_1, \dots, \lambda_N\}$$

so that we find $\{a_i\}_{i=1}^N \subseteq \mathbb{R}$

$$e_k = \sum_{j=1}^N a_j z_j$$

$$\|e_k\|_A^2 = \dots = \sum_{j=1}^N a_j^2 \lambda_j$$

$$\|e_{k+1}(\alpha)\|_A^2 = \dots = \sum_{j=1}^N \lambda_j a_j^2 (1 - \alpha\lambda_j)^2$$

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Iterative methods for systems of linear equations

Today

- Conjugate gradient method

Announcements

- Homework 4 is due Mon, Nov 4, 2024 before class

Recap

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

$$f(x) = \frac{1}{2}x^T Ax - b^T x\,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, the function f is convex. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b = -r(x) = 0 \iff Ax = b$$

What is the benefit of this point of view?

Recap

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

$$f(x) = \frac{1}{2}x^T Ax - b^T x\,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, the function f is convex. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b = -r(x) = 0 \iff Ax = b$$

What is the benefit of this point of view? We now can let loose all what we know about optimization to solve $Ax = b$

Recap

Our first try is applying the method of steepest descent in the direction of the negative gradient

$$-\nabla f = r$$

which happens to be the residual

$$r_k = b - Ax_k$$

$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$$

$$x_{k+1} = x_k + \alpha_k r_k$$

The step length α_k minimizes $f(x_k + \alpha_k r_k)$ as a function of α_k $\rightsquigarrow$ last time

(This is the same as Richardson iterations with damping α_k; what is the difference between α_k and $\bar{\omega}$? $\rightsquigarrow$

)

Recap

Our first try is applying the method of steepest descent in the direction of the negative gradient

$$-\nabla f = r$$

which happens to be the residual

$$r_k = b - Ax_k$$
$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$$
$$x_{k+1} = x_k + \alpha_k r_k$$

The step length α_k minimizes $f(x_k + \alpha_k r_k)$ as a function of α_k $\rightsquigarrow$ last time

(This is the same as Richardson iterations with damping α_k; what is the difference between α_k and $\bar{\omega}$? $\rightsquigarrow$ α_k is computable with just mat-vec with A, whereas $\bar{\omega}$ depends on eigenvalues of A)

Recap

For steepest descent, if A is spd, we obtain

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y\rangle_A = x^T A y$ and $\|\cdot\|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

Proof $\rightsquigarrow$ board

$$x_{k+1}(\alpha) = x_k + \alpha r_k$$

$$\underbrace{x^* - x_{k+1}(\alpha)}_{e_{k+1}(\alpha)} = (I - \alpha A) e_k$$

Because A is spd, have orthogonal eigenbasis

$$\{z_1, \dots, z_N\} \quad \text{and} \quad \{\lambda_1, \dots, \lambda_N\}$$

so that we can find $\{a_i\}_{i=1}^N$ with

$$e_k = \sum_{j=1}^N a_j z_j$$

$$\|e_{k+1}(\alpha)\|_A^2 = \dots = \underline{\sum_{j=1}^N \lambda_j a_j^2 (1 - \alpha \lambda_j)^2}$$

Set $\hat{\alpha} = \dfrac{2}{\underset{\text{max}}{\lambda_1} + \underset{\text{min}}{\lambda_N}}$

$$\|e_{k+1}(\hat{\alpha})\|_A^2 = \sum_{j=1}^N \lambda_j a_j^2 \left(\frac{\lambda_1 + \lambda_N - 2\lambda_j}{\lambda_1 + \lambda_N} \right)^2$$

$$= \sum_{j=1}^N \lambda_j a_j^2 \frac{(\lambda_1 - \lambda_N)^2 - 4(\lambda_1 - \lambda_j)(\lambda_j - \lambda_N)}{(\lambda_1 + \lambda_N)^2}$$

$$\leq \left(\frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N} \right)^2 \sum_{j=1}^N \lambda_j a_j^2 = \left(\frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N} \right)^2 \|e_k\|_A^2$$

Steepest descent chooses α_k such that it minimizes

$$f(x_{k+1}) = f(x_k + \alpha_k r_k)$$

at each step

For the minimizer x^* we have

$$f(x) = \underline{\underline{f(x^*)}} + \frac{1}{2}\|x^* - x\|_A^2 \quad \rightsquigarrow \text{HW4}$$

Thus

$$\min_{\alpha_k} f(x_k + \alpha_k r_k) \iff \min_{\alpha_k} \frac{1}{2}\|\underbrace{x^* - (x_k + \alpha_k r_k)}_{e_{k+1}(\alpha_k)}\|_A^2$$

$$\|e_{k+1}(\alpha_k)\|_A = \min_{\alpha} \|e_{k+1}(\alpha)\|_A$$

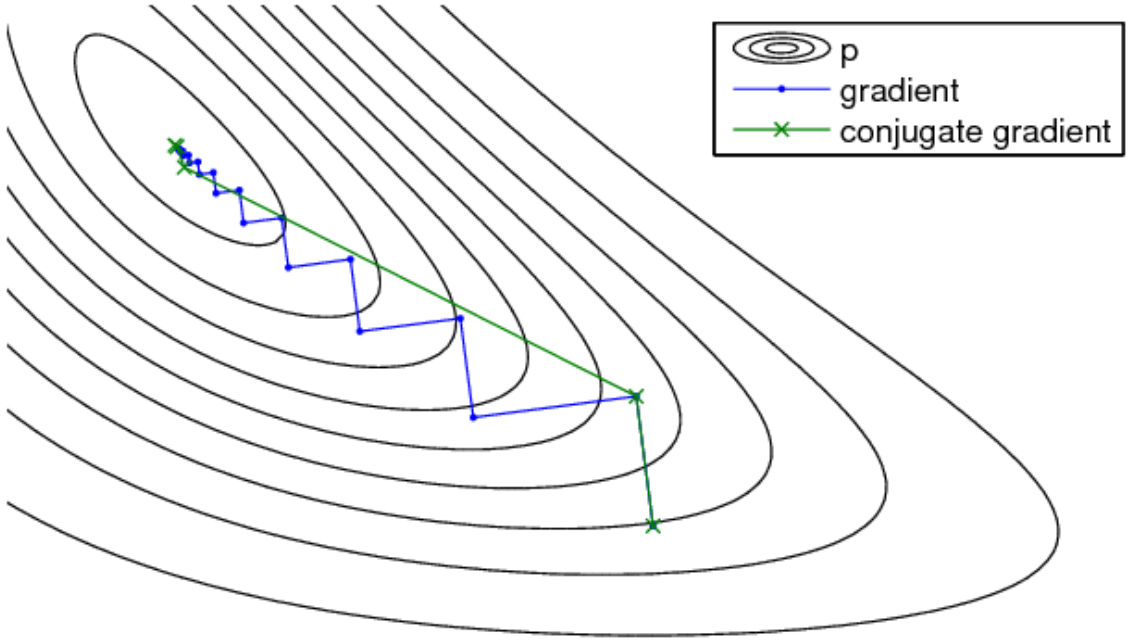
$$\leq \|e_{k+1}(\hat{\alpha})\|_A$$

$$\leq \left(\frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N}\right) \|e_k\|_A$$

By induction

$$\|e_k\|_A \leq \left(\frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N}\right)^k \|e_0\|_A$$

$$\frac{k_2(A) - 1}{k_2(A) + 1} = \frac{\lambda_1/\lambda_N - 1}{\lambda_1/\lambda_N + 1} = \frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N}$$



[Figure: Kuusela et al., 2009]

The convergence behavior of steepest descent in this context can be poor: we eventually get arbitrarily close to the minimum but we can always destroy something of the already achieved when applying the update $\rightsquigarrow$ can we find better search directions?

Conjugate gradient method

- What do all iterative methods we looked at so far have in common?

Conjugate gradient method

- What do all iterative methods we looked at so far have in common?
- All methods so far use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

Conjugate gradient method

- What do all iterative methods we looked at so far have in common?
- All methods so far use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

- The conjugate gradient (CG) method is a variation of steepest descent that *has a memory*.

- Let $p_1, \ldots, p_k$ be the directions up to step k, then CG uses the space

$$x_0 + \operatorname{span}\{p_1, \ldots, p_k\}, \qquad x_0 \text{ starting point}$$

  to find the next iterate x_k and thus

$$x_k = x_0 + \sum_{i=1}^{k} \alpha_i p_i$$

- (Recall that steepest descent uses only the search direction $p_k = r_{k-1} = -\nabla f(x_{k-1})$ to find the iterate x_k)

We want the following

a The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \mathrm{span}(p_1, \ldots, p_k)} f(x)$$

c The step x_k can be calculated easily from x_{k-1}

What do conditions (a) and (b) guarantee?

We want the following

a The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \text{span}(p_1, \ldots, p_k)} f(x)$$

c The step x_k can be calculated easily from x_{k-1}

What do conditions (a) and (b) guarantee? Convergence in N steps because at the N-th step we have $x_0 + \text{span}(p_1, \ldots, p_N) = \mathbb{R}^N$ and thus we minimize f over $\mathbb{R}^N$

Let's start by writing

$$x_k = x_0 + P_{k-1}y + \alpha p_k \,,$$

where $P_{k-1} = [p_1, \ldots, p_{k-1}] \in \mathbb{R}^{N\times(k-1)}, y \in \mathbb{R}^{k-1}, \alpha \in \mathbb{R}$.

Our aim is to determine y and α. So let's look at minimizing $f(x_k)$ w.r.t. y and α

$$f(x_k) = \cdots = \underbrace{f(x_0 + P_{k-1}y)}_{\text{only depends on } y \text{ not on } \alpha} + \alpha p_k^T A P_{k-1} y + \underbrace{\frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T r_0}_{\text{only depends on } \alpha \text{ not on } y}$$

(recall that $f(x) = \frac{1}{2}x^T Ax - b^T x$).

Let's start by writing

$$x_k = x_0 + P_{k-1}y + \alpha p_k \,,$$

where $P_{k-1} = [p_1, \ldots, p_{k-1}] \in \mathbb{R}^{N\times(k-1)}, y \in \mathbb{R}^{k-1}, \alpha \in \mathbb{R}$.

Our aim is to determine y and α. So let's look at minimizing $f(x_k)$ w.r.t. y and α

$$f(x_k) = \cdots = \underbrace{f(x_0 + P_{k-1}y)}_{\text{only depends on } y \text{ not on } \alpha} + \alpha p_k^T A P_{k-1} y + \underbrace{\frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T r_0}_{\text{only depends on } \alpha \text{ not on } y}$$

(recall that $f(x) = \frac{1}{2}x^T Ax - b^T x$).
The mixed term in the middle depends on α and y, otherwise we could optimize separately for y and α. How should we choose p_k?

Let's start by writing

$$x_k = x_0 + P_{k-1}y + \alpha p_k \,,$$

where $P_{k-1} = [p_1, \ldots, p_{k-1}] \in \mathbb{R}^{N\times(k-1)}, y \in \mathbb{R}^{k-1}, \alpha \in \mathbb{R}$.

Our aim is to determine y and α. So let's look at minimizing $f(x_k)$ w.r.t. y and α

$$f(x_k) = \cdots = \underbrace{f(x_0 + P_{k-1}y)}_{\text{only depends on } y \text{ not on } \alpha} + \alpha p_k^T A P_{k-1} y + \underbrace{\frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T r_0}_{\text{only depends on } \alpha \text{ not on } y}$$

(recall that $f(x) = \frac{1}{2}x^T Ax - b^T x$).
The mixed term in the middle depends on α and y, otherwise we could optimize separately for y and α. How should we choose p_k?

Let's choose the search direction p_k such that

$$p_k^T A P_{k-1} = 0$$

which means

$$p_k \in \text{span}\{Ap_1, \ldots, Ap_{k-1}\}^{\perp}$$

Thus, with $p_k^T A P_{k-1} = 0$ we get

$$\min_{x_k \in x_0 + \text{span}\{p_1,\dots,p_k\}} f(x_k) = \min_{y \in \mathbb{R}^{k-1}} f(x_0 + P_{k-1} y) + \min_{\alpha \in \mathbb{R}} \left(\frac{\alpha^2}{2} p_k^T A p_k - \alpha p_k^T r_0 \right)$$

- The first minimization problem is solved for $y = y_{k-1}$ computed from step $k-1$ and then $x_{k-1} = x_0 + P_{k-1} y_{k-1}$ satisfies

$$f(x_{k-1}) = \min_{x_0 + \text{span}\{p_1,\dots,p_{k-1}\}} f(x)$$

- The solution to the second minimization problem is just a scalar

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}$$

$\rightsquigarrow$ satisfy conditions (b) and (c) from above.

- We said the search directions $p_1, \ldots, p_k$ have to be conjugate, i.e., orthogonal w.r.t. A

$$p_i^T A p_j = 0\,, \qquad i, j = 1, \ldots, k\,, i \neq j \tag{1}$$

- One can show that (1) implies that $p_1, \ldots, p_k$ are linearly independent (w.r.t. $\langle \cdot, \cdot, \rangle$), which satisfies condition (a)

- To find the search direction p_k, we want to combine positive aspects of steepest descent and conjugate gradients. In steepest descent we have $p_k = r_{k-1}$. So let's stay close to r_{k-1} but additionally enforce that p_k is A-conjugate to previous search directions $p_1, \ldots, p_{k-1}$

How can we achieve this?

- We said the search directions $p_1, \ldots, p_k$ have to be conjugate, i.e., orthogonal w.r.t. A

$$p_i^T A p_j = 0\,, \qquad i, j = 1, \ldots, k\,, i \neq j \tag{1}$$

- One can show that (1) implies that $p_1, \ldots, p_k$ are linearly independent (w.r.t. $\langle \cdot, \cdot, \rangle$), which satisfies condition (a)

- To find the search direction p_k, we want to combine positive aspects of steepest descent and conjugate gradients. In steepest descent we have $p_k = r_{k-1}$. So let's stay close to r_{k-1} but additionally enforce that p_k is A-conjugate to previous search directions $p_1, \ldots, p_{k-1}$

How can we achieve this? $\rightsquigarrow$ Gram-Schmidt orthogonalization

Apply Gram-Schmidt to r_{k-1} so that we obtain p_k that is A-conjugate to $p_1, \ldots, p_{k-1}$

$$p_k = r_{k-1} - \sum_{j=1}^{k-1} \frac{\langle r_{k-1}, p_j \rangle_A}{\|p_j\|_A^2} p_j$$

We need following technical statements $\rightsquigarrow$ board:

- If $r_{k-1} = b - Ax_{k-1} \neq 0$, then there exists $p_k \in \operatorname{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ such that $p_k^T r_{k-1} \neq 0$ and $p_k^T r_{k-1} = p_k^T r_0$
- It then follows that (why is this helpful?)

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k} = \frac{p_k^T r_{k-1}}{p_k^T A p_k}$$

- If $r_j \neq 0$ for $j < k$, then

$$\langle r_{k-1}, p_j \rangle_A = 0\,, \qquad j < k-1\,.$$

CG technical lemma to:

$$r_{k-1} = b - Ax_{k-1} \neq 0$$

$$\Rightarrow \exists\, p_k \in \operatorname{span}\{Ap_1, \dots, Ap_{k-1}\}^{\perp}$$

with $p_k^T r_{k-1} \neq 0$

for $k=1$: set $p_1 = r_0$

for $k>1$: $r_{k-1} \neq 0$, then

$$x^* = A^{-1} b \notin x_0 + \operatorname{span}\{p_1, \dots, p_{k-1}\}$$

$$b \notin Ax_0 + \operatorname{span}\{Ap_1, \dots, Ap_{k-1}\}$$

$$\underbrace{b - Ax_0}_{r_0} \notin \operatorname{span}\{Ap_1, \dots, Ap_{k-1}\}$$

Thus, there exists $\underline{p_k \in \operatorname{span}\{Ap_1, \dots, Ap_{k-1}\}^{\perp}}$

with $p_k^T r_0 \neq 0$

Select such a $p_k \in \operatorname{span}\{\dots\}^{\perp}$ with $p_k^T r_0 \neq 0$, then

$$\begin{aligned} p_k^T r_{k-1} &= p_k^T (b - Ax_{k-1}) \\ &= p_k^T (b - A(x_0 + P_{k-1} y_{k-1})) \\ &= \underbrace{p_k^T (b - Ax_0)}_{p_k^T r_0} - \underbrace{p_k^T A P_{k-1}}_{=0} y_{k-1} \neq 0 \end{aligned}$$

Additionally

$$p_k^T r_{k-1} = p_k^T r_0$$

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k} = \frac{p_k^T r_{k-1}}{p_k^T A p_k}$$

- Now want to convince us that it holds

$$\forall p \in \text{span}\{p_1, \dots, p_k\}: \; p^T r_k = 0$$

Recall from lecture about lsq problems

$$r = b - A\bar{x} \perp \text{columnspan}(A)$$

Condition (b) of CG tells us

$$x_k = \arg\min_{x \in x_0 + \text{span}\{p_1, \dots, p_k\}} \left[f(x) = \|Ax - b\|_2^2 \right]$$

$$(*) \qquad p^T r_k = 0 \qquad \forall p \in \{p_1, \dots, p_k\}$$

- Statement

$$\langle r_{k-1}, p_j \rangle_A = 0 \qquad j < k-1$$

Consider

$$x_j = x_{j-1} + \alpha_j p_j$$

$$r_j = b - Ax_j = \underbrace{b - Ax_{j-1}}_{r_{j-1}} - \alpha_j A p_j$$
$$= r_{j-1} - \alpha_j A p_j$$

$$\alpha_j A p_j = r_j - r_{j-1} \qquad j < k-1$$

From the Gram-Schmidt process

$$r_j = p_{j+1} + \sum_{i=1}^{j} \frac{\langle r_j, p_i \rangle_A}{\|p_i\|_A^2} p_i \qquad \in \operatorname{span}\{p_1, \dots, p_{j+1}\}$$

Further

$$\operatorname{span}\{p_1, \dots, p_{j+1}\} \subseteq \operatorname{span}\{p_1, \dots, p_{k-1}\}$$

$$\alpha_j A p_j = r_{j-1} - r_j \in \operatorname{span}\{p_1, \dots, p_{k-1}\} \quad (**)$$

By def of α_j

$$\alpha_j = \frac{r_{j-1}^T p_j}{\|p_j\|_A^2} = \frac{1}{\|p_j\|_A^2}\left[\|r_{j-1}\|_2^2 - \sum_{i=1}^{j-1} \frac{\langle r_{j-1}, p_i \rangle_A}{\|p_i\|_A^2} \underbrace{r_{j-1}^T p_i}_{\substack{=0 \\ (*)}}\right]$$

$$= \frac{\|r_{j-1}\|_2^2}{\|p_j\|_A^2} > 0 \quad \text{because } r_j \neq 0 \quad j < k$$

$\Rightarrow \alpha_j \neq 0$; with $(**)$ we obtain

$$A p_j \in \text{span}\{p_1, \dots, p_{k-1}\}, \quad j < k-1$$

$$\langle r_{k-1}, p_j \rangle_A = r_{k-1}^T A p_j$$

$$= \langle r_{k-1}, \underbrace{A p_j}_{} \rangle_2$$

$$\left[\langle r_{k-1}, p \rangle = 0 \quad \forall p \in \text{span}\{ \quad \} \right]$$

$$= \langle r_{k-1}, p_j \rangle_A = 0 \qquad j < k-1$$

This is important because

$$p_k = r_{k-1} - \sum_{j=1}^{k-1} \frac{\langle r_{k-1}, p_j \rangle_A}{\| p_j \|_A^2} p_j$$

$$= r_{k-1} - \frac{\langle r_{k-1}, p_{k-1} \rangle_A}{\| p_{k-1} \|_A^2} p_{k-1}$$

Apply Gram-Schmidt to r_{k-1} so that we obtain p_k that is A-conjugate to $p_1, \ldots, p_{k-1}$

$$p_k = r_{k-1} - \sum_{j=1}^{k-1} \frac{\langle r_{k-1}, p_j \rangle_A}{\|p_j\|_A^2} p_j$$

We need following technical statements $\rightsquigarrow$ board:

- If $r_{k-1} = b - Ax_{k-1} \neq 0$, then there exists $p_k \in \operatorname{span}\{Ap_1, \ldots, Ap_{k-1}\}^\perp$ such that $p_k^T r_{k-1} \neq 0$ and $p_k^T r_{k-1} = p_k^T r_0$
- It then follows that (why is this helpful?)

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k} = \frac{p_k^T r_{k-1}}{p_k^T A p_k}$$

- If $r_j \neq 0$ for $j < k$, then

$$\langle r_{k-1}, p_j \rangle_A = 0\,, \qquad j < k-1\,.$$

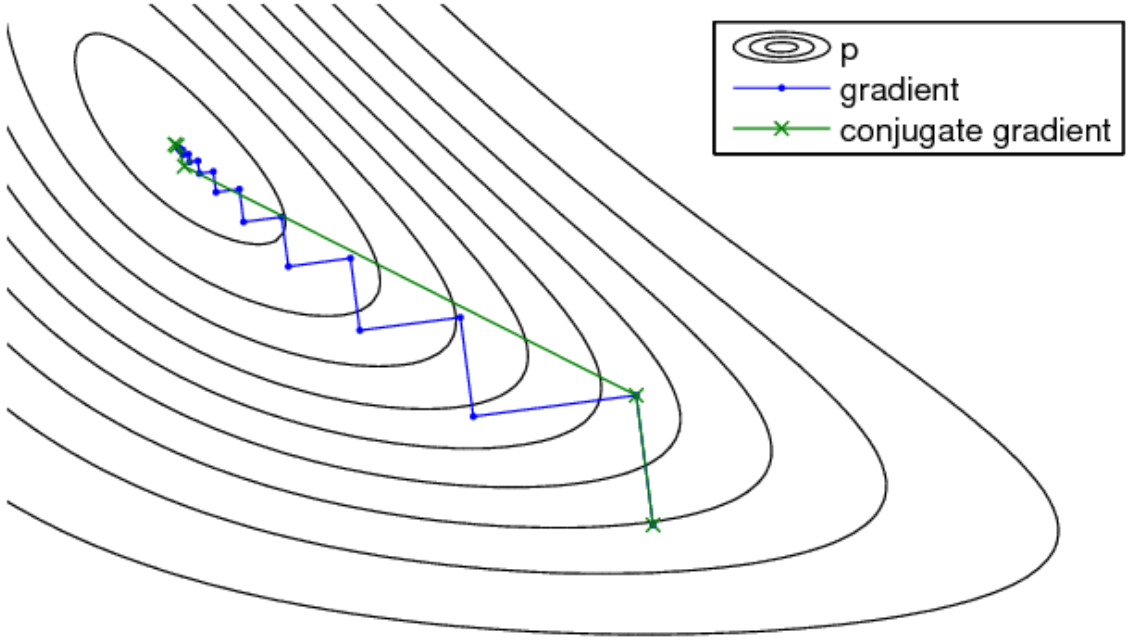
to obtain that (why is this useful?)

$$p_k = r_{k-1} - \frac{\langle r_{k-1}, p_{k-1} \rangle_A}{\|p_{k-1}\|_A^2} p_{k-1}$$

The conjugate gradient method

Choose $x_0 \in \mathbb{R}^N$ and set $p_0 = 0$. For $k = 1, 2, 3, \ldots$, stop if $r_{k-1} = b - Ax_{k-1}$ small

1. Set
$$\beta_{k-1} = \frac{\langle r_{k-1}, p_{k-1} \rangle_A}{\|p_{k-1}\|_A^2}$$
2. Set
$$p_k = r_{k-1} - \beta_{k-1} p_{k-1}$$
3. Set
$$\alpha_k = \frac{r_{k-1}^T p_k}{\|p_k\|_A^2}$$
4. Set
$$x_k = x_{k-1} + \alpha_k p_k$$
5. Set
$$r_k = b - Ax_k$$
and check for convergence



[Figure: Kuusela et al., 2009]

It can be shown that for $k \geq 1$ and $e_j \neq 0, j < k$ it holds

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \|e_0\|_A$$

for spd matrices A. $\rightsquigarrow$ Trefethen & Bau

Krylov subspace

Given an spd matrix $A \in \mathbb{R}^{N \times N}$, the Krylov subspace of order k is

$$\mathcal{K}_k(A, r_0) = \text{span}\left\{r_0, Ar_0, \ldots, A^{k-1}r_0\right\}$$

where, e.g., $r_0 = b - Ax_0$

All search directions of CG are in $\mathcal{K}_k(A, r_0)$ and all iterates $x_1, x_2, \ldots, x_k$ are in $x_0 + \mathcal{K}_k(A, r_0)$

There is a range of other methods that apply to more general matrices than spd that build on approximations in Krylov subspaces to accelerate convergence (e.g., GMRES (general residual method))

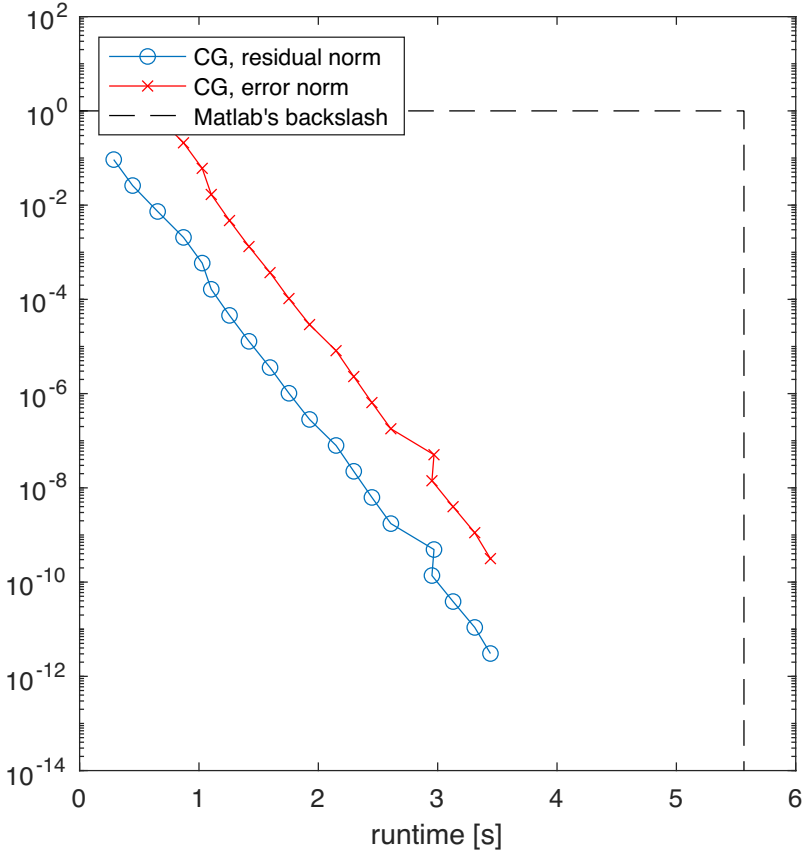
- There are also methods for finding eigenvalues via Krylov methods (Lanczos, Arnoldi iterations)
- Think of Krylov methods has having a memory of previous iterations, whereas, e.g., a power method only looks at the previous iteration (if you like stochastic processes, think of Markovian vs. non-Markovian dynamics)

Matlab implementation

```
function x = conjgrad(A, b, maxIter)

[N, ~] = size(A);
x = zeros(N, 1);
r = b - A*x;
p = r;
alpha = (r'*p)/(p'*A*p);
x = x + alpha*p;
r = b - A*x;

for i=1:maxIter
    beta = (r'*A*p)/(p'*A*p);
    p = r - beta*p;
    alpha = (r'*p)/(p'*A*p);
    x = x + alpha*p;
    r = b - A*x;
end
```

Experiment with 10000×10000 spd matrix

Condition number of this matrix is ≈ 5 (very! well conditioned)



Discussion of the CG method

- In principle, the CG algorithm is a direct solver because is converges after N steps; however, it is mostly used as an iterative method because we don't want to wait for N steps

- The convergence speed of the CG method depends on matrix properties as well. Fast convergence if the spectrum is clustered.

- However, similarly slow convergence can be expected for matrices coming from PDE discretizations and therefore preconditioning is necessary

$$Q^{-1}Ax = Q^{-1}b$$

- Preconditioned CG methods (for example multigrid can act as a preconditioner) are among the fastest solvers and achieve $\mathcal{O}(N)$ in ideal settings.

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Interpolation

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation?

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation? The best-approximation of f in $\mathcal{V}_n$ w.r.t. the induced norm is given by the projection

$$f^* = \Pi_n f \,,$$

where Π_n is the orthogonal projection onto $\mathcal{V}_n$.

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation? The best-approximation of f in $\mathcal{V}_n$ w.r.t. the induced norm is given by the projection

$$f^* = \Pi_n f \,,$$

where Π_n is the orthogonal projection onto $\mathcal{V}_n$.

How can we compute the projection?

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation? The best-approximation of f in $\mathcal{V}_n$ w.r.t. the induced norm is given by the projection

$$f^* = \Pi_n f \,,$$

where Π_n is the orthogonal projection onto $\mathcal{V}_n$.

How can we compute the projection? In many cases, we cannot directly compute the projection of f onto $\mathcal{V}_n$ because we have "too little knowledge about f" Instead?

Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation? The best-approximation of f in $\mathcal{V}_n$ w.r.t. the induced norm is given by the projection

$$f^* = \Pi_n f \,,$$

where Π_n is the orthogonal projection onto $\mathcal{V}_n$.

How can we compute the projection? In many cases, we cannot directly compute the projection of f onto $\mathcal{V}_n$ because we have "too little knowledge about f" Instead? $\rightsquigarrow$ interpolation (/regression)

Interpolation

Consider n pairs of data samples $(x_i, y_i), i = 1, \ldots, n$ with

$$y_i = f(x_i)$$

Based on $\{(x_i, y_i)\}_{i=1}^n$, we now would like to find an approximation $\tilde{f} \in \mathcal{V}_n$ that is "close" to f.

For example, we could enforce the interpolation condition, namely that it holds

$$\tilde{f}(x_i) = f(x_i)\,, \qquad i = 1, \ldots, n$$

We could also use regression $(m > n)$ and minimize, e.g.,

$$\frac{1}{m} \sum_{i=1}^{m} |y_i - \tilde{f}(x_i)|^2$$

The error of $\tilde{f}$ w.r.t. f can then typically be split into two components (we will formalize this moving forward): which?

The error of $\tilde{f}$ w.r.t. f can then typically be split into two components (we will formalize this moving forward): which?

$$\|\tilde{f} - f\| \leq \Lambda(x_1, \ldots, x_n)\|f^* - f\|$$

The projection error $\|f^* - f\|$ describes the best we can do in the space $\mathcal{V}_n$. Even if we had "full knowledge" of f so that we could compute $f^* = \Pi_n f$, we are limited by the expressiveness of the space $\mathcal{V}_n$

Intuitively, we'd also expect that the error of $\tilde{f}$ depends on the points $x_1, \ldots, x_n$ at which we have samples of f. This is captured by the "constant" $\Lambda(x_1, \ldots, x_n)$ that is independent of f but depends on $x_1, \ldots, x_n$.

Polynomial interpolation

Consider $n+1$ pairs $(x_i, y_i), i = 0, \ldots, n$ of a function f with

$$y_i = f(x_i)$$

Polynomial interpolation

Consider $n+1$ pairs $(x_i, y_i), i = 0, \ldots, n$ of a function f with

$$y_i = f(x_i)$$

Let now $\mathbb{P}_n$ be the set of all polynomials up to degree n over $\mathbb{R}$ so that we have for all $P \in \mathbb{P}_n$

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \,, \qquad a_n, \ldots, a_0 \in \mathbb{R}$$

Polynomial interpolation

Consider $n+1$ pairs $(x_i, y_i), i = 0, \ldots, n$ of a function f with

$$y_i = f(x_i)$$

Let now $\mathbb{P}_n$ be the set of all polynomials up to degree n over $\mathbb{R}$ so that we have for all $P \in \mathbb{P}_n$

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \,, \qquad a_n, \ldots, a_0 \in \mathbb{R}$$

We would like to find a $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

- The P is what $\tilde{f}$ was on the previous slide
- By saying P is a polynomial of degree n, we fixed the space $\mathcal{V}_{n+1}$ with the notation of the previous slide

Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

We sometimes refer to this unique polynomial as $P_f(\cdot | x_0, \ldots, x_n) \rightsquigarrow$ board

Given $n+1$ nodes (t_i, y_i) with pairwise distinct $t_0, \dots, t_n$. There exists a unique polynomial $P_n \in \mathbb{P}_n$ such that

$$P(t_i) = y_i, \qquad i = 0, \dots, n$$

If $P, Q \in \mathbb{P}_n$ are two interpolatory polynomials with

$$P(t_i) = Q(t_i), \qquad i = 0, \dots, n$$

then $P - Q$ is a polynomial of at most degree n and has $n+1$ roots $t_0, \dots, t_n$

$$(P - Q)(t_i) = 0$$

and thus $P - Q$ has to be the zero polynomial.

Furthermore, the space $\mathbb{P}_n$ and $\mathbb{R}^{n+1}$ have $n+1$ dimensions and the map

$$P \mapsto [P(t_0), \dots, P(t_n)]$$

is linear.

Uniqueness $\Rightarrow$ injective $\overset{\text{linear over } n+1\text{-dim space}}{\Rightarrow}$ surjectivity.

Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i\,, \qquad i = 0, \ldots, n$$

We sometimes refer to this unique polynomial as $P_f(\cdot|x_0, \ldots, x_n) \rightsquigarrow$ board

Let's try to construct $P_f(\cdot|x_0, \ldots, x_n)$. What do we need to construct $P \in \mathbb{P}_n$ for a data set $\{(x_i, y_i)\}_{i=0}^n$?

Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i\,, \qquad i = 0, \ldots, n$$

We sometimes refer to this unique polynomial as $P_f(\cdot|x_0, \ldots, x_n) \rightsquigarrow$ board

Let's try to construct $P_f(\cdot|x_0, \ldots, x_n)$. What do we need to construct $P \in \mathbb{P}_n$ for a data set $\{(x_i, y_i)\}_{i=0}^n$? A basis of $\mathbb{P}_n$.

Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i\,, \qquad i = 0, \ldots, n$$

We sometimes refer to this unique polynomial as $P_f(\cdot|x_0, \ldots, x_n) \rightsquigarrow$ board

Let's try to construct $P_f(\cdot|x_0, \ldots, x_n)$. What do we need to construct $P \in \mathbb{P}_n$ for a data set $\{(x_i, y_i)\}_{i=0}^n$? A basis of $\mathbb{P}_n$. Let's give the monomial basis $1, x, x^2, \ldots, x^n$ a chance $\rightsquigarrow$ board

poly interpolation: monomials as basis

$$P(x) = a_n x^n + \dots + a_1 x + a_0$$

We have $n+1$ unknowns

$$a_n, \dots, a_0$$

and $n+1$ equations

$$P(x_0) = y_0$$
$$\vdots$$
$$P(x_n) = y_n$$

Linear system

$$\underbrace{\begin{bmatrix} x_0^n & x_0^{n-1} & \dots & x_0 & 1 \\ \vdots & & & & \vdots \\ x_n^n & x_n^{n-1} & \dots & x_n & 1 \end{bmatrix}}_{(n+1)\times(n+1)} \underbrace{\begin{bmatrix} a_n \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}}_{n+1} = \underbrace{\begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix}}_{n+1}$$

$$= V_n$$

It can be shown that

$$\det(V_n) = \prod_{j=0}^{n} \prod_{j=i+1}^{n} (x_i - x_j)$$

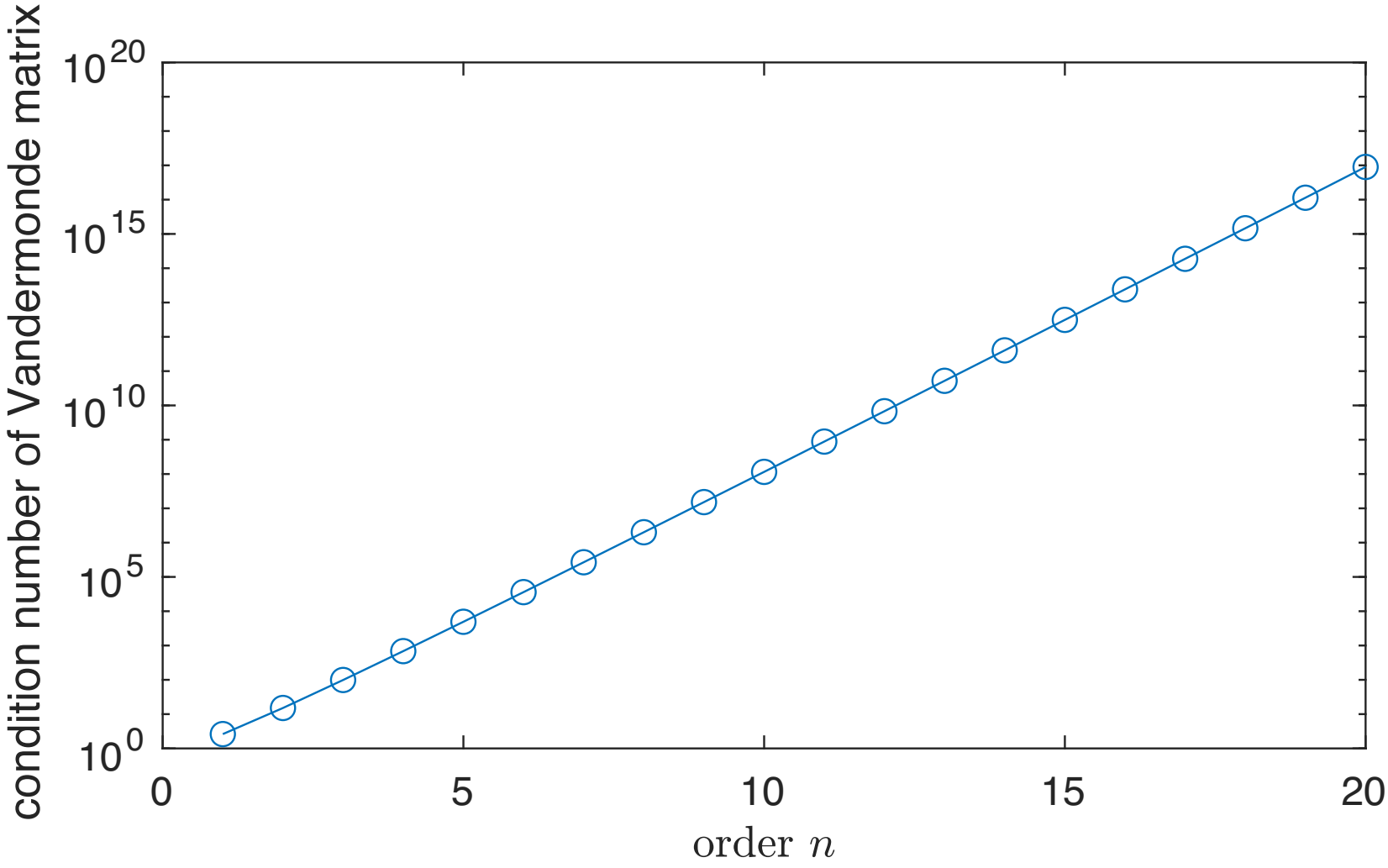
det is non-zero iff $x_i \neq x_j$ for $i \neq j$

$\Rightarrow$ agrees with assumption that nodes $x_0, \dots, x_n$ distinct

What could go wrong?

↳ condition number of V_n is high

↳ Do we really need to invest $\mathcal{O}(n^3)$ costs to compute $P_p(\cdot \mid x_0, \dots, x_n)$?

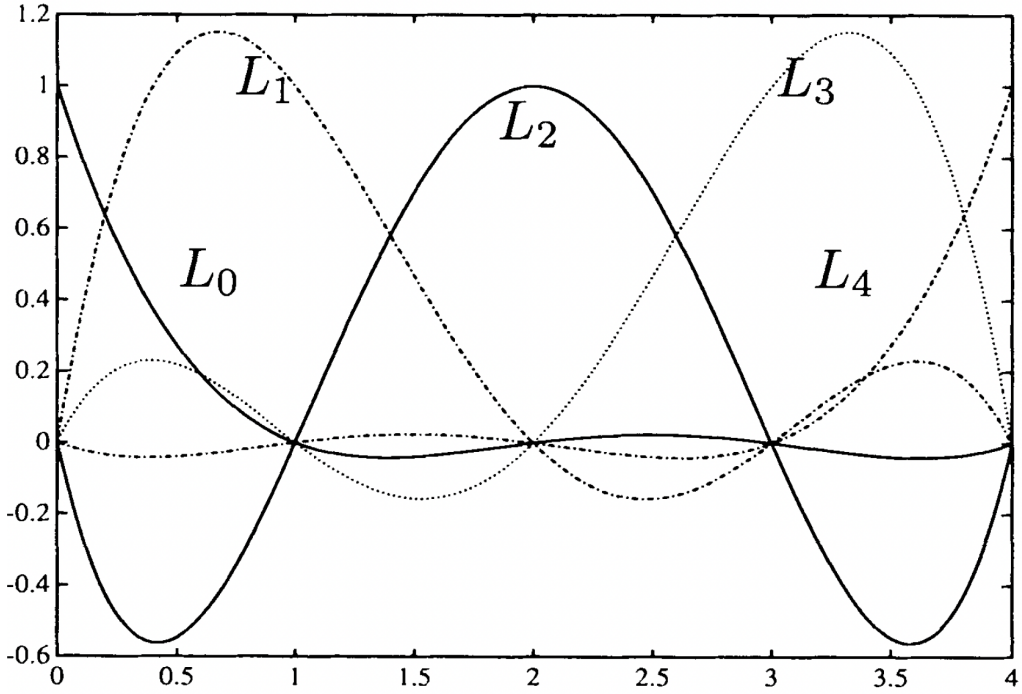


$\rightsquigarrow$ we really should look for another basis

Lagrange basis

The Lagrange polynomials $L_0, \ldots, L_n \in \mathbb{P}_n$ are uniquely defined for distinct $x_0, \ldots, x_n$

$$L_i(x_j) = \delta_{ij}\,, \qquad L_i \in \mathbb{P}_n\,.$$



Lagrange polynomials up to order $n = 4$ for equidistant $x_0, \ldots, x_4$. [Figure: Deuflhard]

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free. Drawback?

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free. Drawback? but the cost of evaluating the polynomial is too high for practical computations

The Lagrange polynomials are orthogonal w.r.t. the following inner product over $\mathbb{P}_n$

$$\langle P, Q \rangle = \sum_{i=0}^{n} P(x_i) Q(x_i) \,, \qquad P, Q \in \mathbb{P}_n$$

Let's try to generalize this to other scalar products to find other orthogonal bases

Orthogonal polynomials

Define an inner product between functions:

$$(f, g) = \int_a^b \omega(x) f(x) g(x)\, dx,$$

where $\omega(x) > 0$ for $a \leq x \leq b$ is a weight function. The induced norm is $\|f\| := \sqrt{(f, f)}$.

Let $P_0, P_1, P_2, \ldots, P_K$ be polynomials of $0, 1, 2, \ldots, K$ order, respectively. They are called orthogonal polynomials on $[a, b]$ with respect to the weight function $\omega(x)$ if it holds

$$(P_i, P_j) = \int_a^b \omega(x) P_i(x) P_j(x) \mathrm{d}x = \delta_{ij} \gamma_i \,, \qquad i, j = 0, \ldots, K \,,$$

with $\gamma_i = \|P_i\|^2 > 0$.

To define orthogonal polynomials uniquely, we normalize them so that the leading coefficient is one, i.e.,

$$P_k(x) = x^k + \dots$$

Theorem: There exist uniquely determined orthogonal polynomials $P_k \in \mathbb{P}_k$ with leading coefficient 1. These polynomials satisfy the 3-term recurrence relation:

$$P_k(x) = (x + a_k)P_{k-1}(x) + b_k P_{k-2}(x), \quad k = 2, 3, \dots$$

with starting values $P_0 = 1$, $P_1 = x + a_1$, where

$$a_k = -\frac{(xP_{k-1}, P_{k-1})}{(P_{k-1}, P_{k-1})}, \quad b_k = -\frac{(P_{k-1}, P_{k-1})}{(P_{k-2}, P_{k-2})}$$

Proof: $\rightsquigarrow$ board

3-term recurrence; show this by induction

↳ only polynomial of degree 0 with leading coefficient 1 is

$$P_0 = 1 \quad \in \mathbb{P}_0$$

↳ $P_1 = x + a_1$; it is unique because a_1 is unique is computed as

$$a_1 = -\frac{(xP_0, P_0)}{(P_0, P_0)}$$

and $(P_0, P_1) = 0$

↳ Suppose $P_0, \dots, P_{k-1}$ have been constructed

↳ P_j has degree j

↳ P_i, P_j are orthogonal if $i \neq j$ w.r.t. $(\cdot,\cdot)$

↳ leading coefficient is 1

If $P_k \in \mathbb{P}_k$ is a polynomial of degree k and normalized (leading coefficient is 1), then

$$\underbrace{P_k}_{x^k + \dots} - \underbrace{x P_{k-1}}_{x(x^{k-1} + \dots)}$$

is of degree $\leq k-1$

the $P_0, \dots, P_{k-1}$ form orthogonal basis of $\mathbb{P}_{k-1}$ w.r.t. weighted inner product so that

$$P_k - x P_{k-1} = \sum_{j=1}^{k-1} c_j P_j$$

with coefficients

$$\underline{c_j} = \frac{(P_k - x P_{k-1}, P_j)}{(P_j, P_j)} \Bigg\} \text{ projection of } P_k - x P_{k-1} \text{ onto basis for } P_j$$

If P_k orthogonal to $P_0, \dots, P_{k-1}$, then

$$(P_k, P_j) = 0, \qquad j = 0, \dots, k-1$$

and thus

$$c_j = -\frac{(x P_{k-1}, P_j)}{(P_j, P_j)} = -\frac{\int w(x)(x P_{k-1}(x) P_j(x))}{(P_j, P_j)}$$

$$= -\frac{(P_{k-1}, x P_j)}{(P_j, P_j)}$$

We obtain

$$c_{k-1} = -\frac{(xP_{k-1}, P_{k-1})}{(P_{k-1}, P_{k-1})}.$$

$$c_{k-2} = -\frac{(P_{k-1}, xP_{k-2})}{(P_{k-2}, P_{k-2})} \qquad \begin{cases} P_{k-1} = (x + a_{k-1})P_{k-2} + b_k P_{k-3} \\ xP_{k-2} = P_{k-1} - a_{k-1}P_{k-2} - b_k P_{k-3} \end{cases}$$

$$= -\frac{(P_{k-1}, P_{k-1} - a_{k-1}P_{k-2} - b_k P_{k-3})}{(P_{k-2}, P_{k-2})}$$

$$= -\frac{(P_{k-1}, P_{k-1})}{(P_{k-2}, P_{k-2})}$$

$c_0 = \dots = c_{k-3} = 0$ because $P_0, \dots, P_{k-1}$ are orthogonal

$$P_k - xP_{k-1} = c_{k-1}P_{k-1} + c_{k-2}P_{k-2}$$

$$P_k = (x + \underbrace{c_{k-1}}_{a_k})P_{k-1} + \underbrace{c_{k-2}}_{b_k}P_{k-2}$$

Numerics with 3-term recurrence

- 3-term recurrence can be used to compute polynomials P_k completely, or
- evaluate P_k at a point x_0 via pre-computing the a's and b's
- However, simple application of 3-term recurrence might not always be stable due to cancellation (coefficients a_k, b_k can be negative)
- Cancellation errors can be avoided with clever numerics (Sec 6.3 in Deuflhard)

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Polynomial interpolation

Today

- Interpolation

Announcements

- Homework 5 is due Mon, Nov 18 before class

Recap: Function approximation

Consider a function $f \in \mathcal{V}$ in a function space $\mathcal{V}$. Let now $\phi_1, \ldots, \phi_n$ be a basis of an n-dimensional space $\mathcal{V}_n$.

The task that we are interested in is finding a function $f^* \in \mathcal{V}_n$ that approximates f with coefficients $c_1, \ldots, c_n$:

$$f^*(x) = \sum_{i=1}^{n} c_i \phi_i(x) \,.$$

If we have an inner product, what is the best approximation? The best-approximation of f in $\mathcal{V}_n$ w.r.t. the induced norm is given by the projection

$$f^* = \Pi_n f \,,$$

where Π_n is the orthogonal projection onto $\mathcal{V}_n$.

How can we compute the projection? In many cases, we cannot directly compute the projection of f onto $\mathcal{V}_n$ because we have "too little knowledge about f" Instead? $\rightsquigarrow$ interpolation (/regression)

Recap: Interpolation

Consider n pairs of data samples $(x_i, y_i), i = 1, \ldots, n$ with

$$y_i = f(x_i)$$

Based on $\{(x_i, y_i)\}_{i=1}^n$, we now would like to find an approximation $\tilde{f} \in \mathcal{V}_n$ that is "close" to f.

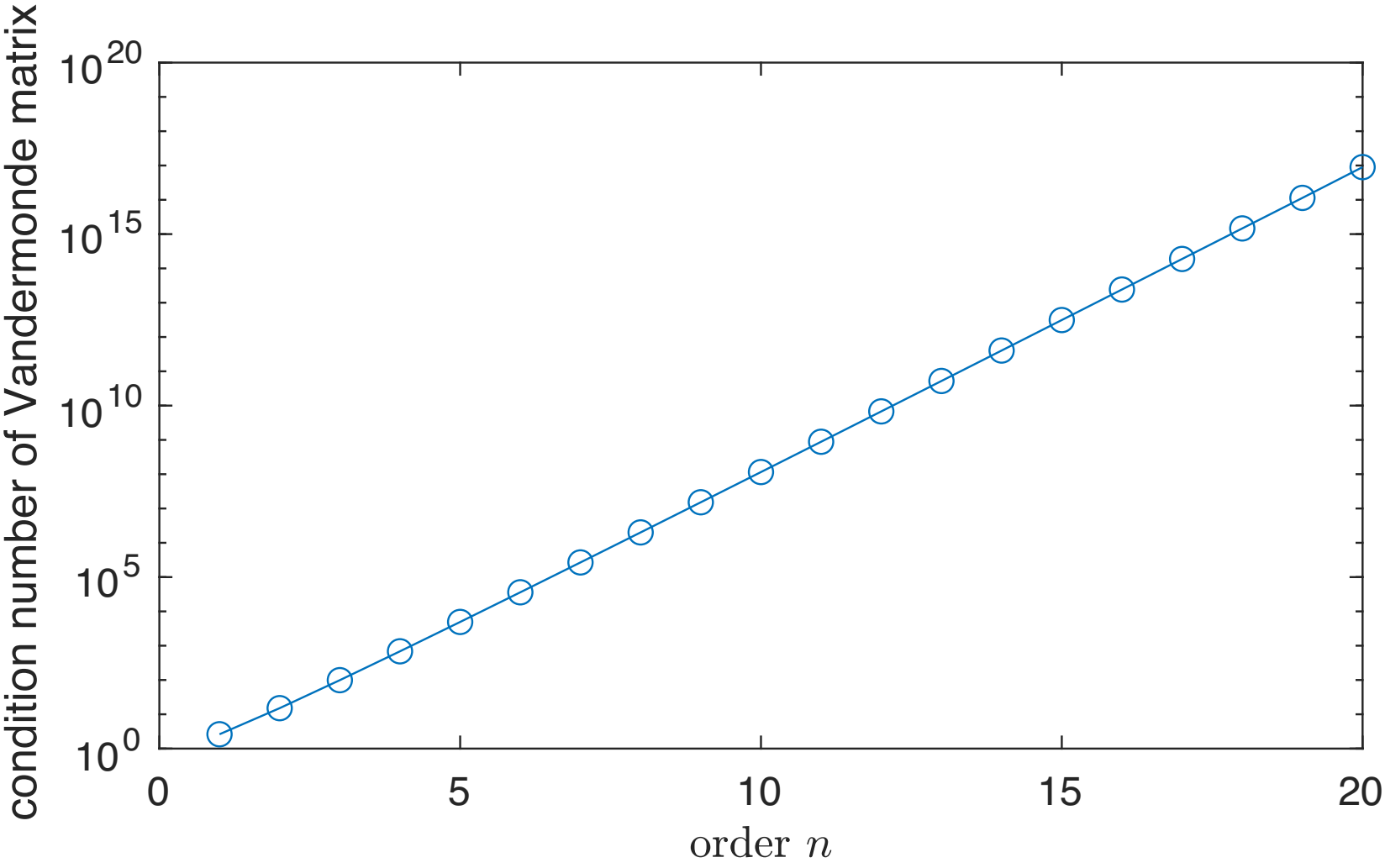
For example, we could enforce the interpolation condition, namely that it holds

$$\tilde{f}(x_i) = f(x_i)\,, \qquad i = 1, \ldots, n$$

We could also use regression ($m > n$) and minimize, e.g.,

$$\frac{1}{m} \sum_{i=1}^{m} |y_i - \tilde{f}(x_i)|^2$$

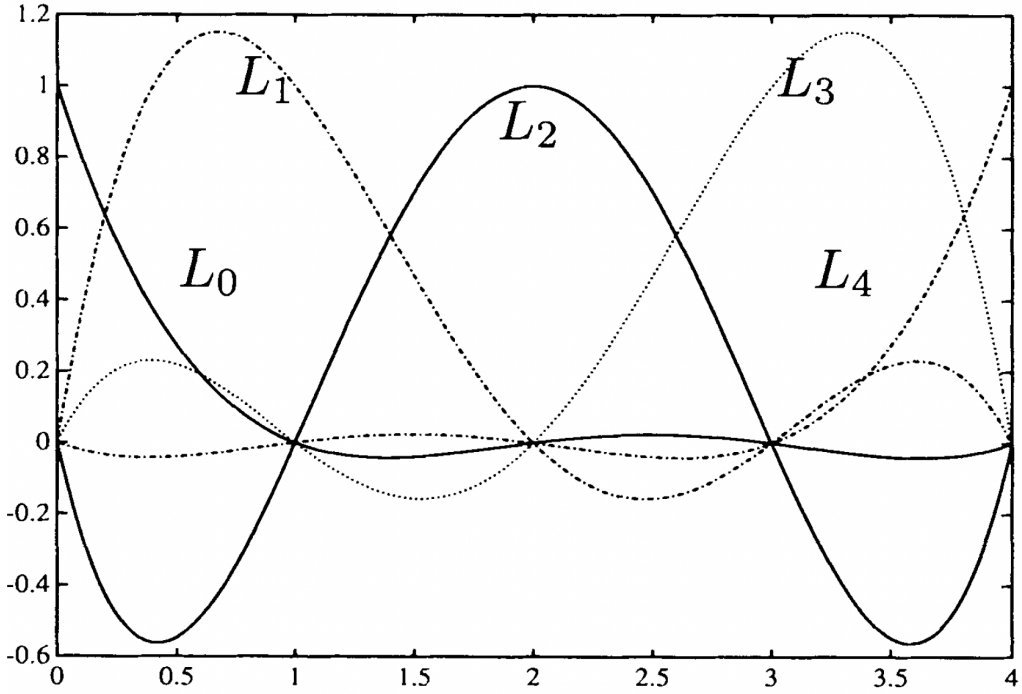
Condition number of Vandermonde matrix



Recap: Lagrange basis

The Lagrange polynomials $L_0, \ldots, L_n \in \mathbb{P}_n$ are uniquely defined for distinct $x_0, \ldots, x_n$

$$L_i(x_j) = \delta_{ij}\,, \qquad L_i \in \mathbb{P}_n\,.$$



Lagrange polynomials up to order $n = 4$ for equidistant $x_0, \ldots, x_4$. [Figure: Deuflhard]

Recap: Orthogonal polynomials

Define an inner product between functions:

$$(f, g) = \int_a^b \omega(x) f(x) g(x)\, dx,$$

where $\omega(x) > 0$ for $a \leq x \leq b$ is a weight function. The induced norm is $\|f\| := \sqrt{(f, f)}$.

Let $P_0, P_1, P_2, \ldots, P_K$ be polynomials of $0, 1, 2, \ldots, K$ order, respectively. They are called orthogonal polynomials on $[a, b]$ with respect to the weight function $\omega(x)$ if it holds

$$(P_i, P_j) = \int_a^b \omega(x) P_i(x) P_j(x) \mathrm{d}x = \delta_{ij} \gamma_i \,, \qquad i, j = 0, \ldots, K \,,$$

with $\gamma_i = \|P_i\|^2 > 0$.

Recap

To define orthogonal polynomials uniquely, we normalize them so that the leading coefficient is one, i.e.,

$$P_k(x) = x^k + \dots$$

Theorem: There exist uniquely determined orthogonal polynomials $P_k \in \mathbb{P}_k$ with leading coefficient 1. These polynomials satisfy the 3-term recurrence relation:

$$P_k(x) = (x + a_k)P_{k-1}(x) + b_k P_{k-2}(x), \quad k = 2, 3, \dots$$

with starting values $P_0 = 1$, $P_1 = x + a_1$, where

$$a_k = -\frac{(xP_{k-1}, P_{k-1})}{(P_{k-1}, P_{k-1})}, \quad b_k = -\frac{(P_{k-1}, P_{k-1})}{(P_{k-2}, P_{k-2})}$$

Chebyshev polynomials

Chebyshev polynomials for $-1 \leq x \leq 1$ are given by the 3-term recurrence: $T_0(x) = 1$, $T_1(x) = x$,

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \quad \text{for } k \geq 2.$$

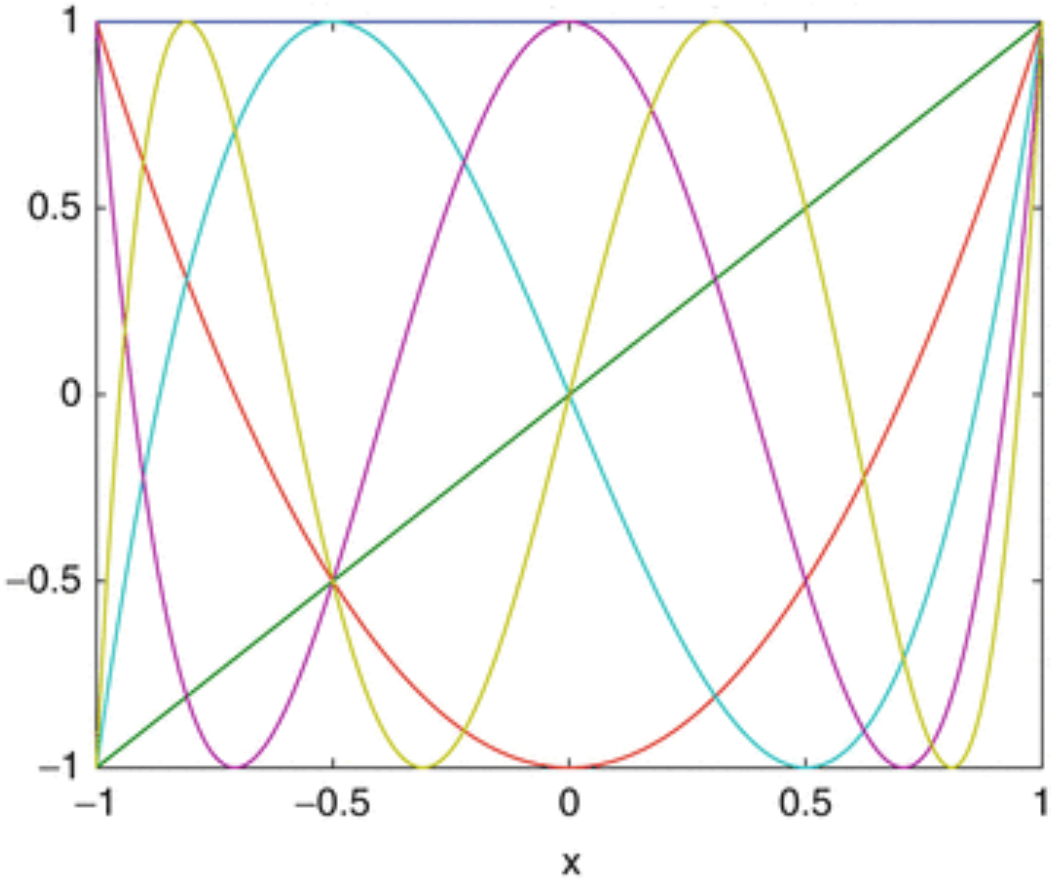
Chebyshev polynomials are orthogonal in the following inner product

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_n(x) T_m(x)\, dx = \begin{cases} 0 & n \neq m \\ \pi & n = m = 0 \\ \pi/2 & n = m \neq 0 \end{cases}$$

The roots of Chebyshev polynomials play an important role in interpolation and numerical quadrature.

Chebyshev polynomials are also given by $T_k(x) = \cos(k \arccos(x))$ with roots

$$x_i = \cos\left(\frac{\pi(i - 1/2)}{k}\right)$$



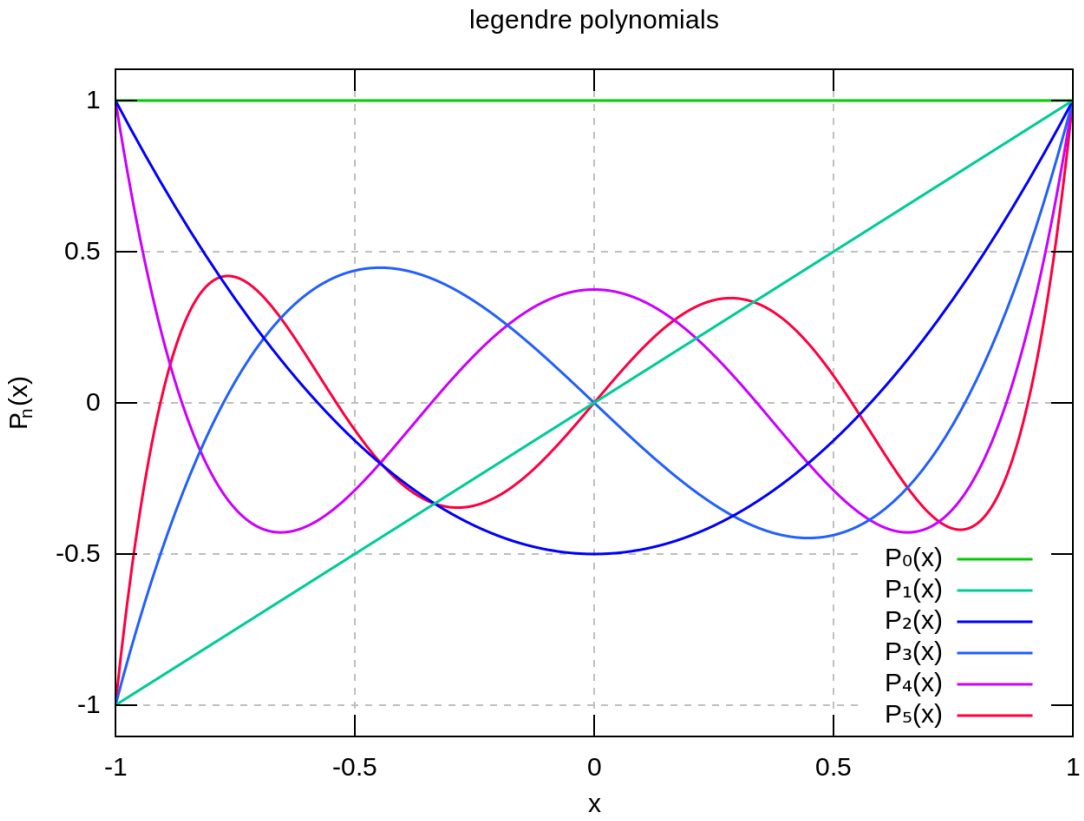
Source: Springer, Encyclopedia of Applied and Computational Mathematics.

Question: What property of the roots of the polynomials do you observe?

Legendre polynomials

Orthogonal polynomials for weight function $\omega \equiv 1$, satisfy $L_0 = 1$, $L_1 = x$, and

$$L_{k+1}(x) = \frac{2k+1}{k+1} x L_k(x) - \frac{k}{k+1} L_{k-1(x)}$$



Approximation error of polynomial interpolation

Let $f : I \to \mathbb{R}$ and let $x_0, x_1, \ldots, x_n \in I$ be $n+1$ distinct nodes. Assume $f \in C^{n+1}(I)$. Then the interpolation error at point $x \in I$ is

$$E_n(x) = f(x) - P_f(x|x_0, \ldots, x_n) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x)\,,$$

where $\xi \in I$ and

$$\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)\,,$$

is the nodal polynomial.

Proof: $\rightsquigarrow$ textbook

What are observations can we make?

Approximation error of polynomial interpolation

Let $f : I \to \mathbb{R}$ and let $x_0, x_1, \ldots, x_n \in I$ be $n+1$ distinct nodes. Assume $f \in C^{n+1}(I)$. Then the interpolation error at point $x \in I$ is

$$E_n(x) = f(x) - P_f(x|x_0, \ldots, x_n) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x)\,,$$

where $\xi \in I$ and

$$\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)\,,$$

is the nodal polynomial.

Proof: $\rightsquigarrow$ textbook

What are observations can we make?

- The error bound requires increasing smoothness with the degree $n \rightsquigarrow$ can we derive bounds for $f \in C^k$ with k fixed independent of degree n?
- The bound critically depends on $\omega_{n+1} \rightsquigarrow$ contains information about the nodes

Let's say the function f is continuous, i.e., $f \in C^0([a,b])$, but not even differentiable.

Define the maximum norm

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|$$

Given points $X = \{x_0, \ldots, x_n\} \subset [a,b]$, define the interpolation error

$$E_{n,\infty}(X) = \|f - P_f(\cdot|X)\|_\infty$$

and the best-approximation error with $f^* \in \mathbb{P}_n$

$$E_n^* = \|f - f^*\|_\infty \leq \|f - \tilde{f}\|_\infty \,, \qquad \forall \tilde{f} \in \mathbb{P}_n$$

Side remark: What does the Stone-Weierstrass theorem tell us about the best-approximation of continuous functions with polynomials?

Let's say the function f is continuous, i.e., $f \in C^0([a,b])$, but not even differentiable.

Define the maximum norm

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|$$

Given points $X = \{x_0, \ldots, x_n\} \subset [a,b]$, define the interpolation error

$$E_{n,\infty}(X) = \|f - P_f(\cdot|X)\|_\infty$$

and the best-approximation error with $f^* \in \mathbb{P}_n$

$$E_n^* = \|f - f^*\|_\infty \leq \|f - \tilde{f}\|_\infty \,, \qquad \forall \tilde{f} \in \mathbb{P}_n$$

Side remark: What does the Stone-Weierstrass theorem tell us about the best-approximation of continuous functions with polynomials? Universal approximation: Uniform convergence (i.e., converge in $\|\cdot\|_\infty$). However, the sequence of polynomials that converges is not necessarily obtained via interpolation.

Theorem Let $f \in C^0([a,b])$ and $X = \{x_0, \ldots, x_n\} \subset [a,b]$. Then

$$E_{n,\infty}(X) = E_n^*(1 + \Lambda_n(X)),$$

where $\Lambda_n(X)$ is the *Lebesgue* constant of X, defined as

$$\Lambda_n(X) = \left\| \sum_{j=0}^{n} |L_j^{(n)}| \right\|_\infty ,$$

where $L_j^{(n)} \in \mathbb{P}_n$ is the j-th Lagrange polynomial with

$$L_j^{(n)}(x_i) = \begin{cases} 1, & i = j, \\ 0, & \text{otherwise} \end{cases}$$

Notice the decomposition of the error into component E_n^* (independent of X but dependent on f) and $\Lambda_n(X)$ (independent of f but dependent on X)

We control the best-approximation error E_n^* with the space $\mathbb{P}_n$

We control the Lebesgue constant $\Lambda_n(X)$ with the nodes $x_0, \ldots, x_n$

What are two important questions to ask about the Lebesgue constant $\Lambda_n(X)$?

*Not necessarily nested

We control the best-approximation error E_n^* with the space $\mathbb{P}_n$

We control the Lebesgue constant $\Lambda_n(X)$ with the nodes $x_0, \ldots, x_n$

What are two important questions to ask about the Lebesgue constant $\Lambda_n(X)$?

- What are "good" nodes $x_0, \ldots, x_n$ that keep $\Lambda_n(X)$ low or even minimize it?
- In the best case, how does $\Lambda_n(X)$ behave with respect to $n \to \infty$

*Not necessarily nested

We control the best-approximation error E_n^* with the space $\mathbb{P}_n$

We control the Lebesgue constant $\Lambda_n(X)$ with the nodes $x_0, \ldots, x_n$

What are two important questions to ask about the Lebesgue constant $\Lambda_n(X)$?

- What are "good" nodes $x_0, \ldots, x_n$ that keep $\Lambda_n(X)$ low or even minimize it?
- In the best case, how does $\Lambda_n(X)$ behave with respect to $n \to \infty$

Famous result by Faber (1914): Given a sequence of any nodes*
$X_n = \{x_{n,0}, x_{n,1}, \ldots, x_{n,n}\} \subset [a, b]$, then there always exists a continuous function f so that $P_f(\cdot|x_0, \ldots, x_n)$ does not converge to f in $\|\cdot\|_\infty$ for $n \to \infty$

Thus, polynomial interpolation does not allow for approximating any continuous function

*Not necessarily nested

We control the best-approximation error E_n^* with the space $\mathbb{P}_n$

We control the Lebesgue constant $\Lambda_n(X)$ with the nodes $x_0, \ldots, x_n$

What are two important questions to ask about the Lebesgue constant $\Lambda_n(X)$?

- What are "good" nodes $x_0, \ldots, x_n$ that keep $\Lambda_n(X)$ low or even minimize it?
- In the best case, how does $\Lambda_n(X)$ behave with respect to $n \to \infty$

Famous result by Faber (1914): Given a sequence of any nodes*
$X_n = \{x_{n,0}, x_{n,1}, \ldots, x_{n,n}\} \subset [a, b]$, then there always exists a continuous function f so that $P_f(\cdot|x_0, \ldots, x_n)$ does not converge to f in $\|\cdot\|_\infty$ for $n \to \infty$

Thus, polynomial interpolation does not allow for approximating any continuous function

However, interpolation works fantastic for "most" functions $\rightsquigarrow$ "Six myths of polynomial interpolation and quadrature," Trefethen, Approximation Theory and Approximation Practice and also chebfun.org

*Not necessarily nested

It also has been shown that for any possible choice $X_n = \{x_{n,0}, \ldots, x_{n,n}\}$, there exists a constant $C > 0$ such that

$$\Lambda_n(X_n) > \frac{2}{\pi} \log(n+1) - C\,, \qquad n = 0, 1, \ldots$$

Thus $\Lambda_n(X_n) \to \infty$ for $n \to \infty$

- This is an instability statement in the sense that "investing more time" (by selecting more grid points), leads to a larger Lebesgue constant
- However, the Lebesgue constant might grow very slowly with n (even slower than E_n^* decreases!)

The Lebesgue constant is also the absolute condition number of polynomial interpolation on $[a, b]$ with points X

$$\kappa_{\text{abs}} = \Lambda_n(X)$$

$\rightsquigarrow$ textbook by Deuflhard

Let's approximate

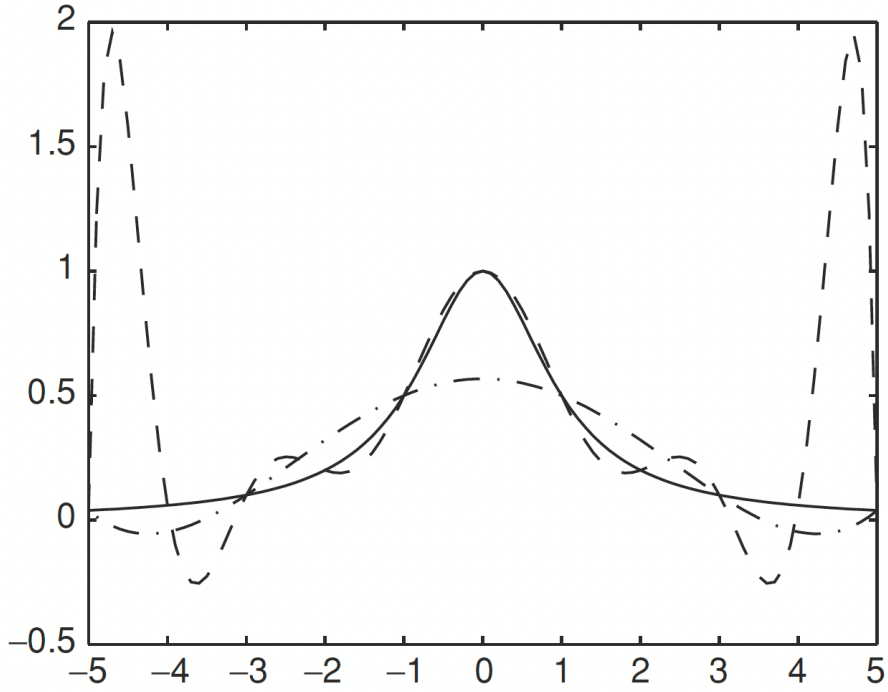
$$f(x) = \frac{1}{1+x^2}\,, \qquad -5 \le x \le 5\,,$$

using polynomial interpolation on equally spaced nodes in $[-5, 5]$.

Let's approximate

$$f(x) = \frac{1}{1+x^2}\,, \qquad -5 \le x \le 5\,,$$

using polynomial interpolation on equally spaced nodes in $[-5, 5]$.



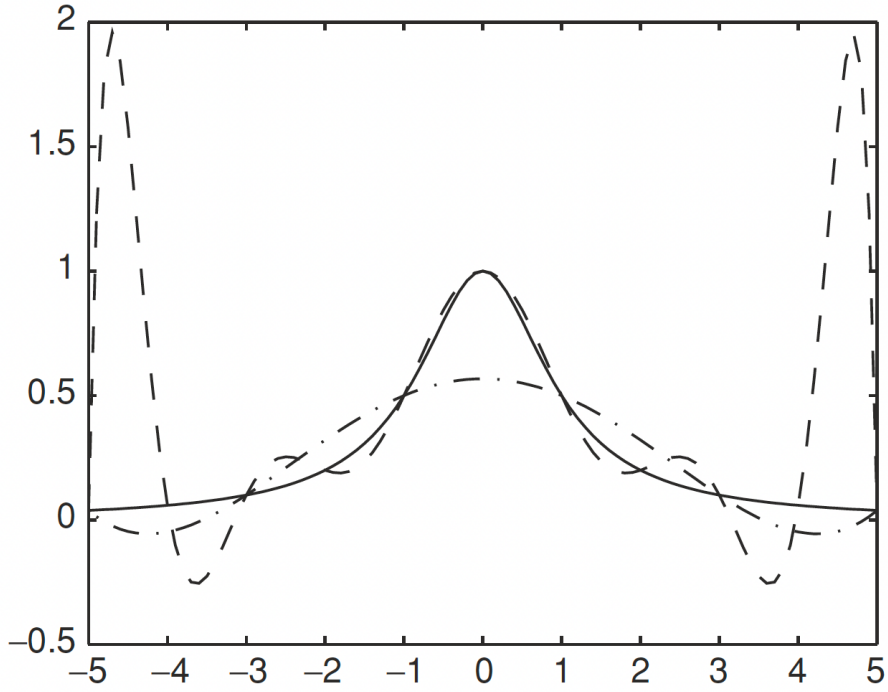
[Figure: Quarteroni]

Interpolants of degree $n = 5$ and $n = 10$ of $f(x) = 1/(1+x^2)$ on equidistant nodes.

Let's approximate

$$f(x) = \frac{1}{1+x^2}\,, \qquad -5 \le x \le 5\,,$$

using polynomial interpolation on equally spaced nodes in $[-5, 5]$.

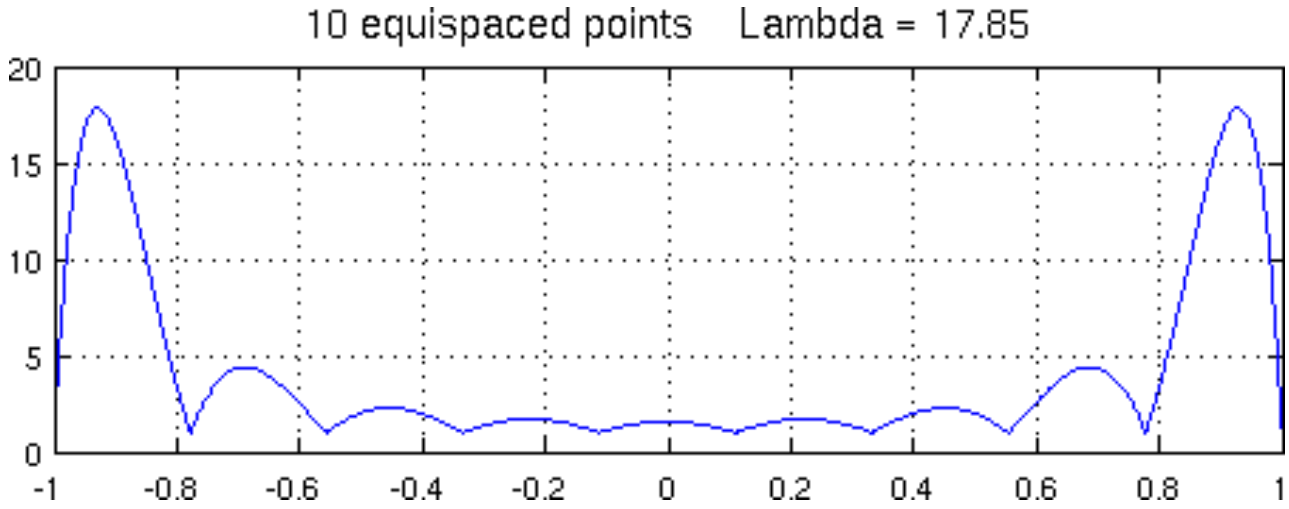


[Figure: Quarteroni]

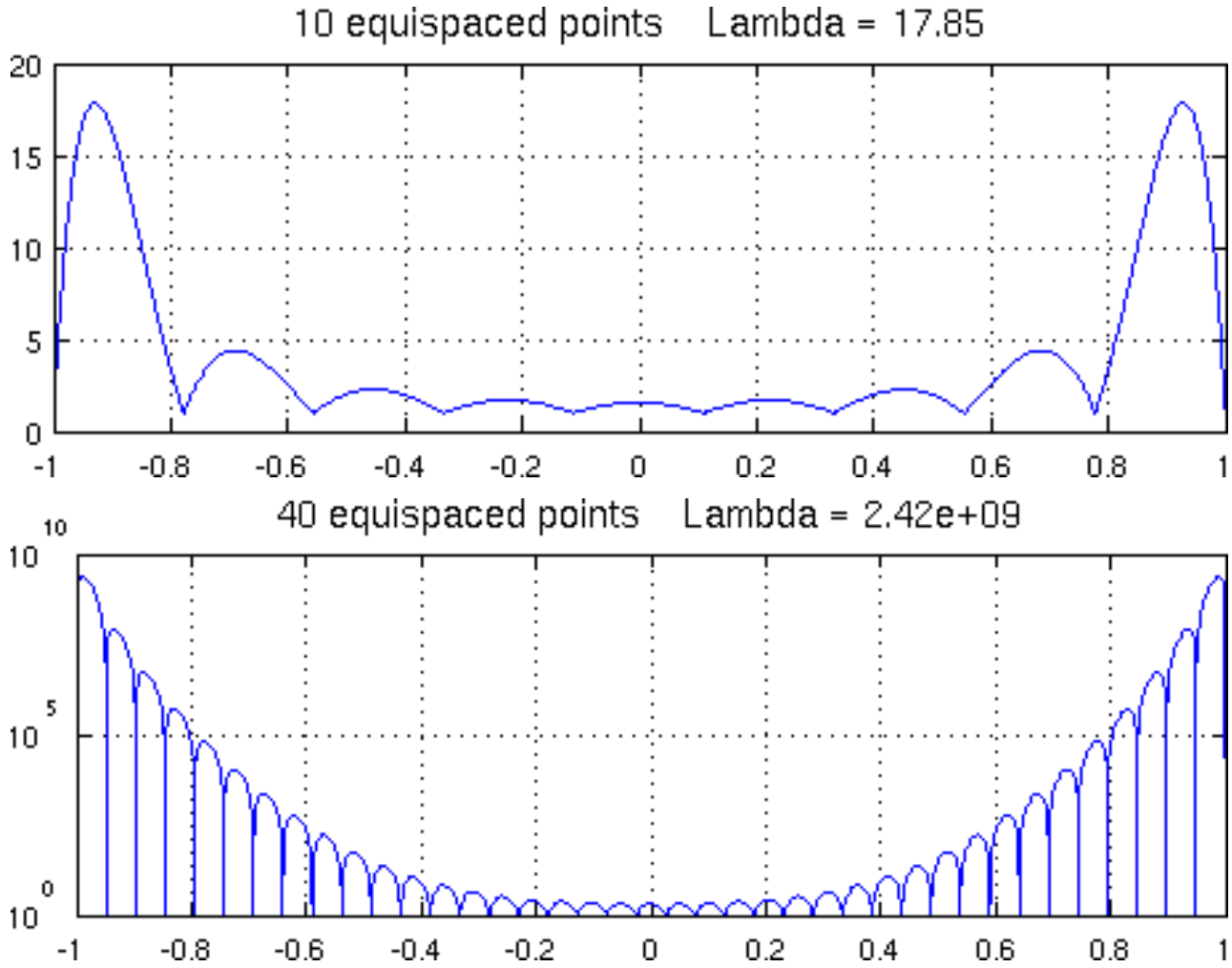
Interpolants of degree $n = 5$ and $n = 10$ of $f(x) = 1/(1+x^2)$ on equidistant nodes. It can be shown that polynomial interpolation does not converge for $|x| > 4$ for this f

It is a very common situation that interpolation on equidistant nodes leads to high oscillations near the interval ends $\rightsquigarrow$ Runge's phenomenon

For equidistant nodes, the Lebesgue constant grows dramatically near the interval ends

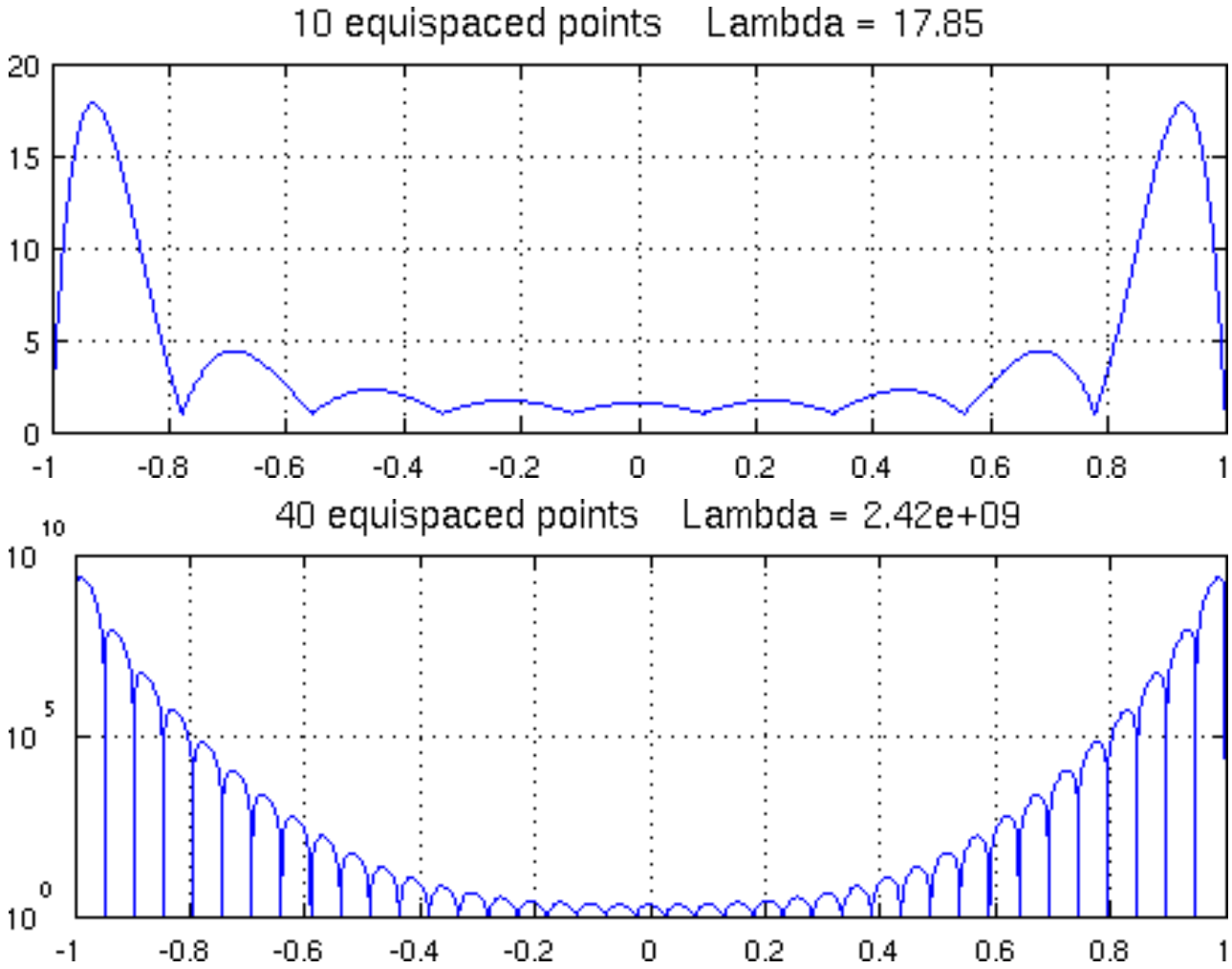


For equidistant nodes, the Lebesgue constant grows dramatically near the interval ends



What is a way out of this?

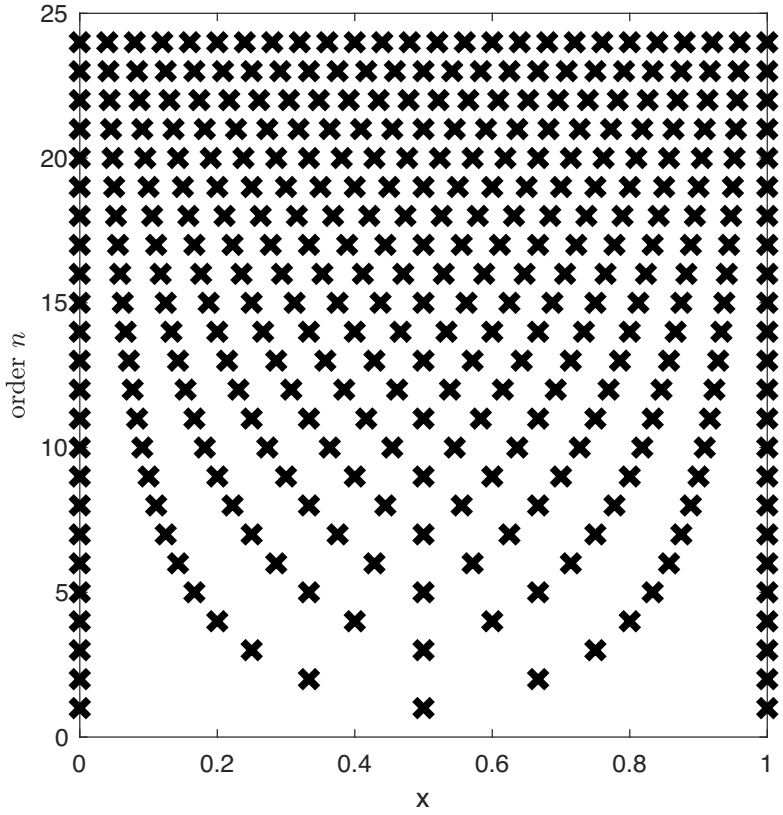
For equidistant nodes, the Lebesgue constant grows dramatically near the interval ends



What is a way out of this? $\rightsquigarrow$ Use different nodes

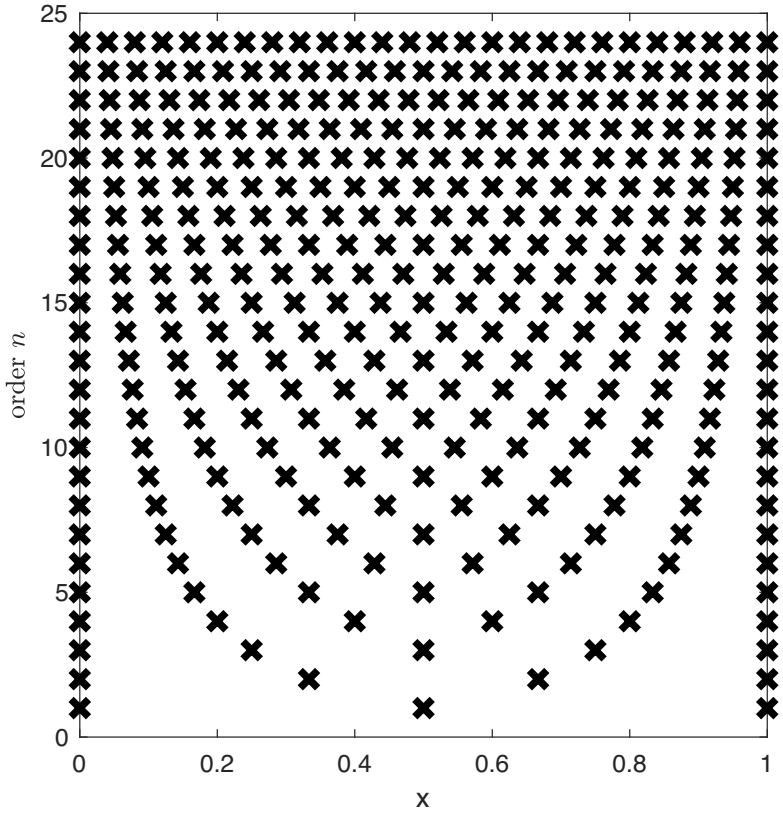
Non-equispaced points

equispaced

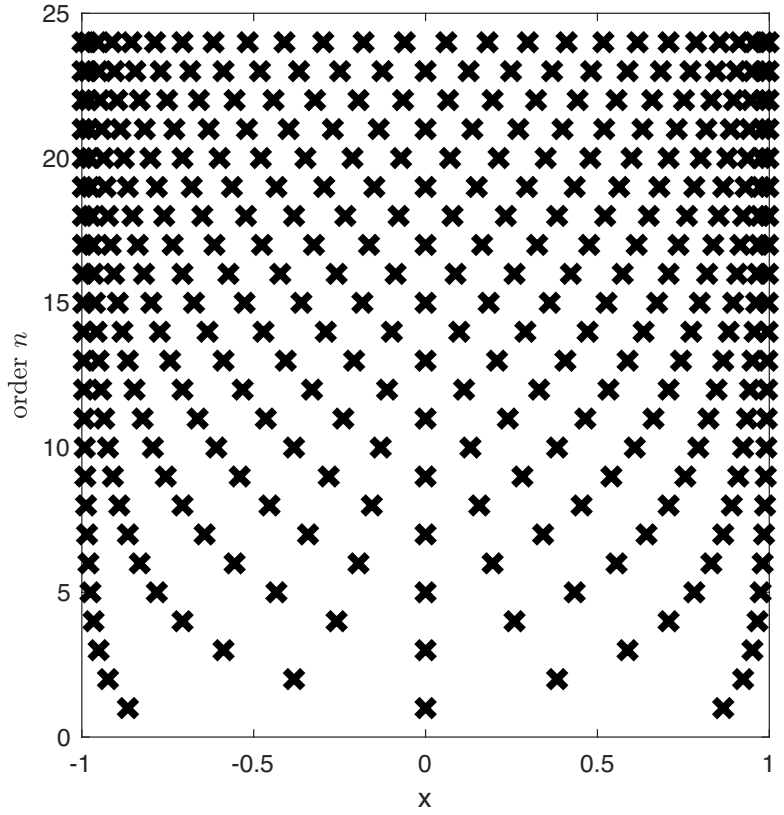


Non-equispaced points

equispaced



Chebyshev nodes



Lebesgue constants for different orders:

n	Λ_n for equidistant nodes	Λ_n for Chebyshev nodes
5	3.106292	2.104398
10	29.890695	2.489430
15	512.052451	2.727778
20	10986.533993	2.900825

Chebyshev nodes are the roots of the Chebyshev polynomials:

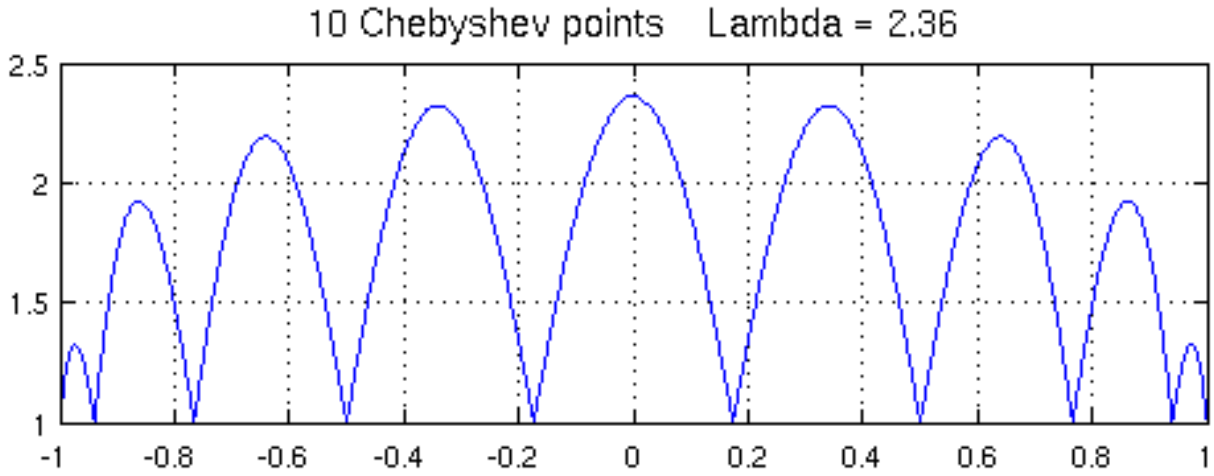
$$t_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \text{ for } i = 0, \ldots, n$$

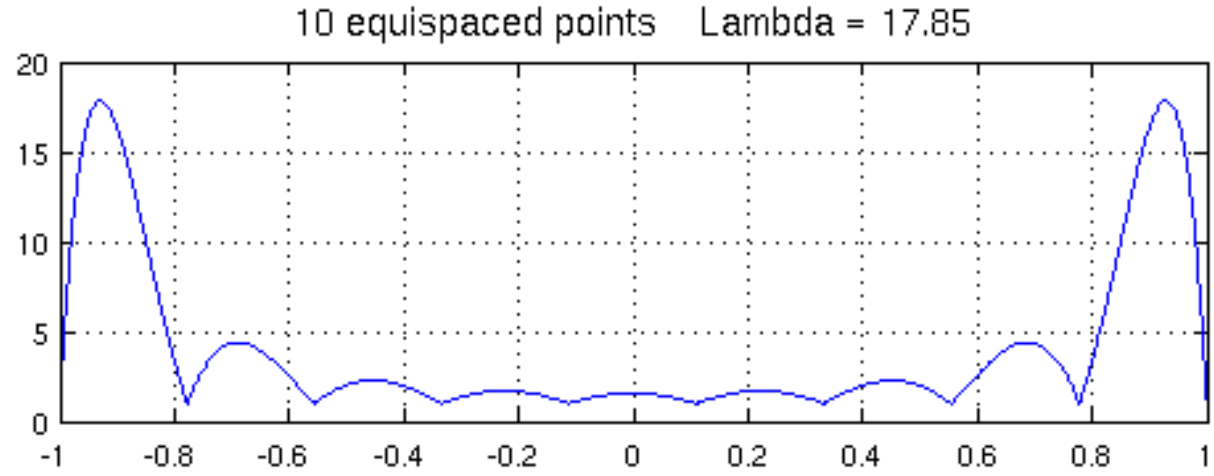
Lebesgue constant for Chebyshev nodes is bounded as

$$\Lambda_n \leq \frac{2}{\pi}\log(n+1) + 1\,,$$

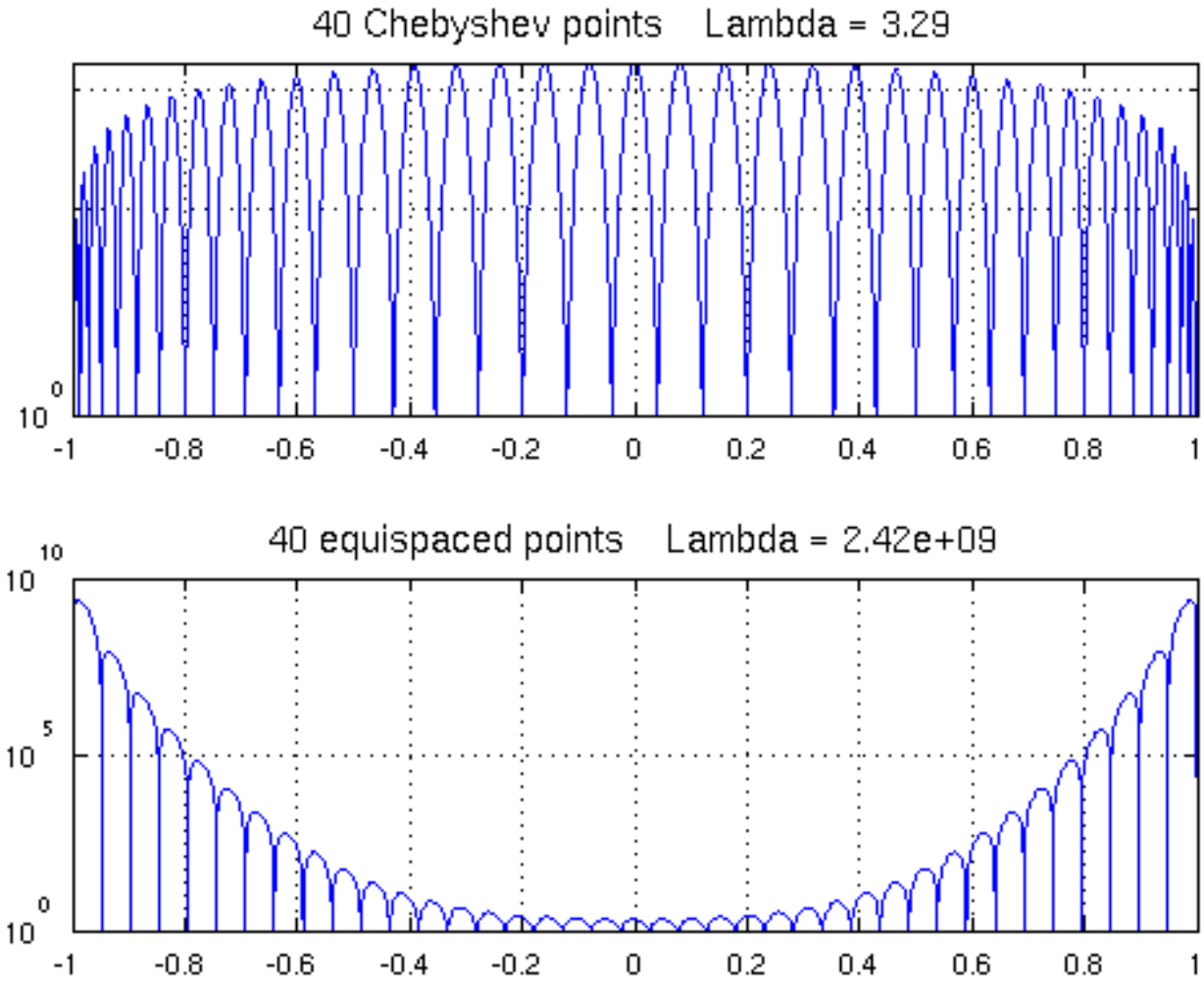
which is close to the lower bound from previous slides

Lebesgue constant for $n = 10$, uniform vs. Chebyshev nodes:





Lebesgue constant for $n = 40$, uniform vs. Chebyshev nodes:



The Lebesgue constant for Chebyshev points is bounded as

$$\Lambda_n \leq \frac{2}{\pi} \log(n+1) + 1\,,$$

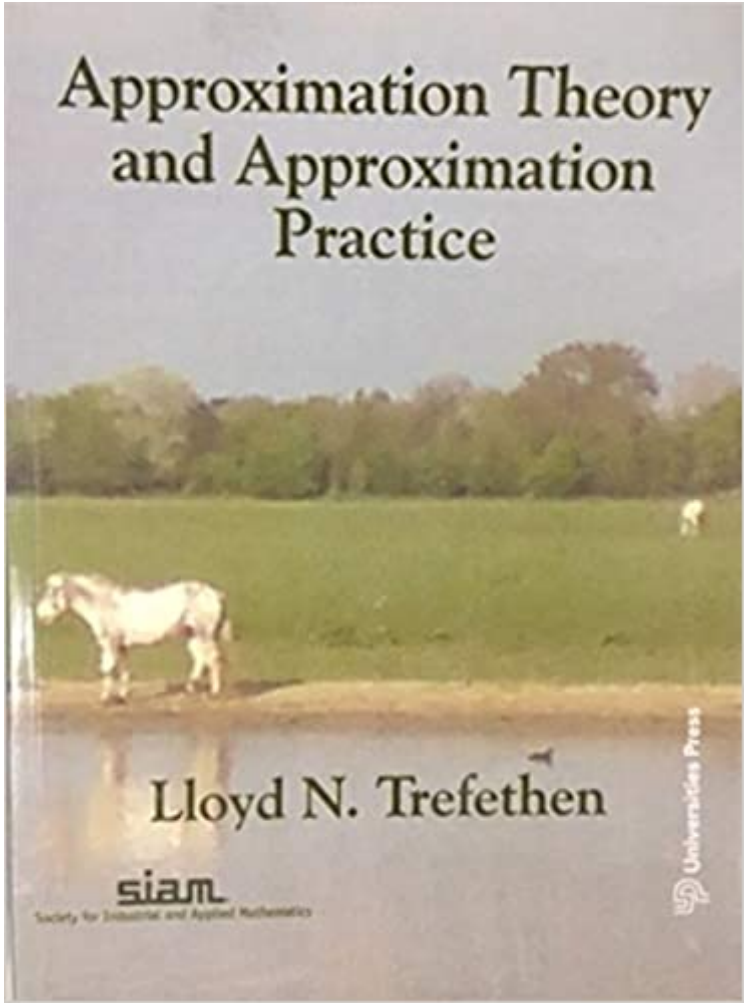
We know that for all interpolation points there exists a continuous function f such that polynomial interpolation does not converge (result by Faber, see previous slides)

This is also true for Cheybshev points! However, if f is Lipschitz continuous, then one can show uniform convergence.

The smoother the function (e.g., continuously differentiable, ν-times continuously differentiable, analytic), the faster interpolation on Chebyshev nodes converges uniformly.

This is a general principle: The more regularity there is in the function, the easier we can approximate it (faster convergence w.r.t. number of degrees of freedom of the approximation)

Beyond the material we cover here...



Pointwise evaluation of interpolating polynomials

Aitken Lemma: The interpolating polynomial $P_f(\cdot|x_0, \ldots, x_n)$ satisfies the recurrence relation

$$P_f(x|x_0, \ldots, x_n) = \frac{(x_0 - x)P_f(x|x_1, \ldots, x_n) - (x_n - x)P_f(x|x_0, \ldots, x_{n-1})}{x_0 - x_n}$$

Proof $\rightsquigarrow$ board

Aitken's lemma

$$\varphi(x) = \frac{(x_0 - x)\, P_f(x \mid x_1, \dots, x_n) - (x_n - x)\, P_f(x \mid x_0, \dots, x_{n-1})}{x_0 - x_n}$$

We now show that φ interpolates f at $x_0, \dots, x_n$. Because we have uniqueness of interpolating polynomials, it follows $\varphi = P_f(\cdot \mid x_0, \dots, x_n)$

for $i = \underline{1}, \dots, \underline{n-1}$

$$\varphi(x_i) = \frac{(x_0 - x_i)\, f(x_i) - (x_n - x_i)\, f(x_i)}{x_0 - x_n}$$

$$= \frac{x_0 - x_i - x_n + x_i}{x_0 - x_n}\, f(x_i) = f(x_i)$$

$$\varphi(x_0) = \frac{0 - (x_n - x_0)\, f(x_0)}{x_0 - x_n} = f(x_0)$$

$$\varphi(x_n) = \frac{(x_0 - x_n)\, f(x_n)}{x_0 - x_n} - 0 = f(x_n)$$

$$\varphi = P_f(\cdot \mid x_0, \dots, x_n)$$

Pointwise evaluation of interpolating polynomials

Aitken Lemma: The interpolating polynomial $P_f(\cdot|x_0,\dots,x_n)$ satisfies the recurrence relation

$$P_f(x|x_0,\dots,x_n) = \frac{(x_0-x)P_f(x|x_1,\dots,x_n) - (x_n-x)P_f(x|x_0,\dots,x_{n-1})}{x_0-x_n}$$

Proof $\rightsquigarrow$ board

Introduce the notation

$$P_{ik} = P_f(x|x_{i-k},\dots,x_i)\,, \qquad i \geq k$$

then $P_f(x|x_0,\dots,x_n) = P_{nn}$ can be computed based on the Neville scheme

$$\begin{aligned} P_{i0} =& f(x_i)\,, \quad i = 0,\dots,n \\ P_{ik} =& P_{i,k-1} + \frac{x-x_i}{x_i-x_{i-k}}(P_{i,k-1} - P_{i-1,k-1})\,, \quad i \geq k \end{aligned}$$

$\rightsquigarrow$ a numerically stable and efficient ($\mathcal{O}(n^2)$) way to evaluate (!) $P_f(x|x_0,\dots,x_n)$

Hermite interpolation

Given

$$a = x_0 \leq x_1 \leq \ldots \leq x_n = b$$

with possibly duplicated nodes. Less information than degrees of freedom?

Hermite interpolation

Given

$$a = x_0 \leq x_1 \leq \ldots \leq x_n = b$$

with possibly duplicated nodes. Less information than degrees of freedom? Therefore, if the node x_i occurs k times, the corresponding node values correspond to $f(x_i), f'(x_i), \ldots, f^{k-1}(x_i)$.

The Hermite interpolation polynomial $p(x)$ is a polynomial of order n, which coincides with the nodal values (and, for duplicated nodes, derivatives at nodal values) at the nodes.

Hermite interpolation

Given

$$a = x_0 \leq x_1 \leq \ldots \leq x_n = b$$

with possibly duplicated nodes. Less information than degrees of freedom? Therefore, if the node x_i occurs k times, the corresponding node values correspond to $f(x_i), f'(x_i), \ldots, f^{k-1}(x_i)$.

The Hermite interpolation polynomial $p(x)$ is a polynomial of order n, which coincides with the nodal values (and, for duplicated nodes, derivatives at nodal values) at the nodes.

Aitken lemma for Hermite interpolation: The (Hermite) interpolating polynomial $P = P_f(\cdot|x_0, \ldots, x_n)$ satisfies, if $x_i \neq x_j$, the recurrence relation:

$$P_f(x|x_0, \ldots, x_n) = \frac{(x_i - x)P_f(x|x_0, \ldots, \hat{x}_i, \ldots, x_n) - (x_j - x)P_f(x|x_0, \ldots, \hat{x}_j, \ldots, x_n)}{x_i - x_j},$$

where the hat symbol indicates that the corresponding node is omitted.

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

Newton polynomial basis

The leading coefficient a_n of the interpolation polynomial (monomial basis!)

$$P_f(x|x_0, \ldots, x_n) = a_n x^n + \cdots + a_0$$

is called the *n-th divided difference*, $[x_0, \ldots, x_n]f := a_n$.

The divided differences are the coefficients $c_0, \ldots, c_n$: The interpolation polynomial $P_f(\cdot|x_0, \ldots, x_n)$ for $x_0 \leq x_1 \leq \cdots \leq x_n$ (not necessarily distinct and thus need $f \in C^{n+1}$) is given by

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i]f \, \omega_i(x).$$

Furthermore,

$$f(x) = P(x) + [x_0, \ldots, x_n, x]f \, \omega_{n+1}(x).$$

Proof $\rightsquigarrow$ board

Newton differences

↳ for $n=0$, $p(x) = a_0$

$$p(x_0) = a_0 = [x_0]f = f(x_0)$$

$$w_0(x) = 1$$

↳ $n>0$ Let

$$P_{n-1} = \sum_{i=0}^{n-1} [x_0, \dots, x_i]f \; w_i(x)$$

interpolates $x_0, \dots, x_{n-1}$

$$P_n = [x_0, \dots, x_n]f \, x^n + a_{n-1} x^{n-1} + \dots + a_0$$

as

$$P_n(x) = [x_0, \dots, x_n]f \, w_n(x) + Q_{n-1}(x)$$

with some polynomial $Q_{n-1} \in \mathbb{P}_{n-1}$ of degree $n-1$

Notice that

$$w_n(x) = \prod_{i=0}^{n-1} (x_i - x)$$

$$w_n(x_i) = 0, \quad i = 0, \dots, n-1$$

$$Q_{n-1}(x_i) = P_n(x_i) - [x_0, \dots, x_n]f \, w_n(x_i)$$
$$= f(x_i)$$

Q_{n-1} interpolates $x_0, \dots, x_{n-1}$

$\Rightarrow$

$$Q_{n-1} = \sum_{i=0}^{n-1} [x_0, \dots, x_i] f \, w_i$$

$$P_n(x) = [x_0, \dots, x_n] f \, w_n(x) + \sum_{i=0}^{n-1} [x_0, \dots, x_i] f \, w_i(x)$$

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Polynomial interpolation

Today

- Interpolation

Announcements

- Homework 5 is due Mon, Nov 18 before class

Recap

Let's say the function f is continuous, i.e., $f \in C^0([a,b])$, but not even differentiable.

Define the maximum norm

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)|$$

Given points $X = \{x_0, \ldots, x_n\} \subset [a,b]$, define the interpolation error

$$E_{n,\infty}(X) = \|f - P_f(\cdot|X)\|_\infty$$

and the best-approximation error with $f^* \in \mathbb{P}_n$

$$E_n^* = \|f - f^*\|_\infty \leq \|f - \tilde{f}\|_\infty \,, \qquad \forall \tilde{f} \in \mathbb{P}_n$$

Recap

Theorem Let $f \in C^0([a,b])$ and $X = \{x_0, \ldots, x_n\} \subset [a,b]$. Then

$$E_{n,\infty}(X) = E_n^*(1 + \Lambda_n(X)),$$

where $\Lambda_n(X)$ is the *Lebesgue* constant of X, defined as

$$\Lambda_n(X) = \left\| \sum_{j=0}^{n} |L_j^{(n)}| \right\|_\infty ,$$

where $L_j^{(n)} \in \mathbb{P}_n$ is the j-th Lagrange polynomial with

$$L_j^{(n)}(x_i) = \begin{cases} 1, & i = j, \\ 0, & \text{otherwise} \end{cases}$$

Notice the decomposition of the error into component E_n^* (independent of X but dependent on f) and $\Lambda_n(X)$ (independent of f but dependent on X)

Recap: Pointwise evaluation of interpolating polynomials

Aitken Lemma: The interpolating polynomial $P_f(\cdot|x_0,\dots,x_n)$ satisfies the recurrence relation

$$P_f(x|x_0,\dots,x_n) = \frac{(x_0-x)P_f(x|x_1,\dots,x_n) - (x_n-x)P_f(x|x_0,\dots,x_{n-1})}{x_0-x_n}$$

Introduce the notation

$$P_{ik} = P_f(x|x_{i-k},\dots,x_i)\,, \qquad i \geq k$$

then $P_f(x|x_0,\dots,x_n) = P_{nn}$ can be computed based on the Neville scheme

$$\begin{aligned} P_{i0} =& f(x_i)\,, \quad i = 0,\dots,n \\ P_{ik} =& P_{i,k-1} + \frac{x-x_i}{x_i-x_{i-k}}(P_{i,k-1} - P_{i-1,k-1})\,, \quad i \geq k \end{aligned}$$

$\rightsquigarrow$ a numerically stable and efficient ($\mathcal{O}(n^2)$) way to evaluate (!) $P_f(x|x_0,\dots,x_n)$

Recap: Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

Recap: Newton polynomial basis

The leading coefficient a_n of the interpolation polynomial (monomial basis!)

$$P_f(x|x_0, \ldots, x_n) = a_n x^n + \cdots + a_0$$

is called the *n-th divided difference*, $[x_0, \ldots, x_n]f := a_n$.

The divided differences are the coefficients $c_0, \ldots, c_n$: The interpolation polynomial $P_f(\cdot|x_0, \ldots, x_n)$ for $x_0 \leq x_1 \leq \cdots \leq x_n$ (not necessarily distinct and thus need $f \in C^{n+1}$) is given by

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i]f \, \omega_i(x).$$

Furthermore,

$$f(x) = P(x) + [x_0, \ldots, x_n, x]f \, \omega_{n+1}(x).$$

Proof $\rightsquigarrow$ board

cont of proof:

$$P_n = [x_0, \dots, x_n] f \, \omega_n + \sum_{i=1}^{n-1} [x_0, \dots, x_i] f \, \omega_i$$

$$P_{n+1} = \sum_{i=1}^{n} [x_0, \dots, x_i] f \, \omega_i + [x_0, \dots, x_n, x_{n+1}] f \, \omega_{n+1}$$

this is true for any x_{n+1} and $P_{n+1}(x_{n+1}) = f(x_{n+1})$

$$\Rightarrow \quad f(x) = P_n(x) + [x_0, \dots, x_n, \underline{x}] f \, \omega_{n+1}(x)$$

Divided differences

The divided differences $[x_0, \ldots, x_n]f$ satisfy the following properties:

- $[x_0, \ldots, x_n]P = 0$ for all $P \in \mathbb{P}_{n-1}$ (because $a_n = 0$ for degree $n-1$ polynom)
- If $x_0 = \ldots = x_n$:

$$[x_0, \ldots, x_n]f = \frac{f^{(n)}(x_0)}{n!}$$

- The following recurrence relation holds for $x_i \neq x_j$:

$$[x_0, \ldots, x_n]f = \frac{([x_0, \ldots, \hat{x}_i, \ldots, x_n]f - [x_0, \ldots, \hat{x}_j, \ldots, x_n]f)}{x_j - x_i}$$

- $[x_0, \ldots, x_n]f = \frac{1}{n!} f^{(n)}(\tau)$ with a $a \leq \tau \leq b$, if f in C^{n+1}

Divided differences

Let us use divided differences to compute the coefficients for the Newton basis for the cubic interpolation polynomial p that satisfies $p(0) = 1$, $p(0.5) = 2$, $p(1) = 0$, $p(2) = 3$.

$p(\underline{0}) = 1, \quad p(\underline{0.5}) = 2, \quad p(\underline{1}) = 0, \quad p(\underline{2}) = 3$

x_i		
0	$[x_0]f = \underline{1}$	
$\frac{1}{2}$	$[x_1]f = 2$	$[x_0, x_1]f = \dfrac{[x_1]f - [x_0]f}{x_1 - x_0} = \dfrac{2-1}{\frac{1}{2}-0} = \underline{\underline{2}}$
1	$[x_2]f = 0$	$[x_1\ x_2]f = \dfrac{[x_2]f - [x_1]f}{x_2 - x_1} = \dfrac{0-2}{1-\frac{1}{2}} = \underline{\underline{-4}}$
2	$[x_3]f = 3$	

$$[x_0 x_1 x_2]f = \frac{[x_1 x_2]f - [x_0 x_1]f}{x_2 - x_0} = \frac{-4-2}{1-0} = \underline{\underline{-6}}$$

$$\vdots$$

$$[x_0 x_1 x_2 x_3]f = \underline{\underline{\frac{16}{3}}}$$

$$p(x) = \sum_{i=0}^{3} c_i \, \omega_i(x) \qquad \omega_i(x) = \prod_{j=0}^{i-1} (x - x_j)$$

$$= \underline{c_0} \cdot 1 + \underline{c_1}(x-0) + \underline{c_2}(x-0)(x-\tfrac{1}{2})$$

$$+ \underline{c_3}(x-0)(x-\tfrac{1}{2})(x-1)$$

$$= 1 + 2x - 6x(x-\tfrac{1}{2}) + \frac{16}{3}x(x-\tfrac{1}{2})(x-1)$$

Divided differences

Let us use divided differences to compute the coefficients for the Newton basis for the cubic interpolation polynomial p that satisfies $p(0) = 1$, $p(0.5) = 2$, $p(1) = 0$, $p(2) = 3$.

x_i				
0	$[x_0]f = 1$			
0.5	$[x_1]f = 2$	$[x_0x_1]f = \frac{[x_1]f-[x_0]f}{x_1-x_0} = 2$		
1	$[x_2]f = 0$	$[x_1x_2]f = \frac{[x_2]f-[x_1]f}{x_2-x_1} = -4$	$[x_0x_1x_2]f = -6$	
2	$[x_3]f = 3$	$[x_2x_3]f = \frac{[x_3]f-[x_2]f}{x_3-x_2} = 3$	$[x_1x_2x_3]f = \frac{14}{3}$	$\frac{16}{3}$

Thus, the interpolating polynomial is

$$p(x) = 1 + 2x + (-6)x(x-0.5) + \frac{16}{3}x(x-0.5)(x-1).$$

Divided differences

Let us now use divided differences to compute the coefficients for the Newton basis for the cubic interpolation polynomial p that satisfies $p(0) = 1$, $p'(0) = 2$, $p''(0) = 1$, $p(1) = 3$.

x_i				
0	$[x_0]f = 1$			
0	$[x_0]f = 1$	$[x_0x_1]f = p'(0) = 2$		
0	$[x_0]f = 1$	$[x_1x_2]f = p'(0) = 2$	$[x_0x_1x_2]f = \frac{p''(0)}{2!} = \frac{1}{2}$	
1	$[x_3]f = 3$	$[x_2x_3]f = \frac{[x_3]f - [x_0]f}{x_3 - x_0} = 2$	$[x_1x_2x_3]f = 0$	$[x_0x_1x_2x_3]f = -\frac{1}{2}$

Thus, the interpolating polynomial is

$$p(t) = 1 + 2t + \frac{1}{2}t^2 + (-\frac{1}{2})t^3$$

Polynomial interpolation

- Polynomial interpolation
- Hermite interpolation
- (Least squares with polynomials)
- What else?

Polynomial interpolation

- Polynomial interpolation
- Hermite interpolation
- (Least squares with polynomials)
- What else? Splines, i.e., piecewise polynomial interpolation

Splines

Assume $(l+2)$ pairwise disjoint nodes:

$$a = x_0 < x_1 < \ldots < x_{l+1} = b.$$

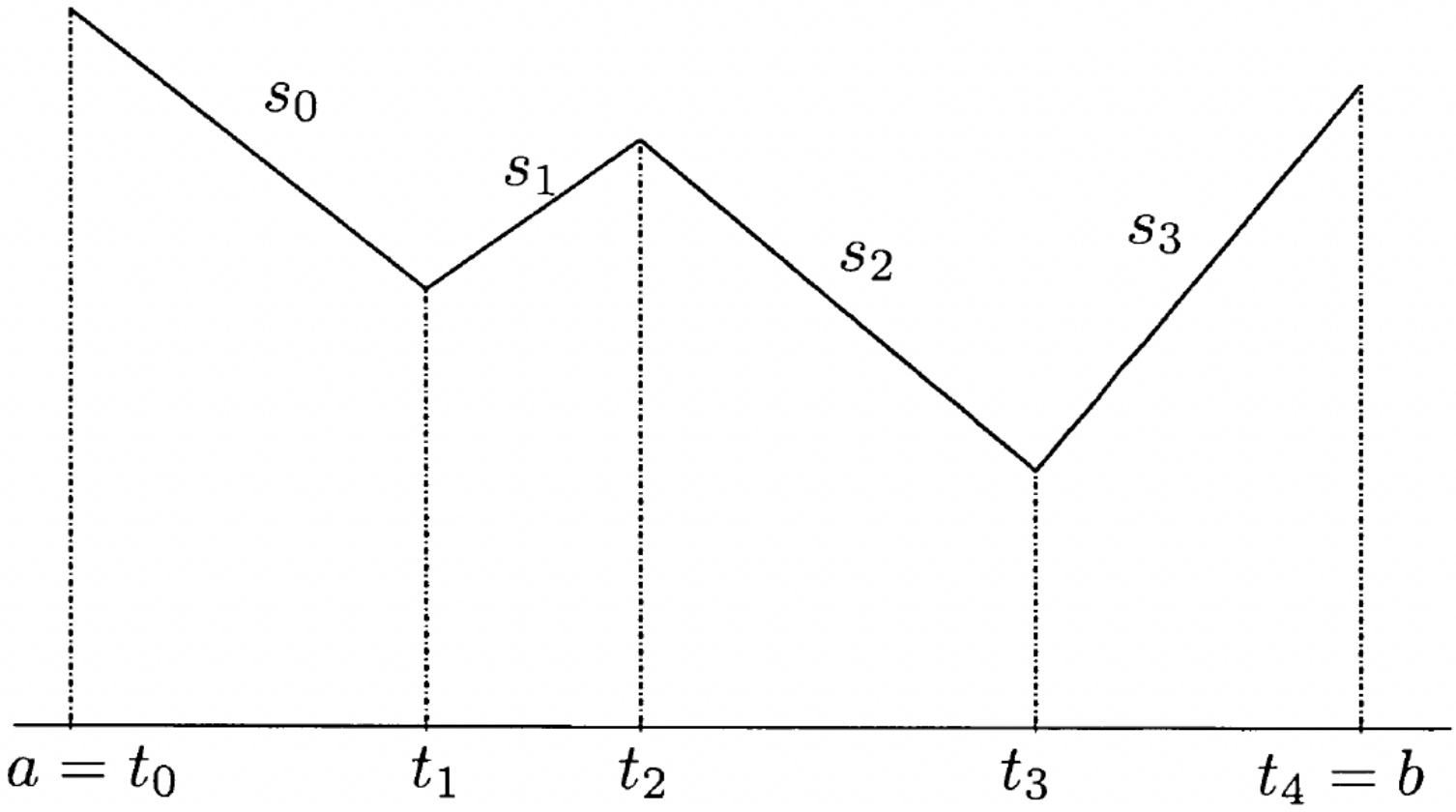
A spline of degree $k-1$ (order k) is a function in C^{k-2} which on each interval $[x_i, x_{i+1}]$ coincides with a polynomial in $\mathbb{P}_{k-1}$.

Most important examples:

- linear splines, $k = 2$
- cubic splines, $k = 4$

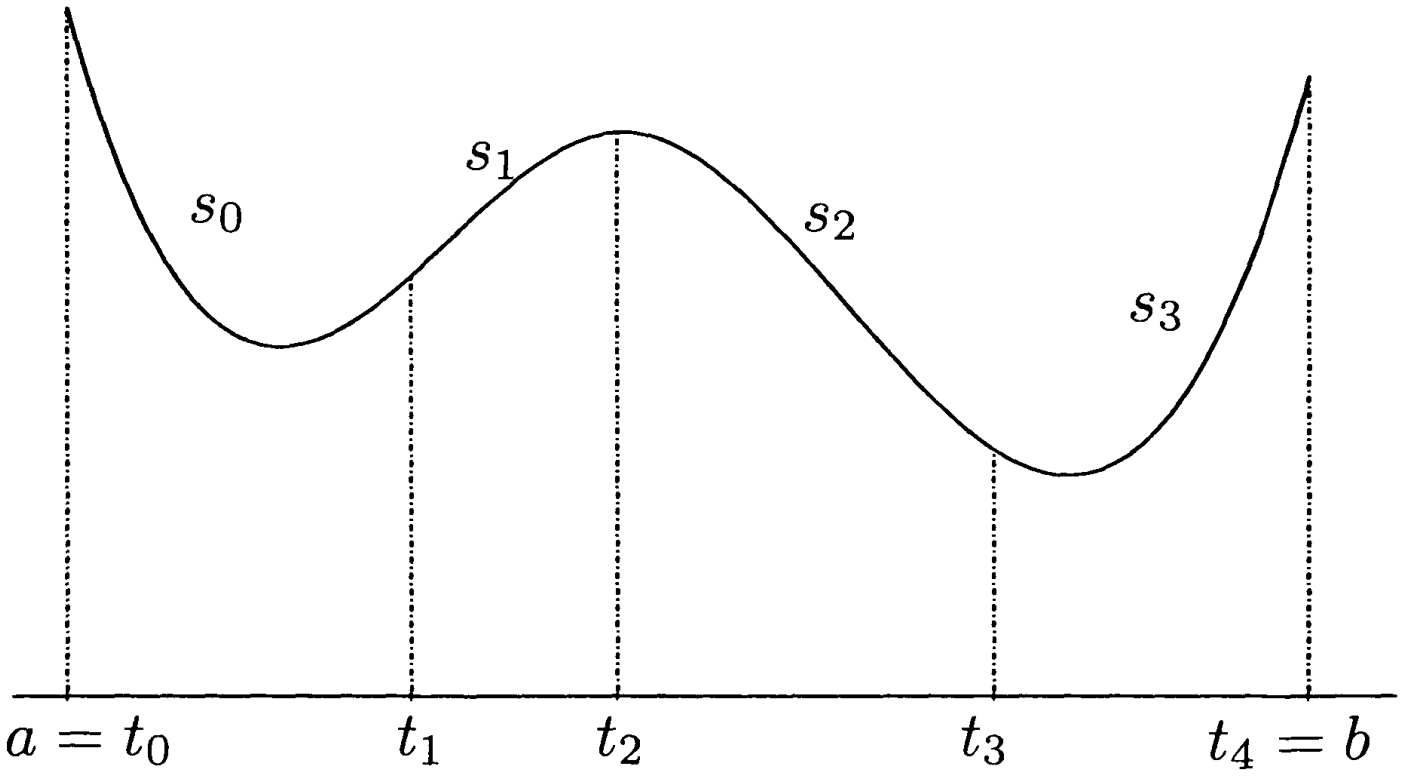
Splines

Linear spline (piecewise linear polynomial)



Splines

Cubic splines look smooth:



Trigonometric Interpolation for periodic functions

Instead of polynomials, use $\sin(jt), \cos(jt)$ for different $j \in \mathbb{N}$.

For $N \geq 1$, we define the set of complex trigonometric polynomials of degree $\leq N-1$ as

$$\boldsymbol{T}_{N-1} := \left\{ \sum_{j=0}^{N-1} c_j e^{ijt}, c_j \in \mathbb{C} \right\},$$

where $i = \sqrt{-1}$.

Complex interpolation problem: Given pairwise distinct nodes $t_0, \ldots, t_{N-1} \in [0, 2\pi)$ and corresponding nodal values $f_0, \ldots, f_{N-1} \in \mathbb{C}$, find a trigonometric polynomial $p \in \boldsymbol{T}_{N-1}$ such that $p(t_i) = f_i$, for $i = 0, \ldots, N-1$.

- There exists exactly one $p \in \boldsymbol{T}_{N-1}$, which solves this interpolation problem.

- Choose the equidistant nodes $t_k := \frac{2\pi k}{N}$ for $k = 0, \ldots, N-1$. The trigonometric polynomial that satisfies $p(t_i) = f_i$ for $i = 0, \ldots, N-1$ has the coefficients

$$c_j = \frac{1}{N} \sum_{k=0}^{N-1} e^{-\frac{2\pi ijk}{N}} f_k.$$

- For equidistant nodes, the linear map from $\mathbb{C}^N \to \mathbb{C}^N$ defined by $(f_0, \ldots, f_{N-1}) \mapsto (c_0, \ldots, c_{N-1})$ is called the discrete Fourier transformation (DFT). The Fast Fourier Transform (FFT) is a (very famous!) algorithm that computes the DFT and its inverse in $O(N \log N)$ flops. $\rightsquigarrow$ Numerical Methods II

Conclusions

- Interpolation means approximating function values in the interior of a domain when there are known samples of the function at a set of interior and boundary nodes
- Given a set of basis functions, interpolation amounts to solving a linear system of equations for the coefficients; there are clever choices to avoid explicitly computing the solution of the system of equations
- Interpolation on equidistant nodes is a bad idea: Not accurate and not stable! Instead, use Chebyshev nodes, then also higher-order polynomials are safe!
- Another option is to use piecewise polynomial interpolation such as splines, which is very common when solving PDEs numerically
- What about higher dimensions?

Solving nonlinear equations

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

What are important properties of a method for solving nonlinear equations?

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

What are important properties of a method for solving nonlinear equations?

- Does it converge? From which starting point x_0?

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

What are important properties of a method for solving nonlinear equations?

- Does it converge? From which starting point x_0?
- How quickly does it converge?

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

What are important properties of a method for solving nonlinear equations?

- Does it converge? From which starting point x_0?
- How quickly does it converge?
- How expensive is each step?

Bisection method

The bisection method exploits that given a continuous function $f : [a, b] \to \mathbb{R}$, such that $f(a)f(b) < 0$, there exists $x^* \in (a, b)$ with $f(x^*) = 0$

- Assumption: f is continuous over $[a, b]$ (very weak assumption!)
- We have chosen a reasonable interval $[a, b]$ so that there exists a solution $x^* \in (a, b)$ with $f(x^*) = 0$

Bisection method

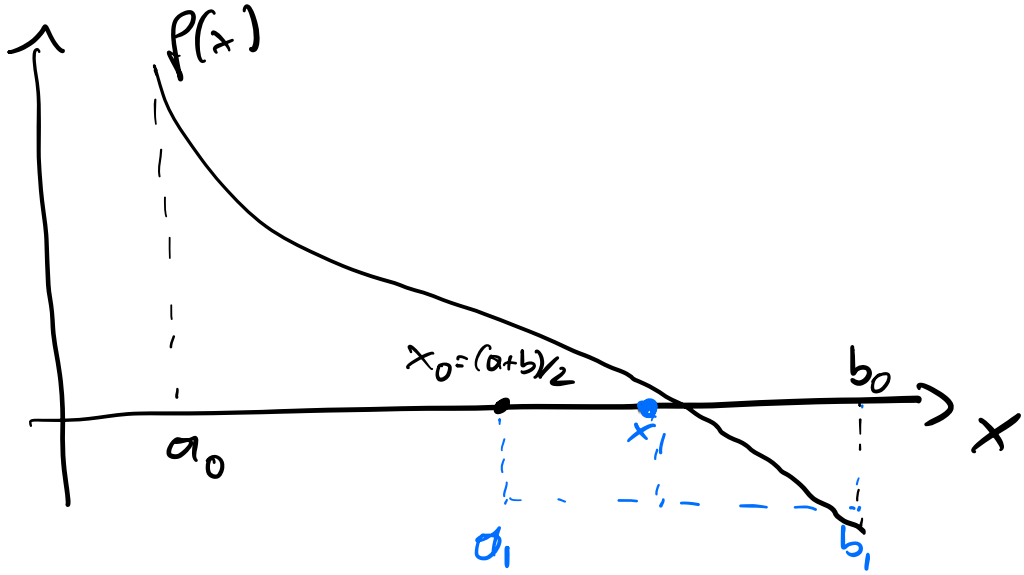
The bisection method exploits that given a continuous function $f : [a, b] \to \mathbb{R}$, such that $f(a)f(b) < 0$, there exists $x^* \in (a, b)$ with $f(x^*) = 0$

- Assumption: f is continuous over $[a, b]$ (very weak assumption!)
- We have chosen a reasonable interval $[a, b]$ so that there exists a solution $x^* \in (a, b)$ with $f(x^*) = 0$

Set $a_0 = a, b_0 = b, x_0 = (a + b)/2$ and iterate for $k = 0, 1, 2, 3, \ldots$ as follows:

1. Set $a_{k+1} = a_k, b_{k+1} = x_k$ if $f(x_k)f(a_k) < 0$
2. Set $a_{k+1} = x_k, b_{k+1} = b_k$ if $f(x_k)f(b_k) < 0$
3. Set $x_{k+1} = (a_{k+1} + b_{k+1})/2$
4. Terminate if $|b_{k+1} - a_{k+1}| \leq \epsilon$

Visualization $\rightsquigarrow$ board



Convergence: set $I_k = [a_k, b_k]$

$$\underline{|I_k|} = b_k - a_k = \frac{b-a}{2^k}$$

Denote the error $e_k = x_k - x^*$ of step k

$$|e_k| = |x_k - x^*| \leq \frac{1}{2}|I_k| = \frac{b-a}{2^{k+1}}$$

$$\lim_{k \to \infty} |e_k| = 0$$

$\Rightarrow$ global convergence

$\Rightarrow$ non-monotone convergence (see plot)

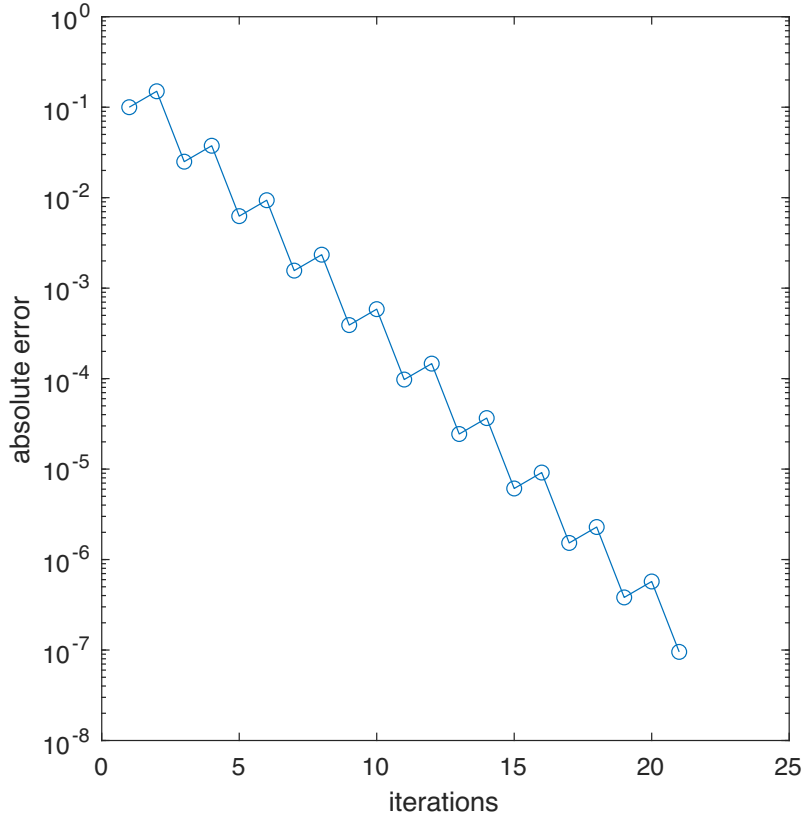
$\Rightarrow$ <u>not</u> linear convergence

Numerical example

Experiment: Solve $f(x) = x^2 - c = 0$ over $[0.5, 1.5]$ with $c = 0.81$ and $x_0 = 1$

```
a = 0.5; b = 1.5; c = 0.81; xStar = sqrt(c);
f = @(x)x^2 - c;
x = (a + b)/2;

res = [x, xStar];
for k=1:20
    if(f(a)*f(x) < 0)
        b = x;
    else
        a = x;
    end
    x = (a + b)/2;
    res(end + 1, :) = [x, xStar];
end
```

1:	1.0000e+00	9.0000e-01
2:	7.5000e-01	9.0000e-01
3:	8.7500e-01	9.0000e-01
4:	9.3750e-01	9.0000e-01
5:	9.0625e-01	9.0000e-01
6:	8.9062e-01	9.0000e-01
7:	8.9844e-01	9.0000e-01
8:	9.0234e-01	9.0000e-01
9:	9.0039e-01	9.0000e-01
10:	8.9941e-01	9.0000e-01
11:	8.9990e-01	9.0000e-01
12:	9.0015e-01	9.0000e-01
13:	9.0002e-01	9.0000e-01
14:	8.9996e-01	9.0000e-01
15:	8.9999e-01	9.0000e-01
16:	9.0001e-01	9.0000e-01
17:	9.0000e-01	9.0000e-01



- Bisection is a slow but sure method.
- It uses no information about the value of the function or its derivatives - only the sign
- There are variants that achieve faster convergence $\rightsquigarrow$ textbook by Quarterioni
- How can we achieve faster convergence in general?

- Bisection is a slow but sure method.
- It uses no information about the value of the function or its derivatives - only the sign
- There are variants that achieve faster convergence $\rightsquigarrow$ textbook by Quarterioni
- How can we achieve faster convergence in general? $\rightsquigarrow$ need to use additional information, at least the function value instead of just the sign

More general formulation via fixed point iterations

Reformulation as fixed point method so that x^* is fixed point

$$x^* = \Phi(x^*)$$

Corresponding iteration: Choose x_0 (initialization) and compute $x_1, x_2, \ldots$ from

$$x_{k+1} = \Phi(x_k)$$

We now want to study when this iteration converges to x^* with $f(x^*) = 0$

Convergence of fixed point methods

A mapping $\Phi : [a, b] \to \mathbb{R}$ is called contractive on $[a, b]$ if there is a $0 \leq \Theta < 1$ such that

$$|\Phi(x) - \Phi(y)| \leq \Theta |x - y| \text{ for all } x, y \in [a, b].$$

If Φ is continuously differentiable on $[a, b]$, then

$$\Theta = \sup_{x,y \in [a,b]} \frac{|\Phi(x) - \Phi(y)|}{|x - y|} = \sup_{z \in [a,b]} |\Phi'(z)|$$

Convergence of fixed point methods

Let $\Phi : [a, b] \to [a, b]$ be contractive with constant $\Theta < 1$. Then:

- There exists a unique fixed point $\bar{x}$ with $\bar{x} = \Phi(\bar{x})$
- For any starting guess x_0 in $[a, b]$, the fixed point iteration converges to $\bar{x}$ and

$$|x_{k+1} - x_k| \leq \Theta |x_k - x_{k-1}|$$

and

$$|\bar{x} - x_k| \leq \frac{\Theta^k}{1 - \Theta} |x_1 - x_0|.$$

The second expression allows to estimate the required number of iterations.

$\rightsquigarrow$ board

Convergence of fixed point methods:

For all $x_0 \in I$, it holds for $x_0, x_1, x_2, \dots$

$$|x_{k+1} - x_k| = |\phi(x_k) - \phi(x_{k-1})| \leq \Theta |x_k - x_{k-1}|$$

because ϕ is contracting with constant Θ.

By induction, we obtain

$$|x_{k+1} - x_k| \leq \Theta^k |x_1 - x_0|$$

$$\downarrow$$

Thus

$$|x_{k+m} - x_k| \leq \overbrace{|x_{k+m} - x_{k+m-1}| + \dots + |x_{k+1} - x_k|}$$

$$\leq (\Theta^{k+m-1} + \dots + \Theta^k) |x_1 - x_0|$$

$$\leq \frac{\Theta^k}{1-\Theta} |x_1 - x_0|$$

because $0 \leq \Theta < 1$

$$(1 + \Theta + \dots + \Theta^{m-1}) \leq \sum_{j=0}^{\infty} \Theta^j = \frac{1}{1-\Theta}$$

$\Rightarrow$ Cauchy sequence $\Rightarrow$ convergence

Show $\lim_{n\to\infty} x_n = x^*$ that is fixed point of ϕ:

$$|x^* - \phi(x^*)| = |x^* - x_{n+1} + x_{n+1} - \phi(x^*)|$$
$$= |x^* - x_{n+1} + \phi(x_n) - \phi(x^*)|$$
$$\leq |x^* - x_{n+1}| + |\phi(x_n) - \phi(x^*)|$$
$$\leq \underbrace{|x^* - x_{n+1}|}_{\xrightarrow[n\to\infty]{} 0} + \theta \underbrace{|x^* - x_n|}_{\xrightarrow[n\to\infty]{} 0}$$

$$|x^* - \phi(x^*)| \leq 0$$
$$|x^* - \phi(x^*)| = 0$$

$\Rightarrow$ fixed point

Uniqueness:

If there are two fixed points x^* and y^* then

$$0 \leq |x^* - y^*| = |\phi(x^*) - \phi(y^*)|$$
$$< \theta |x^* - y^*|$$

Because $\theta < 1$, this is possible only if

$$|x^* - y^*| = 0$$

$\Rightarrow$ uniqueness

Newton's method

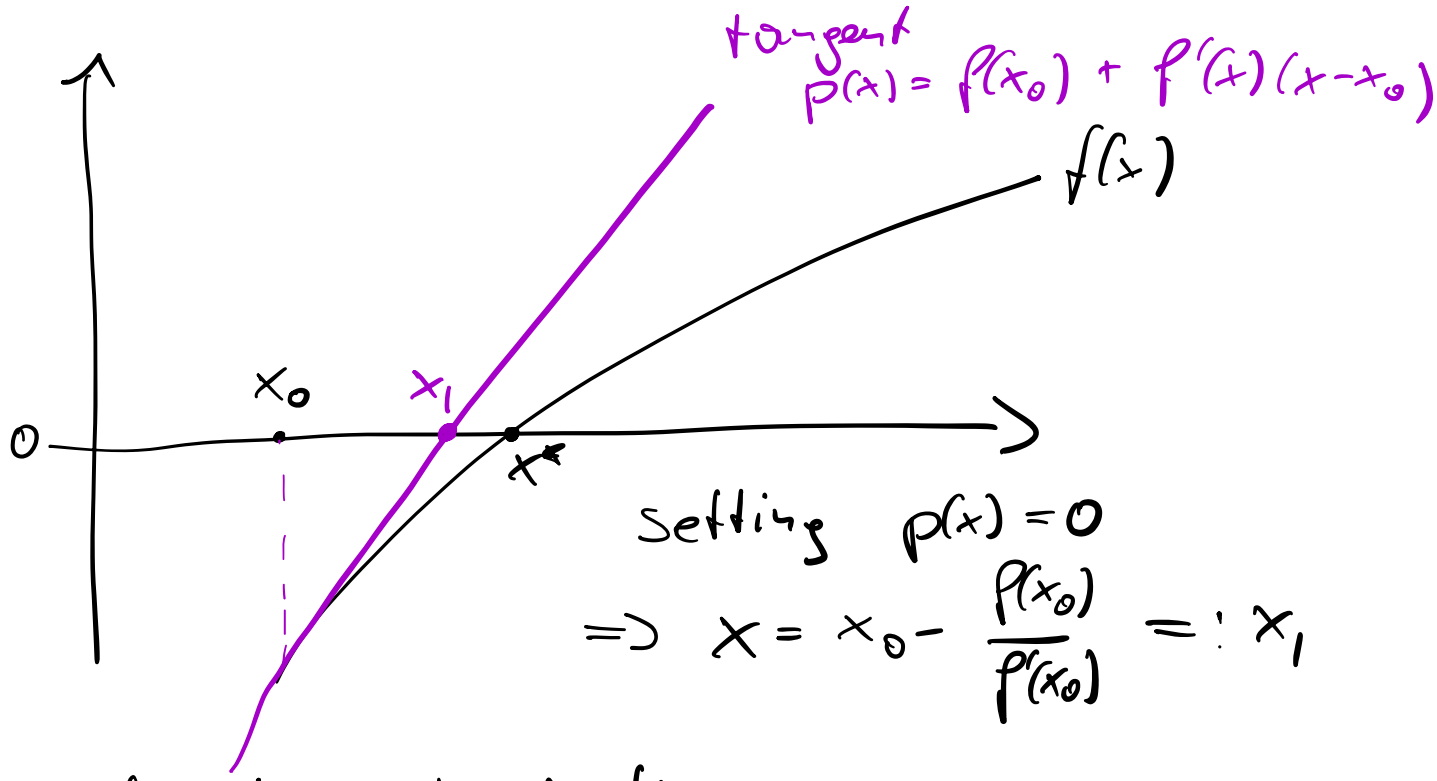
What is the standard approach in numerics when we encounter a nonlinear problem?

Newton's method

What is the standard approach in numerics when we encounter a nonlinear problem?

$\rightsquigarrow$ we linearize

$\rightsquigarrow$ board



setting $p(x) = 0$

$$\Rightarrow x = x_0 - \frac{f(x_0)}{f'(x_0)} =: x_1$$

fixed point iteration

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

$\Rightarrow$ need that f is differentiable

$\Rightarrow$ f' does not vanish

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Interpolation
- Solving systems of nonlinear equations
- Bisection methods

Today

- Newton method

Announcements

- Homework 5 posted and is due Mon, Nov 18 before class

Recap: Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

What are important properties of a method for solving nonlinear equations?

- Does it converge? From which starting point x_0?
- How quickly does it converge?
- How expensive is each step?

Recap: Bisection method

The bisection method exploits that given a continuous function $f : [a, b] \to \mathbb{R}$, such that $f(a)f(b) < 0$, there exists $x^* \in (a, b)$ with $f(x^*) = 0$

- Assumption: f is continuous over $[a, b]$ (very weak assumption!)
- We have chosen a reasonable interval $[a, b]$ so that there exists a solution $x^* \in (a, b)$ with $f(x^*) = 0$

Recap: Bisection method

The bisection method exploits that given a continuous function $f : [a, b] \to \mathbb{R}$, such that $f(a)f(b) < 0$, there exists $x^* \in (a, b)$ with $f(x^*) = 0$

- Assumption: f is continuous over $[a, b]$ (very weak assumption!)
- We have chosen a reasonable interval $[a, b]$ so that there exists a solution $x^* \in (a, b)$ with $f(x^*) = 0$

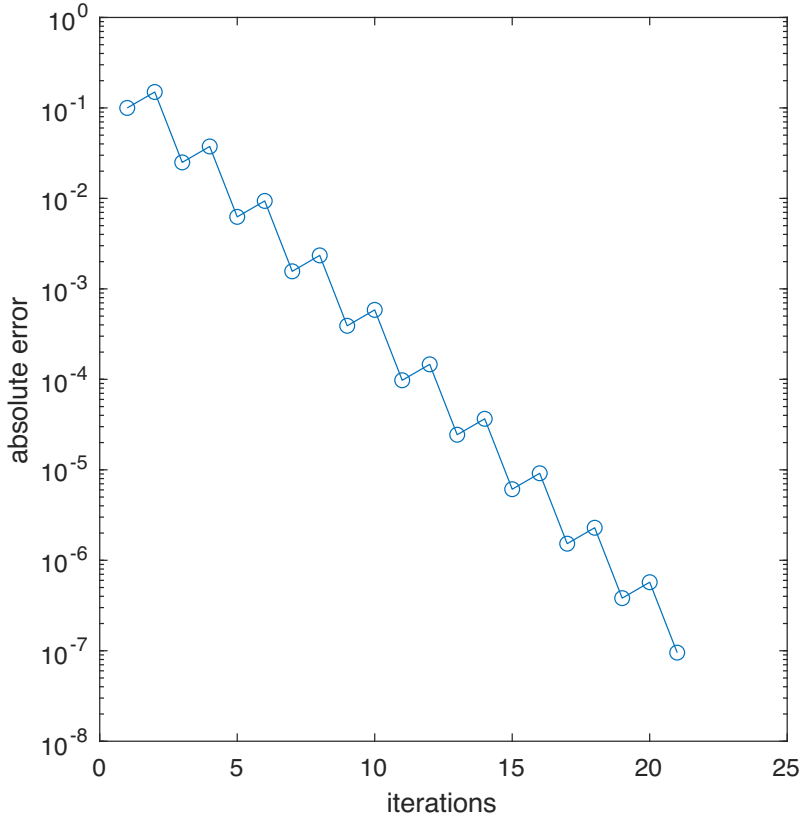
Set $a_0 = a, b_0 = b, x_0 = (a + b)/2$ and iterate for $k = 0, 1, 2, 3, \ldots$ as follows:

1. Set $a_{k+1} = a_k, b_{k+1} = x_k$ if $f(x_k)f(a_k) < 0$
2. Set $a_{k+1} = x_k, b_{k+1} = b_k$ if $f(x_k)f(b_k) < 0$
3. Set $x_{k+1} = (a_{k+1} + b_{k+1})/2$
4. Terminate if $|b_{k+1} - a_{k+1}| \leq \epsilon$

Visualization $\rightsquigarrow$ board

Recap

1:	1.0000e+00	9.0000e-01
2:	7.5000e-01	9.0000e-01
3:	8.7500e-01	9.0000e-01
4:	9.3750e-01	9.0000e-01
5:	9.0625e-01	9.0000e-01
6:	8.9062e-01	9.0000e-01
7:	8.9844e-01	9.0000e-01
8:	9.0234e-01	9.0000e-01
9:	9.0039e-01	9.0000e-01
10:	8.9941e-01	9.0000e-01
11:	8.9990e-01	9.0000e-01
12:	9.0015e-01	9.0000e-01
13:	9.0002e-01	9.0000e-01
14:	8.9996e-01	9.0000e-01
15:	8.9999e-01	9.0000e-01
16:	9.0001e-01	9.0000e-01
17:	9.0000e-01	9.0000e-01

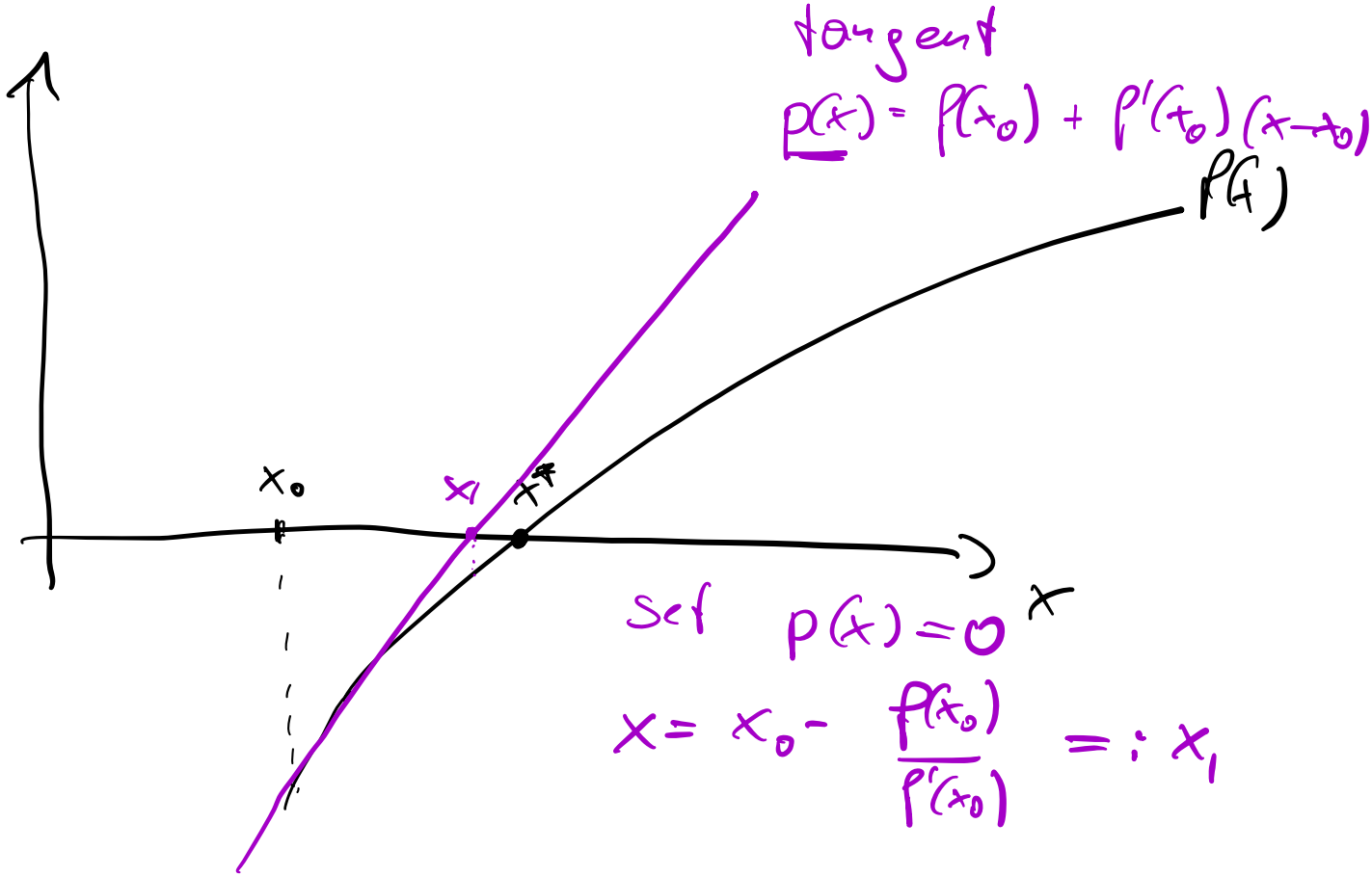


Recap: Newton's method

What is the standard approach in numerics when we encounter a nonlinear problem?

$\rightsquigarrow$ we linearize

$\rightsquigarrow$ board



set $p(x) = 0$

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} =: x_1$$

fixed point iteration

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

→ need differentiable f

→ not vanishing $f'(x)$

In one dimension, solve $f(x) = 0$:

Start with x_0, and compute $x_1, x_2, \ldots$ from

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots$$

Requires $f'(x_k) \neq 0$ to be well-defined (i.e., tangent has nonzero slope).

In one dimension, solve $f(x) = 0$:

Start with x_0, and compute $x_1, x_2, \ldots$ from

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots$$

Requires $f'(x_k) \neq 0$ to be well-defined (i.e., tangent has nonzero slope).

Experiment: Solve $f(x) = x^2 - c = 0$ with $c = 0.81$ and $x_0 = 1$

$$\phi(x) = x - \frac{f(x)}{f'(x)} = x - \frac{x^2 - c}{2x} = x - \frac{x}{2} + \frac{c}{2x} = \frac{1}{2}\left(x + \frac{c}{x}\right)$$

Iterations

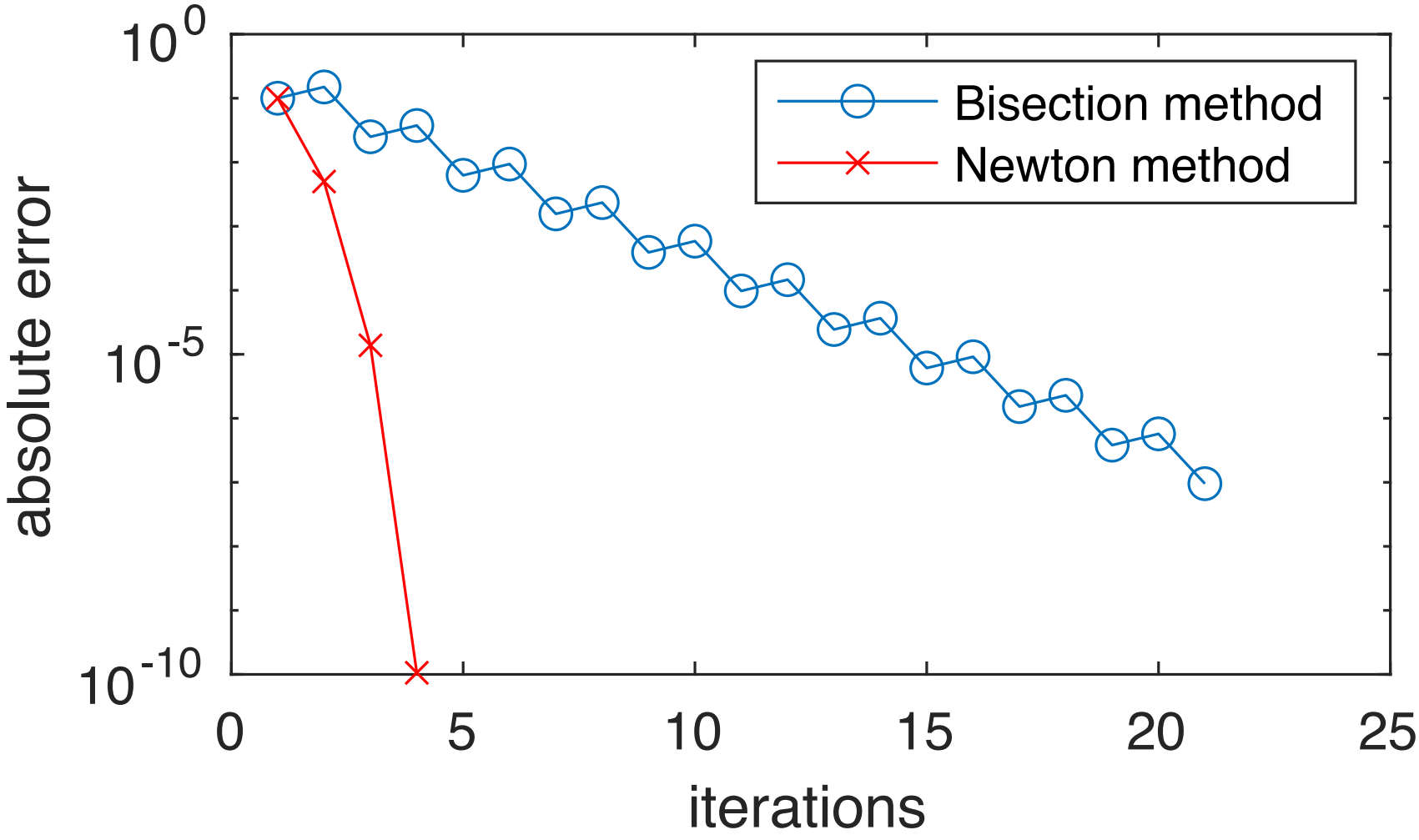
$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{c}{x_k}\right)$$

```
format longE
c = 0.81;
xStar = sqrt(c);
x = 1;
res = [x, xStar];
for i=1:4
    x = 0.5*(x + c/x);
    res(end + 1, :) = [x, xStar];
end
res
```

```
format longE
c = 0.81;
xStar = sqrt(c);
x = 1;
res = [x, xStar];
for i=1:4
    x = 0.5*(x + c/x);
    res(end + 1, :) = [x, xStar];
end
res
```

```
res =
      1.000000000000000e+00      9.000000000000000e-01
      9.050000000000000e-01      9.000000000000000e-01
      9.000138121546961e-01      9.000000000000000e-01
      9.000000001059849e-01      9.000000000000000e-01
      9.000000000000000e-01      9.000000000000000e-01
```

$\rightsquigarrow$ very quick convergence; certainly faster than linear convergence


Bisection method
Newton method
absolute error
iterations
10^{0}
10^{-5}
10^{-10}
0
5
10
15
20
25

Newton's method

Let $F : \mathbb{R}^n \to \mathbb{R}^n$, $n \geq 1$ and solve

$$F(\boldsymbol{x}) = 0.$$

Truncated Taylor expansion of F about starting point $\boldsymbol{x}^0$:

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}^0) + F'(\boldsymbol{x}^0)(\boldsymbol{x} - \boldsymbol{x}^0).$$

Hence:

$$\boldsymbol{x}^1 = \boldsymbol{x}^0 - F'(\boldsymbol{x}^0)^{-1} F(\boldsymbol{x}^0)$$

Newton iteration: Start with $\boldsymbol{x}^0 \in \mathbb{R}^n$, and for $k = 0, 1, \ldots$ compute

$$F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

Requires that $F'(\boldsymbol{x}^k) \in \mathbb{R}^{n \times n}$ is invertible.

Newton's method

Newton iteration: Start with $\boldsymbol{x}^0 \in \mathbb{R}^n$, and for $k = 0, 1, \ldots$ compute

$$F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

Equivalently:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - F'(\boldsymbol{x}^k)^{-1}F(\boldsymbol{x}^k)$$

Newton's method is affine invariant, that is, the sequence is invariant to affine transformations $\rightsquigarrow$ board

Newton is affine invariant:

Solving $F(x)=0$ is equivalent to solving

$$AF(x)=0$$

for regular $A \in \mathbb{R}^{n\times n}$

Set $G(x)=AF(x)$, then apply Newton to G

$$\begin{aligned} y_{k+1} &= y_k - G'(y_k)^{-1} G(y_k) \\ &= y_k - (AF'(y_k))^{-1}(AF(y_k)) \\ &= y_k - F'(y_k)^{-1} A^{-1} A F(y_k) \\ &= y_k - F'(y_k)^{-1} F(y_k) \end{aligned}$$

$\Rightarrow$ iterates are the same even if we transform $F \to G$ via regular matrix

$\Rightarrow$ <u>not</u> affected by bad scaling

Convergence of Newton's method

Assumptions on F: $D \subset \mathbb{R}^n$ open and convex, $F : D \to \mathbb{R}^n$ continuously differentiable with $F'(\boldsymbol{x})$ invertible for all $\boldsymbol{x}$, and there exists $\omega \geq 0$ such that

$$\|F'(\boldsymbol{x})^{-1}(F'(\boldsymbol{x} + s\boldsymbol{v}) - F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

for all $s \in [0, 1]$, $\boldsymbol{x} \in D, \boldsymbol{v} \in \mathbb{R}^n$ with $\boldsymbol{x} + \boldsymbol{v} \in D$.

Assumptions on $\boldsymbol{x}^*$ and $\boldsymbol{x}^0$: There exists a solution $\boldsymbol{x}^* \in D$ and a starting point $\boldsymbol{x}^0 \in D$ such that

$$\rho := \|\boldsymbol{x}^* - \boldsymbol{x}^0\| \leq \frac{2}{\omega} \text{ and } B_\rho(\boldsymbol{x}^*) \subset D$$

where

$$B_\rho(\boldsymbol{x}^*) = \{\boldsymbol{y} \in \mathbb{R}^n \mid \|\boldsymbol{y} - \boldsymbol{x}^*\| < \rho\}$$

Q: Meaning of ω?

Theorem: Under the assumptions of the previous slide, the Newton sequence $\boldsymbol{x}^k$ stays in $B_\rho(\boldsymbol{x}^*)$ and $\lim_{k\to\infty} \boldsymbol{x}^k = \boldsymbol{x}^*$, and

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \frac{\omega}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2$$

Moreover, the solution x^* is unique in $B_{2/\omega}(x^*)$.

Proof: $\rightsquigarrow$ board

Consider the Lipschitz condition

- $\| F'(x)^{-1} (F'(x+sv) - F'(v)) v \|_2 \leq s\omega \|v\|_2^2$
  $\forall s \in [0,1], x \in D, v \in \mathbb{R}^n \quad x+v \in D$

and

- $\displaystyle\int_0^1 [F'(x+s(y-x)) - F'(x)](y-x)\,ds = \dots =$
  $= F(y) - F(x) - F'(x)(y-x)$

Now take $v = y-x$ with $y \in D$, then

$$\int_0^1 \| F'(x)^{-1} [F'(x+s(y-x)) - F'(x)](y-x) \| \, ds$$
$$\leq \int_0^1 s\omega \|y-x\|^2 \, ds$$

$$\| F'(x)^{-1} [F(y) - F(x) - F'(x)](y-x)] \|$$
$$= \| \int_0^1 F'(x)^{-1} [F'(x+s(y-x)) - F'(x)](y-x)\,ds \|$$
$$\leq \int_0^1 \| F'(x)^{-1} [F'(x+s(y-x)) - F'(x)](y-x) \| \, ds$$
$$\leq \int_0^1 s\omega \|y-x\|^2 \, ds$$
$$= \omega \frac{s^2}{2} \|y-x\|^2 \Big|_0^1 = \frac{\omega}{2} \|y-x\|^2 \qquad (*)$$

Now convergence of Newton

$$x^{k+1} - x^* = x^k - F'(x^k)^{-1} F(x^k) - x^*$$

$$= x^k - x^* - F'(x^k)^{-1}\left(F(x^k) - F(x^*)\right)$$

$$= F'(x^k)^{-1}\left[-F(x^k) + F(x^*) - F'(x^k)(x^* - x^k)\right]$$

with (*) follows

$$\|x^{k+1} - x^*\| \leq \frac{\omega}{2} \|x^* - x^k\|^2 \Rightarrow \text{quadratic rate}$$

Now show that sequence (x^k) remains in ball $B_\rho(x^*)$

Have:

$$0 < \|x^k - x^*\| \leq \|x^* - x^0\| =: \rho$$

$$\|x^{k+1} - x^*\| \leq \underbrace{\frac{\omega}{2} \|x^* - x^k\|}_{\leq \rho \frac{\omega}{2}} \|x^* - x^k\|$$

Now we need assumption about starting point

We have that $x^0 \in D$ satisfies

$$\rho = \|x^* - x^0\| < \frac{2}{\omega} \quad \text{and} \quad B_\rho(x^*) \subseteq D$$

$$\Rightarrow \rho \frac{\omega}{2} < 1$$

$$\|x^{k+1} - x^*\| < \|x^k - x^*\| \leq \|x^* - x^0\| = \rho$$

$\Rightarrow (x^k)$ remains in $B_\rho(x^*)$

Thus $\{x^n\} \subseteq B_\rho(x^*) \subseteq D$

important for F' to remain invertible

Uniqueness: the solution x^* is unique in $B_{2/\omega}(x^*)$

$\hookrightarrow$ $x^{**} \in \underline{B_{2/\omega}(x^*)}$ is another solution

so that $F(x^{**}) = 0$ and $\underline{\|x^* - x^{**}\| < 2/\omega}$

With $(*)$ we obtain

$$\|x^{**} - x^*\| = \|0 - 0 - F'(x^*)^{-1} F'(x^*)(x^{**} - x^*)\|$$

$$\overset{F(x^*) = F(x^{**}) = 0}{=} \|F'(x^*)^{-1}(F(x^{**}) - F(x^*) - F'(x^*)(x^* - x^{**}))\|$$

$$\overset{\text{use } (*)}{\leq} \underbrace{\frac{\omega}{2}\|x^{**} - x^*\|}_{<1} \|x^{**} - x^*\|$$

thus only $x^{**} = x^*$

Theorem: Under the assumptions of the previous slide, the Newton sequence $\boldsymbol{x}^k$ stays in $B_\rho(\boldsymbol{x}^*)$ and $\lim_{k\to\infty} \boldsymbol{x}^k = \boldsymbol{x}^*$, and

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \frac{\omega}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2$$

Moreover, the solution x^* is unique in $B_{2/\omega}(x^*)$.

Proof: $\rightsquigarrow$ board

Summary: The Newton method converges locally and quadratically.

Role of initialization

Choice of initialization $\boldsymbol{x}^0$ is critical. Depending on the initialization, the Newton iteration might

- not converge (it could "blow up" or "oscillate" between two points)
- converge to different solutions
- fail because it hits a point where the Jacobian is not invertible (this cannot happen if the conditions of the convergence theorem are satisfied)
- ...

Sometimes, continuation ideas must be used to find good initializations: Solve simpler problems first and use solution as starting point for harder problems.

General comments

The "more nonlinear" a problem, the harder it is to solve.

$$\|F'(\boldsymbol{x})^{-1}(F'(\boldsymbol{x}+s\boldsymbol{v})-F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

Very nonlinear $\rightsquigarrow$ $F'(x)$ changes a lot $\rightsquigarrow$ ω large (need x_0 closers to x^* required)

General comments

The "more nonlinear" a problem, the harder it is to solve.

$$\|F'(\boldsymbol{x})^{-1}(F'(\boldsymbol{x}+s\boldsymbol{v})-F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

Very nonlinear $\rightsquigarrow$ $F'(x)$ changes a lot $\rightsquigarrow$ ω large (need x_0 closers to x^* required)

Computation of Jacobian can be costly/complicated
$\rightsquigarrow$ sometimes approximate $F'(x^k)$

General comments

The "more nonlinear" a problem, the harder it is to solve.

$$\|F'(\boldsymbol{x})^{-1}(F'(\boldsymbol{x}+s\boldsymbol{v})-F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

Very nonlinear $\rightsquigarrow$ $F'(x)$ changes a lot $\rightsquigarrow$ ω large (need x_0 closers to x^* required)

Computation of Jacobian can be costly/complicated
$\rightsquigarrow$ sometimes approximate $F'(x^k)$

There's no reliable black-box solver for nonlinear problems; at least for higher-dimensional problems, the structure of the problem must be taken into account.

General comments

The "more nonlinear" a problem, the harder it is to solve.

$$\|F'(\boldsymbol{x})^{-1}(F'(\boldsymbol{x}+s\boldsymbol{v})-F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

Very nonlinear $\rightsquigarrow$ $F'(x)$ changes a lot $\rightsquigarrow$ ω large (need x_0 closers to x^* required)

Computation of Jacobian can be costly/complicated
$\rightsquigarrow$ sometimes approximate $F'(x^k)$

There's no reliable black-box solver for nonlinear problems; at least for higher-dimensional problems, the structure of the problem must be taken into account.

"Classification of mathematical problems as linear and nonlinear is like classification of the Universe as bananas and non-bananas."

Overview

Nonlinear least squares—Gauss-Newton

Nonlinear least-squares problems

Assume a least squares problem, where the parameters $\boldsymbol{x}$ do *not* enter linearly into the model.

Instead of

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|A\boldsymbol{x} - \boldsymbol{b}\|^2,$$

we have with $F : D \to \mathbb{R}^m, m \geq n,\ D \subset \mathbb{R}^n$:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} g(\boldsymbol{x}) := \frac{1}{2}\|F(\boldsymbol{x})\|^2, \qquad \text{where } F(\boldsymbol{x})_i = \varphi(t_i, \boldsymbol{x}) - b_i, 1 \leq i \leq m$$

A (local) minimum $\boldsymbol{x}^*$ of this optimization problem satisfies:

$$g'(\boldsymbol{x}) = 0, \quad g''(\boldsymbol{x}) \text{ is positive definite.}$$

Nonlinear least-squares problems

The derivative of $g(\cdot)$ is

$$G(\boldsymbol{x}) := g'(\boldsymbol{x}) = F'(\boldsymbol{x})F(\boldsymbol{x})$$

Setting $G(\boldsymbol{x}) = 0$ gives a nonlinear system in $\boldsymbol{x}$, $G : D \to \mathbb{R}^m$.

Let's try to solve it $G(\boldsymbol{x}) = 0$ using Newton's method:

$$G'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -G(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

where

$$G'(\boldsymbol{x}) = F'(\boldsymbol{x})^T F'(\boldsymbol{x}) + F''(\boldsymbol{x})^T F(\boldsymbol{x}) \qquad \text{Hessian of } g \text{ (objective)}$$

$\rightsquigarrow$ second-order information of F enters through $F''(X)^T$

Q: What are you observing about how second-order information enters?

Nonlinear least-squares problems

The derivative of $g(\cdot)$ is

$$G(\boldsymbol{x}) := g'(\boldsymbol{x}) = F'(\boldsymbol{x})F(\boldsymbol{x})$$

Setting $G(\boldsymbol{x}) = 0$ gives a nonlinear system in $\boldsymbol{x}$, $G : D \to \mathbb{R}^m$.

Let's try to solve it $G(\boldsymbol{x}) = 0$ using Newton's method:

$$G'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -G(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

where

$$G'(\boldsymbol{x}) = F'(\boldsymbol{x})^T F'(\boldsymbol{x}) + F''(\boldsymbol{x})^T F(\boldsymbol{x}) \qquad \text{Hessian of } g \text{ (objective)}$$

$\rightsquigarrow$ second-order information of F enters through $F''(X)^T$

Q: What are you observing about how second-order information enters?$\rightsquigarrow$ multiplied with $F(x)$

Nonlinear least-squares problems

If the data is compatible with the model, which means that the model can perfectly fit the data with zero training error, then $F(\boldsymbol{x}^*) = 0$

Then, term involving $F''(\boldsymbol{x}^*)$ drops out in $G'(\boldsymbol{x}^*)$ anyway as we move towards $\boldsymbol{x}^*$.

If $\|F(\boldsymbol{x}^*)\|$ is small, and thus data *almost* compatible with model, then neglecting that term might not make the convergence much slower.

Also, it's expensive to compute $F''(\boldsymbol{x})$

Nonlinear least-squares problems—Gauss-Newton

The resulting Newton method for the nonlinear least squares problem is called Gauss-Newton method: Initialize $\boldsymbol{x}^0$ and for $k = 0, 1, \ldots$ solve

$$F'(\boldsymbol{x}^k)^T F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F'(\boldsymbol{x}^k)^T F(\boldsymbol{x}^k) \quad \text{(solve)} \tag{1}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k. \quad \text{(update step)}$$

Nonlinear least-squares problems—Gauss-Newton

The resulting Newton method for the nonlinear least squares problem is called Gauss-Newton method: Initialize $\boldsymbol{x}^0$ and for $k = 0, 1, \ldots$ solve

$$F'(\boldsymbol{x}^k)^T F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F'(\boldsymbol{x}^k)^T F(\boldsymbol{x}^k) \quad \text{(solve)} \tag{1}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k. \quad \text{(update step)}$$

Q: Should we implement GN like this?

Nonlinear least-squares problems—Gauss-Newton

The resulting Newton method for the nonlinear least squares problem is called Gauss-Newton method: Initialize $\boldsymbol{x}^0$ and for $k = 0, 1, \ldots$ solve

$$F'(\boldsymbol{x}^k)^T F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F'(\boldsymbol{x}^k)^T F(\boldsymbol{x}^k) \quad \text{(solve)} \tag{1}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k. \quad \text{(update step)}$$

Q: Should we implement GN like this?

The solve step is the normal equation for the linear least squares problem

$$\min_{\Delta\boldsymbol{x}} \|F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k + F(\boldsymbol{x}^k)\|. \tag{2}$$

so we better solve (2) rather than directly (1)

Convergence of Gauss-Newton method

Assumptions on F: $D \subset \mathbb{R}^n$ open and convex, $F : D \to \mathbb{R}^m$, $m \geq n$ continuously differentiable with $F'(\boldsymbol{x})$ has full rank for all $\boldsymbol{x}$, and let $\omega \geq 0, 0 \leq \kappa^* < 1$ such that

$$\|F'(\boldsymbol{x})^+(F'(\boldsymbol{x}+s\boldsymbol{v}) - F'(\boldsymbol{x}))\boldsymbol{v}\| \leq s\omega\|\boldsymbol{v}\|^2$$

for all $s \in [0,1]$, $\boldsymbol{x} \in D, \boldsymbol{v} \in \mathbb{R}^n$ with $\boldsymbol{x}+\boldsymbol{v} \in D$.

Assumptions on $\boldsymbol{x}^*$ and $\boldsymbol{x}^0$: Assume there exists a solution $\boldsymbol{x}^* \in D$ of the least squares problem and a starting point $\boldsymbol{x}^0 \in D$ such that

$$\|F'(\boldsymbol{x})^+F(\boldsymbol{x}^*)\| \leq \kappa^*\|\boldsymbol{x}-\boldsymbol{x}^*\|$$

$$\rho := \|\boldsymbol{x}^* - \boldsymbol{x}^0\| \leq \frac{2(1-\kappa^*)}{\omega} := \sigma$$

Theorem: Then, the sequence $\boldsymbol{x}^k$ stays in $B_\rho(\boldsymbol{x}^*)$ and $\lim_{k\to\infty} \boldsymbol{x}^k = \boldsymbol{x}^*$, and

$$\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^*\| \leq \frac{\omega}{2}\|\boldsymbol{x}^k - \boldsymbol{x}^*\|^2 + \underbrace{\kappa^*\|\boldsymbol{x}^k - \boldsymbol{x}^*\|}_{\rightsquigarrow \text{ linear convergence if } \kappa^*>0!}$$

$\rightsquigarrow$ we usually want to choose models that are "almost compatible" which means κ^* is often very small

Conclusions

- Solving nonlinear systems of equations ("root finding") is iterative in nature in general
- The order of convergence matters; quadratic is good enough but mind costs per step
- Newton's method is second order but requires derivatives/Jacobian evaluations.
- In higher dimensions, a good initial guess is critical for Newton's method
- There are many variants of Newton's method (e.g., Quasi-Newton methods) that avoid the computational costs of computing the Hessian
- (Machine learning is using first-order methods only anyway...)

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Two weeks ago

- Function approximation
- Interpolation with polynomials
- Interpolation beyond polynomials

Today

- Numerical integration
- Newton-Cotes

Announcements

- Homework 5 is posted and due Mon, Nov 18 before class

Numerical quadrature

Numerical integration

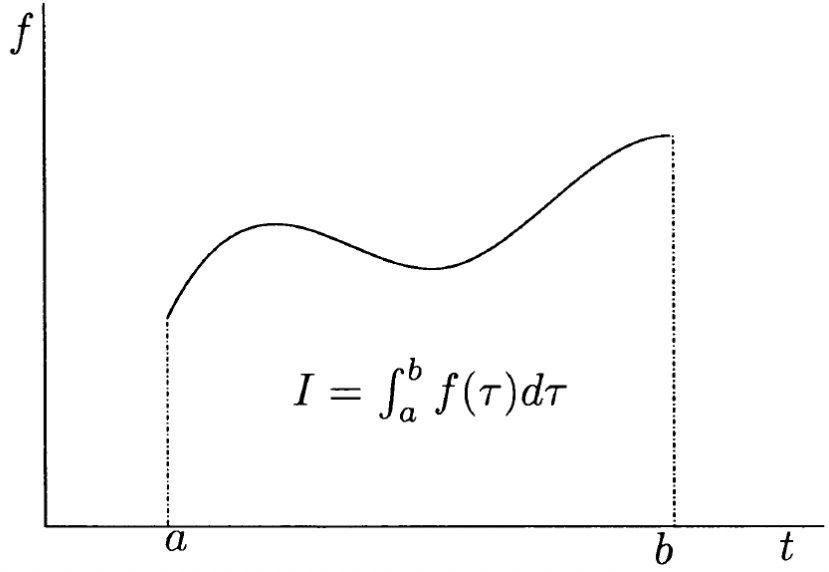
We want to approximate the definite integral

$$I(f) = I_a^b(f) = \int_a^b f(t)\,dt$$

numerically.

Properties of the integral:

- I is linear
- positive, i.e., if f is nonnegative, then $I(f)$ is nonnegative
- additive w.r.t. the interval bounds: $I_a^c = I_a^b + I_b^c$



Condition of numerical integration

Lets study the map

$$([a,b],f) \to \int_a^b f(t)\,dt,$$

where we use the L^1-norm for f

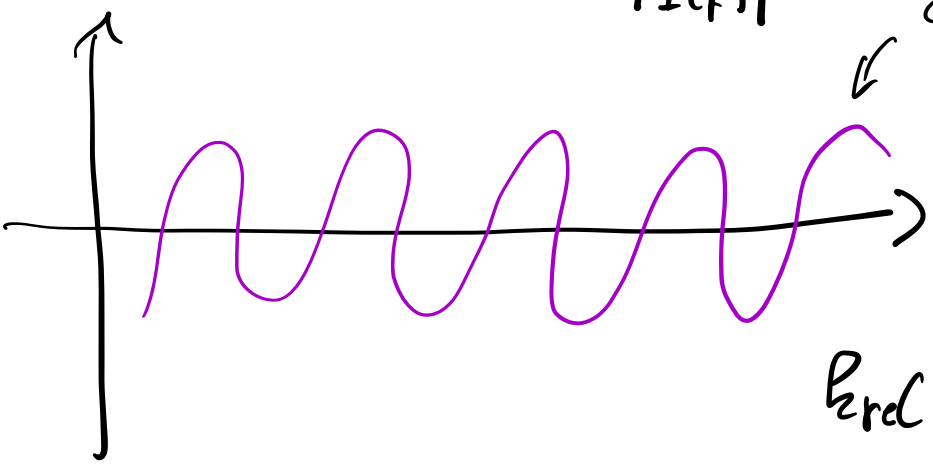
$$\|f\|_1 = \int_a^b |f(t)|\mathrm{d}t = I(|f|)$$

The absolute and relative condition numbers of integration are:

$$\kappa_{\text{abs}} = 1,$$
$$\kappa_{\text{rel}} = \frac{I(|f|)}{|I(f)|}.$$

What does this mean?

Condition number $\quad \kappa_{rel} = \dfrac{I(|f|)}{|I(f)|}$



changes sign often

$I(f) \approx 0$

$I(|f|) \gg 0$

$\kappa_{rel} = \dfrac{\text{large}}{\text{small}} \gg 1$

comparable to addition

$(a,b) \mapsto a+b \qquad a, b \in \mathbb{R}$

$\kappa_{rel} = \dfrac{|a|+|b|}{|a+b|}$

if $a \approx -b$, then poorly condition

Condition of numerical integration

Lets study the map

$$([a,b],f) \to \int_a^b f(t)\,dt,$$

where we use the L^1-norm for f

$$\|f\|_1 = \int_a^b |f(t)|\mathrm{d}t = I(|f|)$$

The absolute and relative condition numbers of integration are:

$$\kappa_{\text{abs}} = 1,$$
$$\kappa_{\text{rel}} = \frac{I(|f|)}{|I(f)|}.$$

What does this mean? Integration is harmless w.r.t. the absolute condition number, and problematic w.r.t. the relative condition number if $I(f)$ is small and f changes sign

$\rightsquigarrow$ board

Numerical integration

We are looking for a map

$$\hat{I} : \begin{cases} C([a,b]) & \to \mathbb{R} \\ f & \mapsto \hat{I}(f) \end{cases}$$

such that the integration error $|I(f) - \hat{I}(f)|$ is small.

What ideas come to your mind?

Numerical integration

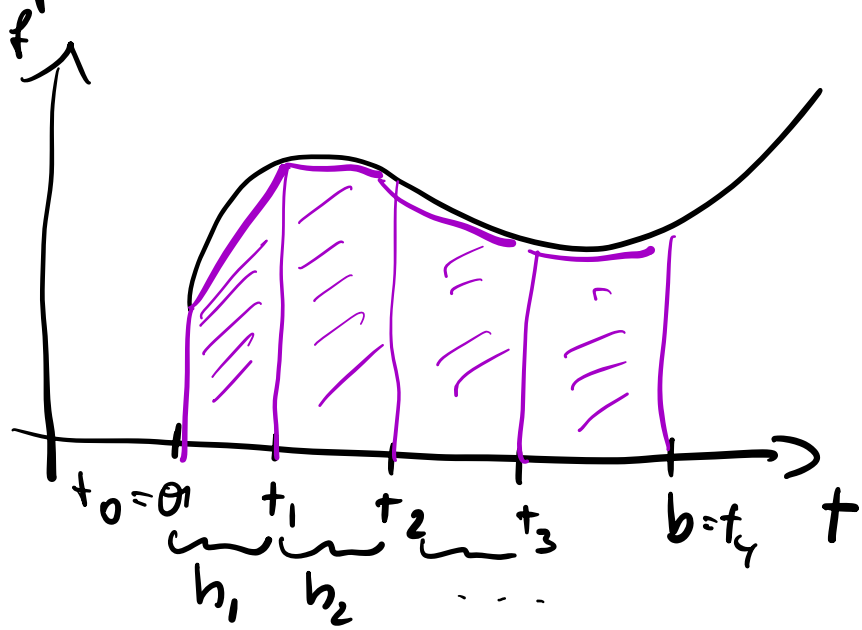
We are looking for a map

$$\hat{I} : \begin{cases} C([a,b]) & \to \mathbb{R} \\ f & \mapsto \hat{I}(f) \end{cases}$$

such that the integration error $|I(f) - \hat{I}(f)|$ is small.

What ideas come to your mind? Example: Trapezoidal sum $\rightsquigarrow$ board

Trapezoidal sum



$$T^{(n)} = \sum_{i=1}^{n} T_i, \qquad T_i = \frac{h_i}{2}\left(f(t_{i-1}) + f(t_i)\right)$$

Lower and upper bound sum:

$$R_{min}^{(n)} = \sum_{i=1}^{n} h_i \min_{t \in [t_{i-1}, t_i]} f(t)$$

$$R_{max}^{(n)} = \sum_{i=1}^{n} h_i \max_{t \in [t_{i-1}, t_i]} f(t)$$

$$R_{min}^{(n)} \le T^{(n)} \le R_{max}^{(n)}$$

$$\downarrow \qquad\qquad \downarrow$$

$$I(f) \qquad\qquad I(f)$$

for continuous f
$n \to \infty$

$$T^{(n)} \xrightarrow{n \to \infty} I(f)$$

Numerical integration

We are looking for a map

$$\hat{I} : \begin{cases} C([a,b]) & \to \mathbb{R} \\ f & \mapsto \hat{I}(f) \end{cases}$$

such that the integration error $|I(f) - \hat{I}(f)|$ is small.

What ideas come to your mind? Example: Trapezoidal sum $\rightsquigarrow$ board

General quadrature formula:

$$\hat{I}(f) = \sum_{i=0}^{n} \lambda_i f(t_i),$$

with weights λ_i and nodal points t_i, $i = 0, 1, \ldots, n$.

Newton-Cotes formulas

Trapezoidal rule replaces f by easy-to-integrate piecewise linear approximation $\hat{f}$.
What other easy-to-integrate approximations could we use?

Newton-Cotes formulas

Trapezoidal rule replaces f by easy-to-integrate piecewise linear approximation $\hat{f}$.
What other easy-to-integrate approximations could we use?

Polynomials!

Quick recap on polynomial interpolation

Recap: Interpolation

Consider n pairs of data samples $(x_i, y_i), i = 1, \ldots, n$ with

$$y_i = f(x_i)$$

Based on $\{(x_i, y_i)\}_{i=1}^n$, we now would like to find an approximation $\tilde{f} \in \mathcal{V}_n$ that is "close" to f.

For example, we could enforce the interpolation condition, namely that it holds

$$\tilde{f}(x_i) = f(x_i)\,, \qquad i = 1, \ldots, n$$

We could also use regression $(m > n)$ and minimize, e.g.,

$$\frac{1}{m} \sum_{i=1}^{m} |y_i - \tilde{f}(x_i)|^2$$

Recap: Orthogonal polynomials

Define an inner product between functions:

$$(f, g) = \int_a^b \omega(x) f(x) g(x)\, dx,$$

where $\omega(x) > 0$ for $a \leq x \leq b$ is a weight function. The induced norm is $\|f\| := \sqrt{(f, f)}$.

Let $P_0, P_1, P_2, \ldots, P_K$ be polynomials of $0, 1, 2, \ldots, K$ order, respectively. They are called orthogonal polynomials on $[a, b]$ with respect to the weight function $\omega(x)$ if it holds

$$(P_i, P_j) = \int_a^b \omega(x) P_i(x) P_j(x) \mathrm{d}x = \delta_{ij} \gamma_i \,, \qquad i, j = 0, \ldots, K \,,$$

with $\gamma_i = \|P_i\|^2 > 0$.

Recap

To define orthogonal polynomials uniquely, we normalize them so that the leading coefficient is one, i.e.,

$$P_k(x) = x^k + \dots$$

Theorem: There exist uniquely determined orthogonal polynomials $P_k \in \mathbb{P}_k$ with leading coefficient 1. These polynomials satisfy the 3-term recurrence relation:

$$P_k(x) = (x + a_k)P_{k-1}(x) + b_k P_{k-2}(x), \quad k = 2, 3, \dots$$

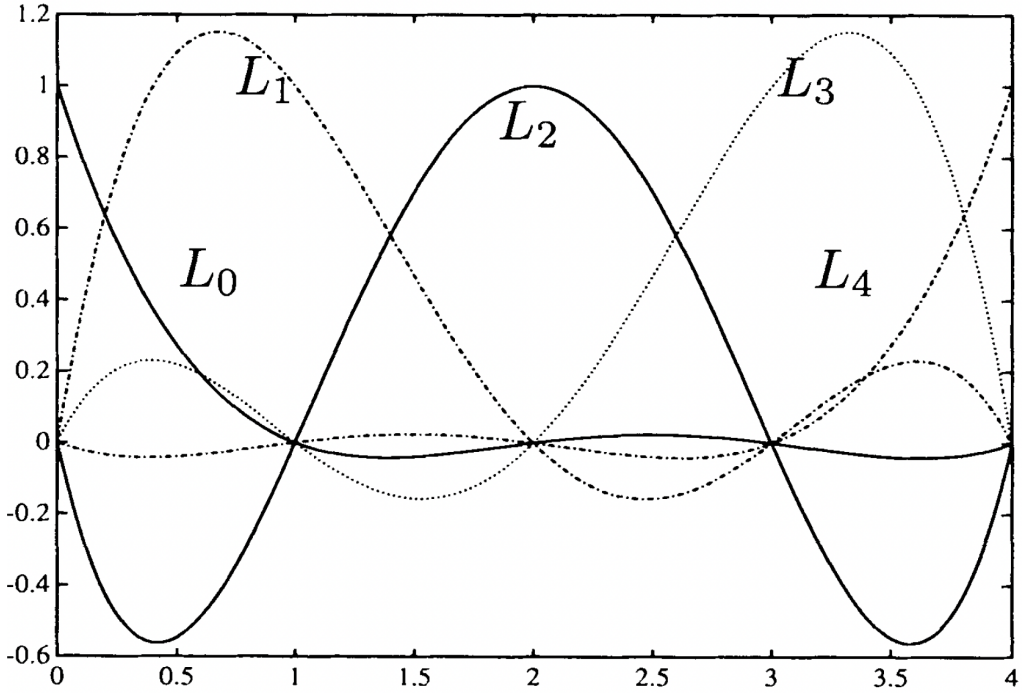
with starting values $P_0 = 1$, $P_1 = x + a_1$, where

$$a_k = -\frac{(xP_{k-1}, P_{k-1})}{(P_{k-1}, P_{k-1})}, \quad b_k = -\frac{(P_{k-1}, P_{k-1})}{(P_{k-2}, P_{k-2})}$$

Recap: Lagrange basis

The Lagrange polynomials $L_0, \ldots, L_n \in \mathbb{P}_n$ are uniquely defined for distinct $x_0, \ldots, x_n$

$$L_i(x_j) = \delta_{ij}\,, \qquad L_i \in \mathbb{P}_n\,.$$



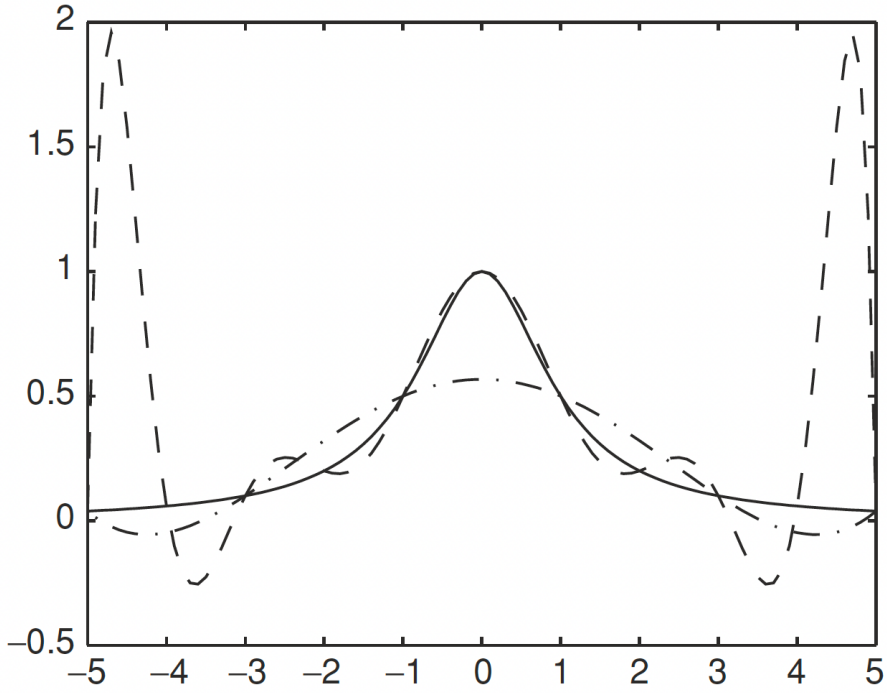
Lagrange polynomials up to order $n = 4$ for equidistant $x_0, \ldots, x_4$. [Figure: Deuflhard]

Recap: Runge's phenomenon

Let's approximate

$$f(x) = \frac{1}{1+x^2}\,, \qquad -5 \le x \le 5\,,$$

using polynomial interpolation on equally spaced nodes in $[-5, 5]$.



[Figure: Quarteroni]

Interpolants of degree $n = 5$ and $n = 10$ of $f(x) = 1/(1+x^2)$ on equidistant nodes. It can be shown that polynomial interpolation does not converge for $|x| > 4$ for this f

It is a very common situation that interpolation on equidistant nodes leads to high oscillations near the interval ends $\rightsquigarrow$ Runge's phenomenon

Recap: Newton polynomial basis

The leading coefficient a_n of the interpolation polynomial (monomial basis!)

$$P_f(x|x_0, \ldots, x_n) = a_n x^n + \cdots + a_0$$

is called the *n-th divided difference*, $[x_0, \ldots, x_n]f := a_n$.

The divided differences are the coefficients $c_0, \ldots, c_n$: The interpolation polynomial $P_f(\cdot|x_0, \ldots, x_n)$ for $x_0 \leq x_1 \leq \cdots \leq x_n$ (not necessarily distinct and thus need $f \in C^{n+1}$) is given by

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i]f \, \omega_i(x).$$

Furthermore,

$$f(x) = P(x) + [x_0, \ldots, x_n, x]f \, \omega_{n+1}(x).$$

Back to quadrature

Newton-Cotes formulas continued

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt$$

Newton-Cotes formulas (cont'd)

Quadrature formulas defined in this way are exact for polynomials $P \in \mathbb{P}_n$ of degree less than or equal to n

$$\hat{I}(P) = I(P_n(P)) = I(P)\,, \qquad \text{for all } P \in \mathbb{P}_n$$

Theorem: For $(n+1)$ pairwise distinct nodes $t_0, \ldots, t_n$, there exists exactly one quadrature formula (i.e., unique weights $\lambda_0, \ldots, \lambda_n$)

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(t_i)\,,$$

that is exact for all $p \in \mathbb{P}_n$.

Proof $\rightsquigarrow$ board

$n+1$ pairwise distinct nodes $t_0, \dots, t_n$, then there exists one and only one quad rule

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(t_i)$$

which is exact for all polynomials $P \in \mathbb{P}_n$.

↳ have $\hat{I}$ that is exact for $P \in \mathbb{P}_n$

insert Lagrange polynomials for $t_0, \dots, t_n$

$$L_{in} \in \mathbb{P}_n$$

$$I(L_{in}) = \hat{I}(L_{in}) = (b-a) \sum_{j=0}^{n} \lambda_j \underbrace{L_{in}(t_j)}_{\delta_{ij}}$$

exact!

$$= (b-a) \lambda_j$$

$$\lambda_j = \frac{1}{(b-a)} I(L_{jn})$$

As long as $I(L_{in}) = \hat{I}(L_{in})$ we get the same weights.

$\Rightarrow$ unique

Equidistantly spaced nodes

$$h_i = h = \frac{b-a}{n}\,, \qquad t_i = a + ih\,, \qquad i = 0, \ldots, n$$

then quadrature formulas are called the *Newton-Cotes formulas* with weights

$$\lambda_{in} = \frac{1}{b-a}\int_a^b \prod_{i\neq j} \frac{t-t_i}{t_i-t_j}\,dt = \frac{1}{n}\int_0^n \prod_{i\neq j} \frac{s-j}{i-j}\,ds$$

These weights are independent of the interval boundaries a and b and can be pre-computed once and for all:

Table 9.1. Newton-Cotes weights λ_{in} for $n = 1, \ldots, 4$.

n	$\lambda_{0n}, \ldots, \lambda_{nn}$	Error	Name
1	$\frac{1}{2}$ $\frac{1}{2}$	$\frac{h^3}{12} f''(\tau)$	Trapezoidal rule
2	$\frac{1}{6}$ $\frac{4}{6}$ $\frac{1}{6}$	$\frac{h^5}{90} f^{(4)}(\tau)$	Simpson's rule, Kepler's barrel rule
3	$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{8}$ $\frac{1}{8}$	$\frac{3h^5}{80} f^{(4)}(\tau)$	Newton's 3/8-rule
4	$\frac{7}{90}$ $\frac{32}{90}$ $\frac{12}{90}$ $\frac{32}{90}$ $\frac{7}{90}$	$\frac{8h^7}{945} f^{(6)}(\tau)$	Milne's rule

Lemma Let $f \in C^2([a,b])$ be a twice continuously differentiable function. Then the integration error of the trapezoidal rule

$$T = \frac{b-a}{2}(f(a) + f(b))$$

with step size $h = b - a$ can be expressed by

$$T - \int_a^b f = \frac{(b-a)^3}{12} f''(\tau)\,,$$

for some $\tau \in [a,b]$

Proof $\rightsquigarrow$ board

Integration errors of other Newton-Cotes formulas are listed in the table on the previous slide.

Helper mean value theorem:

Let $g, h \in C([a,b])$ be cont. functions on $[a,b]$, where g has only one sign,

either $g(t) \geq 0$ or $g(t) < 0$ on $[a,b]$

then there exists $\tau \in [a,b]$ such that

$$\int_a^b h(t)\, g(t)\, dt = h(\tau) \int_a^b g(t)\, dt$$

Let us assume $g(t) \geq 0$.

$$\min_{t \in [a,b]} h(t) \int_a^b g(s)\, ds \leq \int_a^b h(s) g(s)\, ds$$

$$\leq \max_{t \in [a,b]} h(t) \int_a^b g(s)\, ds$$

Therefore, for cont. function

$$F(t) = \int_a^b h(s) g(s)\, ds - h(t) \int_a^b g(s)\, ds$$

there exists $t_0, t_1 \in [a,b]$ with

$$F(t_0) \geq 0 \qquad F(t_1) \leq 0$$

Because cont. function F, we find $\tau \in [a,b]$

$$F(\tau) = 0.$$

$$\Rightarrow \int_a^b h(s) g(s)\, ds = h(\tau) \int_a^b g(s)\, ds$$

Error of trapezoidal rule:

Let $f \in C^2([a,b])$ be twice cont. diff. Then the approximation error of the trap. rule

$$T = \frac{b-a}{2}\left(f(a) + f(b)\right)$$

with step size $h = b-a$ can be expressed as

$$T - \int_a^b f = \frac{h^3}{12} f''(\tau), \qquad \tau \in [a,b].$$

Linear interpolation $P \in \mathbb{P}_1$ satisfies

$$f(t) = P(t) + [a,b,t]f\,(t-a)(t-b)$$

($\rightsquigarrow$ Newton interpolation)

Also know that for $f \in C^2([a,b])$

$$[a,b,t]f = \frac{f''(\tau(t))}{2} \qquad \tau(t) \in [a,b]$$

$$\int_a^b f = \int_a^b P(t)\,dt + \int_a^b [a,b,t]f\,\underbrace{(t-a)(t-b)}_{\substack{\leq 0 \\ a \leq t \leq b}}\,dt$$

Because $g(t) = (t-a)(t-b)$ has one sign over $[a,b]$, $\exists\, \hat{t} \in [a,b]$ such that

$$\int [a,b,t]f\, g(t) = \underline{[a,b,\hat{t}]f} \int g(t)$$

$$\int_a^b f = \underbrace{T}_{\text{trapezoidal rule } \int P(t)\,dt} + \frac{f''(\tau(t))}{2} \underbrace{\int_a^b (t-a)(t-b)\,dt}_{= -\frac{(b-a)^3}{6}}$$

$$T - \int_a^b f = \frac{(b-a)}{12} f''(\tau) \qquad \tau \in [a,b]$$

We use equidistantly spaced nodes?

*almost equal because the f does not change too much (smoothness) from t_i to t_{i+1}

We use equidistantly spaced nodes? We know that this leads to poorly conditioned interpolation problems!

*almost equal because the f does not change too much (smoothness) from t_i to t_{i+1}

We use equidistantly spaced nodes? We know that this leads to poorly conditioned interpolation problems!

Recall: Polynomial interpolation can lead to Runge's phenomenon $\rightsquigarrow$ high frequency oscillations $\rightsquigarrow$ we saw that the relative condition number of numerical integration can increase for oscillatory functions

Another point of view: The weights $\lambda_{n0}, \ldots, \lambda_{nn}$ of Newton-Cotes formulas can become *negative* for larger n $\rightsquigarrow$ cancellation because we subtract "almost equal numbers*" in

$$\hat{I}(f) = (b-a) \sum_{i=0} \lambda_i f(t_i)$$

Weights are positive up to order 7, then some start to become negative.

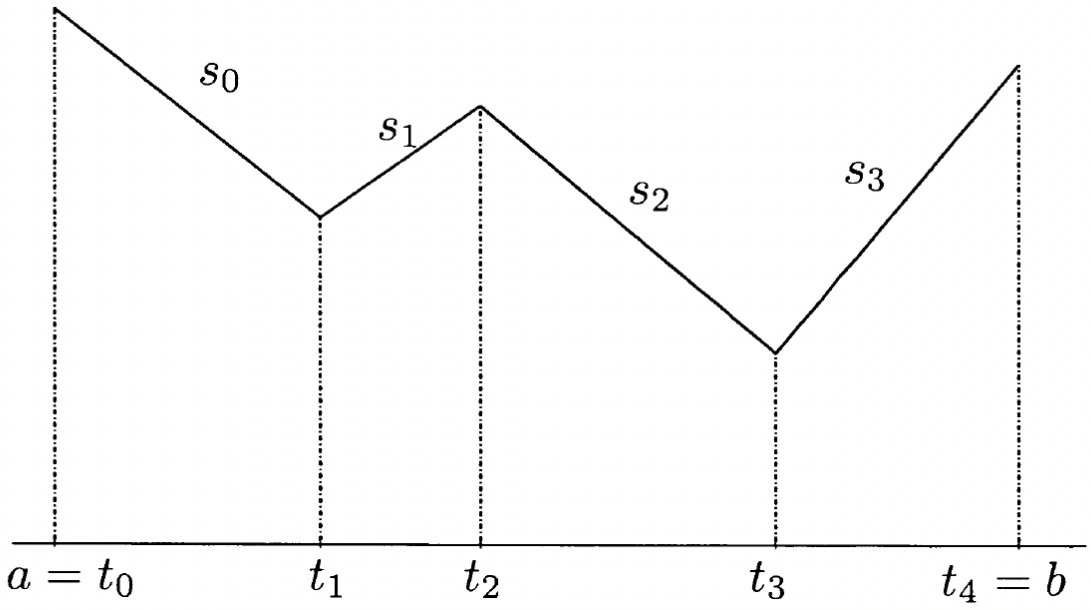
*almost equal because the f does not change too much (smoothness) from t_i to t_{i+1}

Trapezoidal sums

To avoid poorly conditioned problems, let us split the integration interval $[a, b]$ into n sub-intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, n$. Then consider the rule

$$\hat{I}(f) = \sum_{i=1}^{n} \hat{I}_{t_{i-1}}^{t_i}(f),$$

where $\hat{I}_{t_{i-1}}^{t_i}$ is a quadrature formula on the interval $[t_{i-1}, t_i]$.



Same assumptions on f as before. We have seen already the trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

Proof $\rightsquigarrow$ board

Trapezoidal sum

$\hookrightarrow \exists\, \tau_i \in [t_{i-1}, t_i]$ such that

$$T_i - \int_{t_{i-1}}^{t_i} f = \frac{h^3}{12} f''(\tau_i) \quad \Big\} \text{ just showed this}$$

$$T(h) - \int_0^b f = \sum_{i=1}^{n} \left(T_i - \int_{t_{i-1}}^{t_i} f\right)$$

$$= \sum_{i=1}^{n} \frac{h^3}{12} f''(\tau_i)$$

$$= \sum_{i=1}^{n} \frac{h^2}{12} \frac{(b-a)}{n} f''(\tau_i)$$

$$= \frac{(b-a)h^2}{12} \frac{1}{n} \sum_{i=1}^{n} f''(\tau_i)$$

Now get rid of dependence of τ on intervals

$$\min_{t \in [a,b]} f''(t) \leq \frac{1}{n} \sum_{i=1}^{n} f''(\tau_i) \leq \max_{t \in [a,b]} f''(t)$$

Mean value theorem: $\exists\, \tau \in [a,b]$

$$\frac{1}{n} \sum_{i=1}^{n} f''(\tau_i) = f''(\tau)$$

$$\Rightarrow T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)$$

Same assumptions on f as before. We have seen already the trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

Proof $\rightsquigarrow$ board

What did we achieve with this?

Same assumptions on f as before. We have seen already the trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

Proof $\rightsquigarrow$ board

What did we achieve with this? $\rightsquigarrow$ We can increase n (and thus decrease h) to reduce the error without increasing the degree of the underlying polynomial

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last week

- Numerical integration
- Newton-Cotes formulas

Today

- Gauss quadrature
- Integration in multiple dimensions

Announcements

- Homework 6 is posted and due Mon, Dec 2 before class

Recap: Condition of numerical integration

Lets study the map

$$([a,b],f) \to \int_a^b f(t)\,dt,$$

where we use the L^1-norm for f

$$\|f\|_1 = \int_a^b |f(t)|\mathrm{d}t = I(|f|)$$

The absolute and relative condition numbers of integration are:

$$\kappa_{\text{abs}} = 1,$$
$$\kappa_{\text{rel}} = \frac{I(|f|)}{|I(f)|}.$$

What does this mean?

Recap: Condition of numerical integration

Lets study the map

$$([a,b],f) \to \int_a^b f(t)\,dt,$$

where we use the L^1-norm for f

$$\|f\|_1 = \int_a^b |f(t)|\mathrm{d}t = I(|f|)$$

The absolute and relative condition numbers of integration are:

$$\kappa_{\text{abs}} = 1,$$
$$\kappa_{\text{rel}} = \frac{I(|f|)}{|I(f)|}.$$

What does this mean? Integration is harmless w.r.t. the absolute condition number, and problematic w.r.t. the relative condition number if $I(f)$ is small and f changes sign

Recap: Newton-Cotes formulas

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt$$

Recap: Newton-Cotes formulas (cont'd)

Quadrature formulas defined in this way are exact for polynomials $P \in \mathbb{P}_n$ of degree less than or equal to n

$$\hat{I}(P) = I(P_n(P)) = I(P)\,, \qquad \text{for all } P \in \mathbb{P}_n$$

Theorem: For $(n+1)$ pairwise distinct nodes $t_0, \ldots, t_n$, there exists exactly one quadrature formula (i.e., unique weights $\lambda_0, \ldots, \lambda_n$)

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(t_i)\,,$$

that is exact for all $p \in \mathbb{P}_n$.

Recap

Equidistantly spaced nodes

$$h_i = h = \frac{b-a}{n}, \qquad t_i = a + ih, \qquad i = 0, \ldots, n$$

then quadrature formulas are called the *Newton-Cotes formulas* with weights

$$\lambda_{in} = \frac{1}{b-a} \int_a^b \prod_{i \neq j} \frac{t - t_i}{t_i - t_j} \, dt = \frac{1}{n} \int_0^n \prod_{i \neq j} \frac{s - j}{i - j} \, ds$$

These weights are independent of the interval boundaries a and b and can be pre-computed once and for all:

Table 9.1. Newton-Cotes weights λ_{in} for $n = 1, \ldots, 4$.

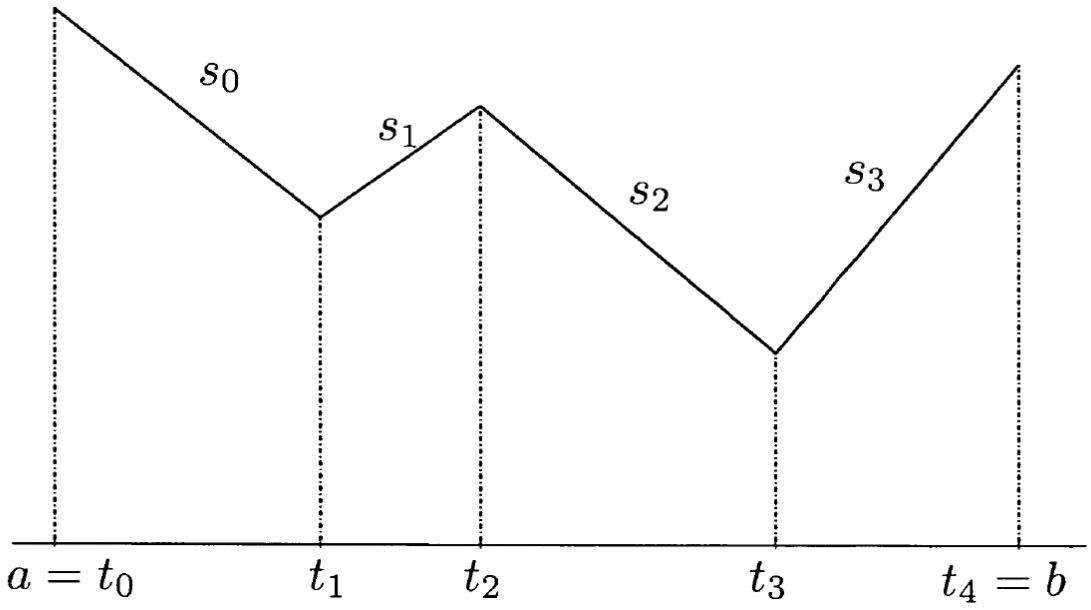
n	$\lambda_{0n}, \ldots, \lambda_{nn}$	Error	Name
1	$\frac{1}{2} \quad \frac{1}{2}$	$\frac{h^3}{12} f''(\tau)$	Trapezoidal rule
2	$\frac{1}{6} \quad \frac{4}{6} \quad \frac{1}{6}$	$\frac{h^5}{90} f^{(4)}(\tau)$	Simpson's rule, Kepler's barrel rule
3	$\frac{1}{8} \quad \frac{3}{8} \quad \frac{3}{8} \quad \frac{1}{8}$	$\frac{3h^5}{80} f^{(4)}(\tau)$	Newton's 3/8-rule
4	$\frac{7}{90} \quad \frac{32}{90} \quad \frac{12}{90} \quad \frac{32}{90} \quad \frac{7}{90}$	$\frac{8h^7}{945} f^{(6)}(\tau)$	Milne's rule

Recap: Trapezoidal sums

To avoid poorly conditioned problems, let us split the integration interval $[a, b]$ into n sub-intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, n$. Then consider the rule

$$\hat{I}(f) = \sum_{i=1}^{n} \hat{I}_{t_{i-1}}^{t_i}(f),$$

where $\hat{I}_{t_{i-1}}^{t_i}$ is a quadrature formula on the interval $[t_{i-1}, t_i]$.



Recap: Error of Trapezoidal sum

Trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

What did we achieve with this?$\rightsquigarrow$

Recap: Error of Trapezoidal sum

Trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

What did we achieve with this?$\rightsquigarrow$ We can increase n (and thus decrease h) to reduce the error without increasing the degree of the underlying polynomial

Gauss-Christoffel quadrature

Besides piecewise approximations, what else could we do to avoid issues with high-degree polynomials?

Gauss-Christoffel quadrature

Besides piecewise approximations, what else could we do to avoid issues with high-degree polynomials? Change the nodes

So far, we allowed to choose weights but the nodes were given (e.g., equidistant). Now, let's allow changing the nodes $t_0, \ldots, t_n$ too.

What highest degree can we hope for?

Gauss-Christoffel quadrature

Besides piecewise approximations, what else could we do to avoid issues with high-degree polynomials? Change the nodes

So far, we allowed to choose weights but the nodes were given (e.g., equidistant). Now, let's allow changing the nodes $t_0, \ldots, t_n$ too.

What highest degree can we hope for? The best we can hope for is exact interpolation up to polynomials of degree $2n+1$ ($2(n+1)$ coefficients) based on a non-rigorous counting argument of having $n+1$ DoFs because of nodes $t_0, \ldots, t_n$ and $n+1$ DoFs because of weights $\lambda_0, \ldots, \lambda_n$.

Also, for generalization, we consider quadrature of weighted integrals, with a positive weight function $\omega(t)$:

$$I(f) = \int_a^b \omega(t) f(t)\, dt$$

with weight functions $\omega(t) = 1, \omega(t) = 1/\sqrt{1-t^2}, \ldots$.

Our goal is the construction of quadrature formulas of the form

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in}),$$

to approximate $I(f)$. Thus, for a given n, we seek $n+1$ notes $\tau_{0n}, \ldots, \tau_{nn}$ and $n+1$ weights $\lambda_{0n}, \ldots, \lambda_{nn}$ so that polynomials up to as high degree N as possible can be integrated exactly

$$\hat{I}_n(P) = I(P), \qquad \text{for all } P \in \mathbb{P}_N$$

Gauss-Christoffel quadrature (cont'd)

Can we say something about the nodes $\tau_{0n}, \ldots, \tau_{nn}$?

If $\hat{I}_n$ is exact for all polynomials $P \in \mathbb{P}_{2n+1}$, then the polynomials $\{P_j\}$ that are defined by their root representation

$$P_{j+1}(t) = (t - \tau_{0j}) \dots (t - \tau_{jj}) \in \mathbb{P}_{n+1}$$

are orthogonal with respect to the scalar product that is induced by w

$$(f, g) = \int_a^b w(t) f(t) g(t) \, dt$$

Proof: $\hat{I}_n$ exact for polynomials in $\mathbb{P}_{2n+1}$:

for $j < n+1$, have $P_j P_{n+1} \in \mathbb{P}_{2n+1}$

$$(P_j, P_{n+1}) = \int w \, P_j P_{n+1} = \hat{I}_n(P_j P_{n+1})$$

$$= \sum_{i=0}^{n} \lambda_{in} P_j(\tau_{in}) \underbrace{P_{n+1}(\tau_{in})}_{=0} = 0$$

because $P_{n+1}(t) = \prod_{i=0}^{n} (t - \tau_{in})$.

Gauss-Christoffel quadrature (cont'd)

Can we say something about the nodes $\tau_{0n}, \ldots, \tau_{nn}$?

Theorem: For $n \in \mathbb{N}$, consider nodes $\tau_{0n}, \ldots, \tau_{nn}$. For an n, define $\hat{I}$ as

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in})$$

and let it be exact for polynomials $p \in \mathbb{P}_{2n+1}$

$$\hat{I}_n(p) = \int_a^b \omega(t) p(t) \mathrm{d}t \,.$$

Then, the polynomials $\{P_n\}$ given by

$$P_{n+1}(t) = (t - \tau_{0n}) \cdot \ldots \cdot (t - \tau_{nn}) \in \mathbb{P}_{n+1}$$

are orthogonal with respect to the scalar product induced by $\omega(t)$ $\rightsquigarrow$ proof

Gauss-Christoffel quadrature (cont'd)

Therefore, the nodes τ_{in} have to be roots of pairwise orthogonal polynomials $\{P_j\}$ of degree $\deg(P_j) = j$

For a given ω, we already know that the set of orthogonal polynomials $\{P_j\}$ is unique (if leading coefficient is 1)

We also know that the roots of these polynomials are real and have to lie in (a, b) $\rightsquigarrow$ proof board

For each ω and $n \in \mathbb{N}$, this gives us a unique set of nodes, namely the roots of the corresponding orthogonal polynomial P_{n+1}

We thus have fixed $n + 1$ degrees of freedom so far...

The orthogonal polynomial $P_k \in \mathbb{P}_k$ possess exactly k simple roots in (a,b).

Let $t_1, \dots, t_m$ be m distinct points in (a,b) of which P_k changes sign. We now show that $m=k$.

Consider the polynomial

$$Q(t) = (t-t_1)(t-t_2) \dots (t-t_m)$$

The polynomial changes sign at the same points in (a,b) as P_k. Thus: $\underbrace{w(t)Q(t)P_k(t)}$ does not change sign

$$(Q, P_k) = \int_a^b w(t)\, Q(t)\, P_k(t)\, dt \neq 0$$

But P_k orthogonal to all $P \in \mathbb{P}_{k-1}$, thus $\deg(Q) = m \geq k$.

What do we know about the weights for fixed $t_0, \ldots, t_n$ when we want to integrate exactly polynomials up to degree n?

What do we know about the weights for fixed $t_0, \ldots, t_n$ when we want to integrate exactly polynomials up to degree n?

Because we want to be able to exactly integrate polynomials up to degree $2n+1$, we already know that to even achieve exactness up to degree n, the weights are fixed for given nodes $t_0, \ldots, t_n$:

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t) \mathrm{d}t \,, \qquad \text{Lagrange poly } L_{in}(\tau_{jn}) = \delta_{ij}$$

("For $n+1$ pairwise distinct nodes, the exists only one quadrature formula that exactly integrate polynomials up to degree n.")

Theorem: Let $\tau_{0n}, \ldots, \tau_{nn}$ be the roots of the $(n+1)$st orthogonal polynomial for the weight ω. Then any quadrature formula $\hat{I}$ is exact for polynomials up to order n if and only if it is exact up to order $2n+1$.

Proof

Let $\tau_{0n}, \dots, \tau_{nn}$ be the roots of the $(n+1)$-st orthogonal polynomial P_{n+1} with weight w. Then any quad rule

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_i f(\tau_{in})$$

satisfies

$$\hat{I}_n \text{ exact on } \mathbb{P}_n \iff \hat{I}_n \text{ exact on } \mathbb{P}_{2n+1}$$

„$\Rightarrow$": Suppose that $\hat{I}_n$ is exact on $\mathbb{P}_n$.

$P \in \mathbb{P}_{2n+1}$, then there exist polynomials

$$Q, R \in \mathbb{P}_n$$

such that

$$P = Q P_{n+1} + R$$

P_{n+1} is orthogonal to $\mathbb{P}_n$ w.r.t. weight w

$$\int w P = \underbrace{\int w Q P_{n+1}}_{=0} + \int w R = \int w R = \hat{I}_n(R)$$

Additionally, with $P_{n+1}(\tau_{in}) = 0$ obtain

$$\hat{I}_n(R) = \sum_{i=0}^{n} \lambda_{in} R(\tau_{in})$$

$$= \sum_{i=0}^{n} \lambda_{in} \left(Q(\tau_{in}) \underbrace{P_{n+1}(\tau_{in})}_{=0} + R(\tau_{in}) \right) = \hat{I}_n(P)$$

„$\Leftarrow$": ...

Weight functions for Gauss-Christoffel quadrature

$\omega(t)$	Interval $I = [a, b]$	Orthogonal polynomials
$\frac{1}{\sqrt{1-t^2}}$	$[-1, 1]$	Chebyshev polynomials T_n
e^{-t}	$[0, \infty]$	Laguerre polynomials L_n
e^{-t^2}	$[-\infty, \infty]$	Hermite polynomials H_n
1	$[-1, 1]$	Legendre polynomials P_n

Corresponding quadrature rules are usually prefixed with "Gauss-", i.e., "Gauss-Legendre quadrature", or "Gauss-Chebyshev quadrature".

Summary of Gauss quadrature

There exist uniquely determined nodes $\tau_{0n}, \ldots, \tau_{nn}$ and weights $\lambda_{0n}, \ldots, \lambda_{nn}$ such that the quadrature formula

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in})$$

integrates exactly all polynomials of degree less than or equal to $2n+1$, i.e.,

$$\hat{I}_n(P) = \int_a^b \omega P\,, \qquad P \in \mathbb{P}_{2n+1}\,.$$

The nodes τ_{in} are the roots of the $n+1$-st orthogonal polynomial P_{n+1} with respect to the weight function ω and the weights are

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\mathrm{d}t\,,$$

with the Lagrange polynomials $L_{in}(\tau_{jn}) = \delta_{ij}$. Furthermore, the weights are all positive $\lambda_{in} > 0$. $\rightsquigarrow$ proof

Unique nodes $\tau_{0n}, \dots, \tau_{nn}$ and weights $\lambda_{0n}, \dots, \lambda_{nn}$ such that

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in})$$

integrates exactly $P \in \mathbb{P}_{2n+1}$. The weights are positive.

Let $Q \in \mathbb{P}_{2n+1}$ be a polynomial so that τ_{kn} is the only node at which it doesnot vanish:

$$Q(\tau_{kn}) \neq 0, \quad Q(\tau_{in}) = 0 \quad i \neq k$$

Then

$$\int_a^b w Q = \hat{I}_n(Q) = \lambda_{kn} Q(\tau_{kn})$$

and so $\lambda_{kn} = \frac{1}{Q(\tau_{kn})} \int w Q.$

if we set

$$Q(t) = \left(\frac{P_{n+1}(t)}{(t - \tau_{kn})} \right)^2$$

then $Q \in \mathbb{P}_{2n}$ has these properties

Weights

$$\lambda_{kn} = \frac{1}{Q(\tau_{kn})} \int_a^b w Q = \int w \left(\frac{P_{n+1}(t)}{P'_{n+1}(\tau_{kn})(t - \tau_{kn})} \right)^2 dt > 0$$

$$Q(\tau_{kn}) = [P'_{n+1}(\tau_{kn})]^2$$

Approximation error

For any function $f \in C^{2n+2}$, the approximation error of Gauss quadrature can be expressed in the form

$$\int_a^b \omega f - \hat{I}_n(f) = \frac{f^{(2n+2)}(\tau)}{(2n+2)!}(P_{n+1}, P_{n+1}),$$

with some $\tau \in [a, b]$.

$\rightsquigarrow$ proof

For any function $f \in C^{2n+2}$, the approximation error of Gauss quadrature can be expressed as

$$\int_a^b w f - \hat{I}_n(f) = \frac{f^{(2n+2)}(\tau)}{(2n+2)!} (P_{n+1}, P_{n+1})$$

with some $\tau \in [a,b]$, P_{n+1} ortho. poly.

We employ the Newton remainder of the Hermite interpolant $P \in \underline{\mathbb{P}_{2n+1}}$ of f for the nodes

$$\tau_{0n}, \tau_{0n}, \dots, \tau_{nn}, \tau_{nn}$$

so that

$$f(t) = P(t) + [t, \tau_{0n}, \tau_{0n}, \dots, \tau_{nn}, \tau_{nn}] f \, (t-\tau_{0n})^2 \dots (t-\tau_{nn})^2$$

Notice that

$$(t-\tau_{0n})^2 \dots (t-\tau_{nn})^2 = P_{n+1}^2$$

Because $\hat{I}_n$ integrates P exactly, we have

$$\int w f = \underbrace{\int w P}_{} + \frac{f^{(2n+2)}(\tau)}{(2n+2)!} \int w \, P_{n+1}^2$$

$$= \underbrace{\sum_{i=0}^{n} \lambda_{in} \underbrace{P(\tau_{in})}_{= f(\tau_{in})}}_{\hat{I}_n(f)} + \frac{f^{(2n+2)}(\tau)}{(2n+2)!} (P_{n+1}, P_{n+1})$$

Summary of Gauss quadrature (cont'd)

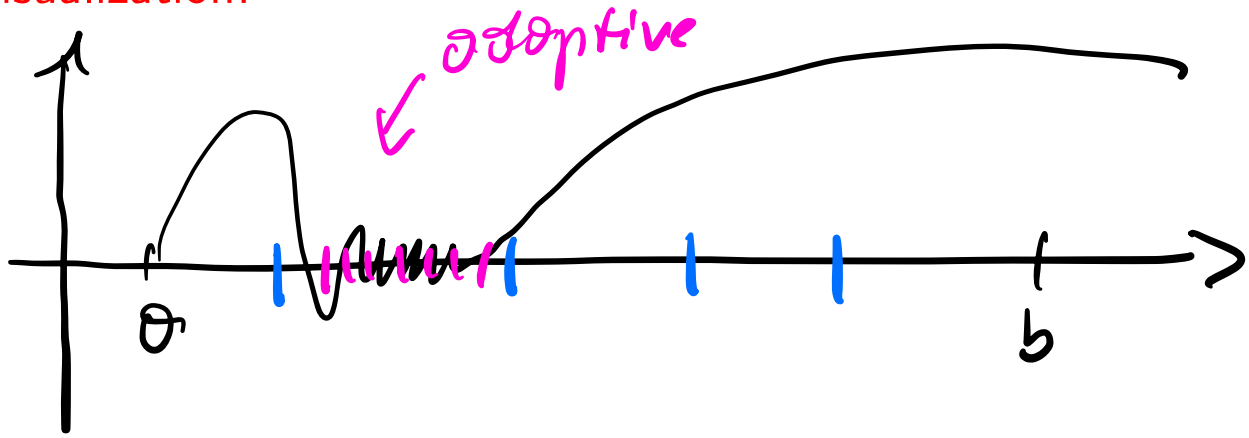
- Gauss quadrature gives the highest degree $2n+1$ of polynomials that can be exactly integrated with $n+1$ function evaluations in general
- Accuracy in Gauss-(Chebyshev, Laguerre, Hermite, Legendre,...) w.r.t. to polynomial degree can only be improved by increasing number of points; not by better weights
- Of particular interest are quadrature points for infinite intervals (Laguerre, Hermite)
- Interval partitioning superior, but only possible for $\omega \equiv 1$ (Gauss-Legendre); otherwise weight function is different in each sub-interval
- Gauss quadrature is typically used in finite-element approximation to integrate over a local element. However, in many other cases in scientific computing, interval partitioning is superior.

Adaptive interval partitioning

Idea: On each sub-interval, estimate the quadrature error by either:

- Using a higher-order quadrature (e.g., Simpson rule), or
- Comparing the error on a subinterval with the error on a refinement

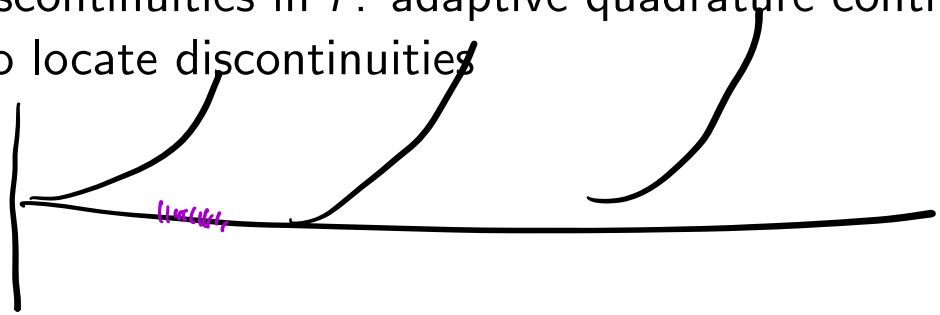
Then, subdivide the interval depending on the error estimation, and repeat $\rightsquigarrow$ visualization!



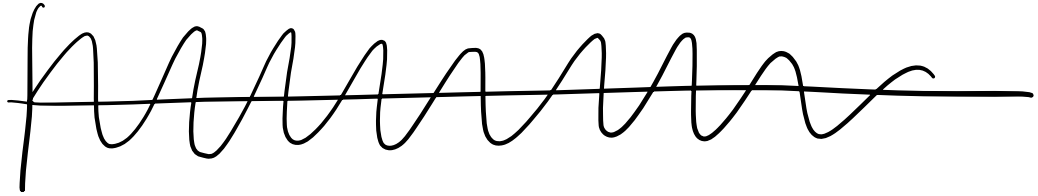
Such a method refines the nodes a posteriori (after having seen the function to integrate).

Difficult cases for numerical integration

- (Unknown) discontinuities in f: adaptive quadrature continues to refine, which can be used to locate discontinuities



- Highly oscillating integrals



- (Weakly) singular integrals

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Today

Last time

- Interpolation and quadrature in one dimension

Today

- Function approximation and interpolation in higher dimensions
- Quadrature in higher dimensions

Announcements

- Homework 6 is due Mon, Dec 2 before class

Recap: Newton-Cotes formulas

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt$$

Recap

Equidistantly spaced nodes

$$h_i = h = \frac{b-a}{n}, \qquad t_i = a + ih, \qquad i = 0, \ldots, n$$

then quadrature formulas are called the *Newton-Cotes formulas* with weights

$$\lambda_{in} = \frac{1}{b-a} \int_a^b \prod_{i \neq j} \frac{t - t_i}{t_i - t_j} \, dt = \frac{1}{n} \int_0^n \prod_{i \neq j} \frac{s - j}{i - j} \, ds$$

These weights are independent of the interval boundaries a and b and can be pre-computed once and for all:

Table 9.1. Newton-Cotes weights λ_{in} for $n = 1, \ldots, 4$.

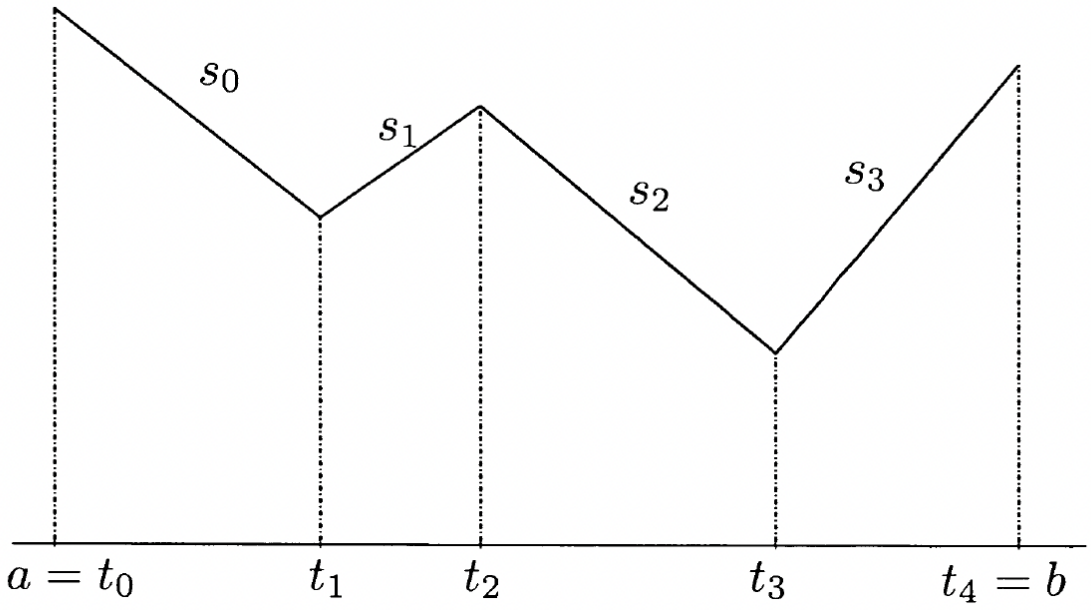
n	$\lambda_{0n}, \ldots, \lambda_{nn}$	Error	Name
1	$\frac{1}{2} \quad \frac{1}{2}$	$\frac{h^3}{12} f''(\tau)$	Trapezoidal rule
2	$\frac{1}{6} \quad \frac{4}{6} \quad \frac{1}{6}$	$\frac{h^5}{90} f^{(4)}(\tau)$	Simpson's rule, Kepler's barrel rule
3	$\frac{1}{8} \quad \frac{3}{8} \quad \frac{3}{8} \quad \frac{1}{8}$	$\frac{3h^5}{80} f^{(4)}(\tau)$	Newton's 3/8-rule
4	$\frac{7}{90} \quad \frac{32}{90} \quad \frac{12}{90} \quad \frac{32}{90} \quad \frac{7}{90}$	$\frac{8h^7}{945} f^{(6)}(\tau)$	Milne's rule

Recap: Trapezoidal sums

To avoid poorly conditioned problems, let us split the integration interval $[a, b]$ into n sub-intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, n$. Then consider the rule

$$\hat{I}(f) = \sum_{i=1}^{n} \hat{I}_{t_{i-1}}^{t_i}(f),$$

where $\hat{I}_{t_{i-1}}^{t_i}$ is a quadrature formula on the interval $[t_{i-1}, t_i]$.



Recap: Error of Trapezoidal sum

Trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

What did we achieve with this?$\rightsquigarrow$

Recap: Error of Trapezoidal sum

Trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

It has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

What did we achieve with this?$\rightsquigarrow$ We can increase n (and thus decrease h) to reduce the error without increasing the degree of the underlying polynomial

Summary of Gauss quadrature

There exist uniquely determined nodes $\tau_{0n}, \ldots, \tau_{nn}$ and weights $\lambda_{0n}, \ldots, \lambda_{nn}$ such that the quadrature formula

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in})$$

integrates exactly all polynomials of degree less than or equal to $2n+1$, i.e.,

$$\hat{I}_n(P) = \int_a^b \omega P\,, \qquad P \in \mathbb{P}_{2n+1}\,.$$

The nodes τ_{in} are the roots of the $n+1$-st orthogonal polynomial P_{n+1} with respect to the weight function ω and the weights are

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\mathrm{d}t\,,$$

with the Lagrange polynomials $L_{in}(\tau_{jn}) = \delta_{ij}$. Furthermore, the weights are all positive $\lambda_{in} > 0$.

Integration in higher dimensions

Integration in higher dimensions

A separable integral can be integrated dimension-wise:

$$I_L = \int_a^b \int_a^b \phi(x,y)\mathrm{d}x\mathrm{d}y = \int_a^b \phi^{(x)}(x)\mathrm{d}x \int_a^b \phi^{(y)}(y)\mathrm{d}x\,,$$

where

$$\phi(x,y) = \phi^{(x)}(x)\phi^{(y)}(y) \tag{1}$$

Recall that one idea of numerical quadrature is to replace f with an easy-to-integrate $\hat{f}$

- Choose a basis $\phi_1(x,y),\ldots,\phi_n(x,y)$ and approximate

  $$f(x,y) \approx \sum_{i=1}^{n} c_i\phi_i(x,y)\,,$$

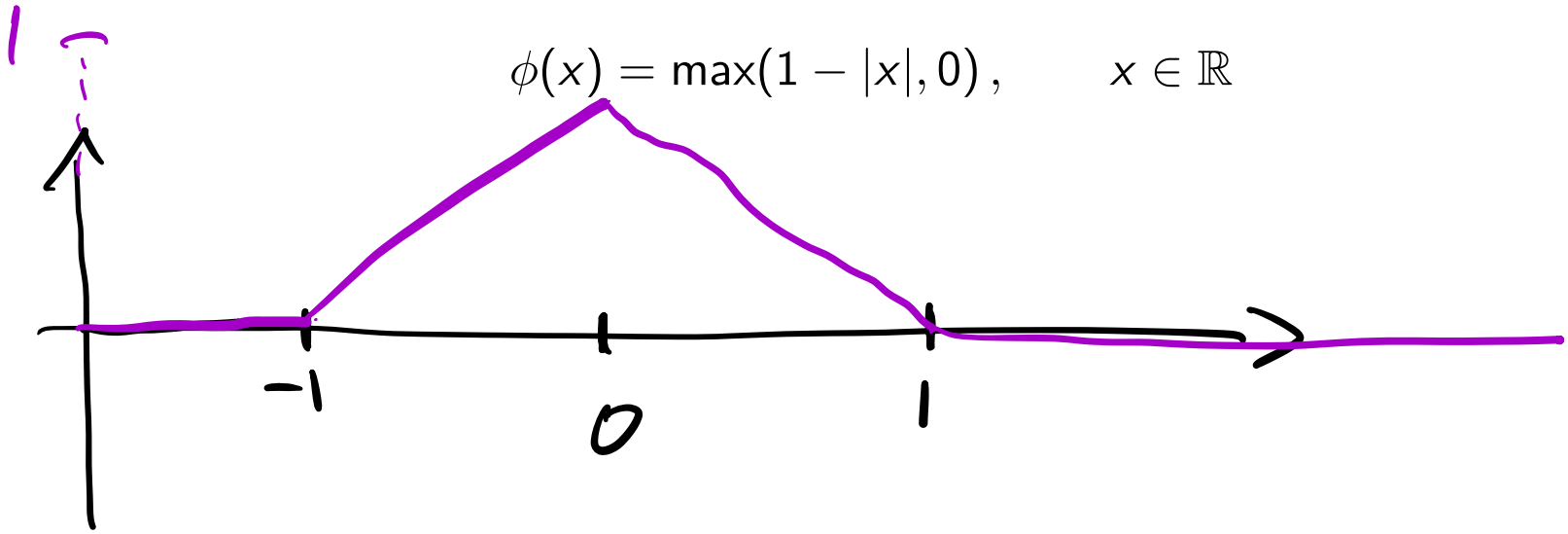
  and choose $\phi_i(x,y)$ such that (1) holds.
- Then integrate

  $$I(f) \approx \sum_{i=1}^{n} c_i \hat{I}(\phi_i^{(x)})\hat{I}(\phi_i^{(y)})$$

One way to build such a basis is via tensor products of linear functions (multi-dimensional analog to piecewise linear quadrature)

Define the "mother hat" function

$$\phi(x) = \max(1 - |x|, 0)\,, \qquad x \in \mathbb{R}$$



If we want $n = 2^l + 1$ basis functions, then translate and dilate it to center it on the grid points

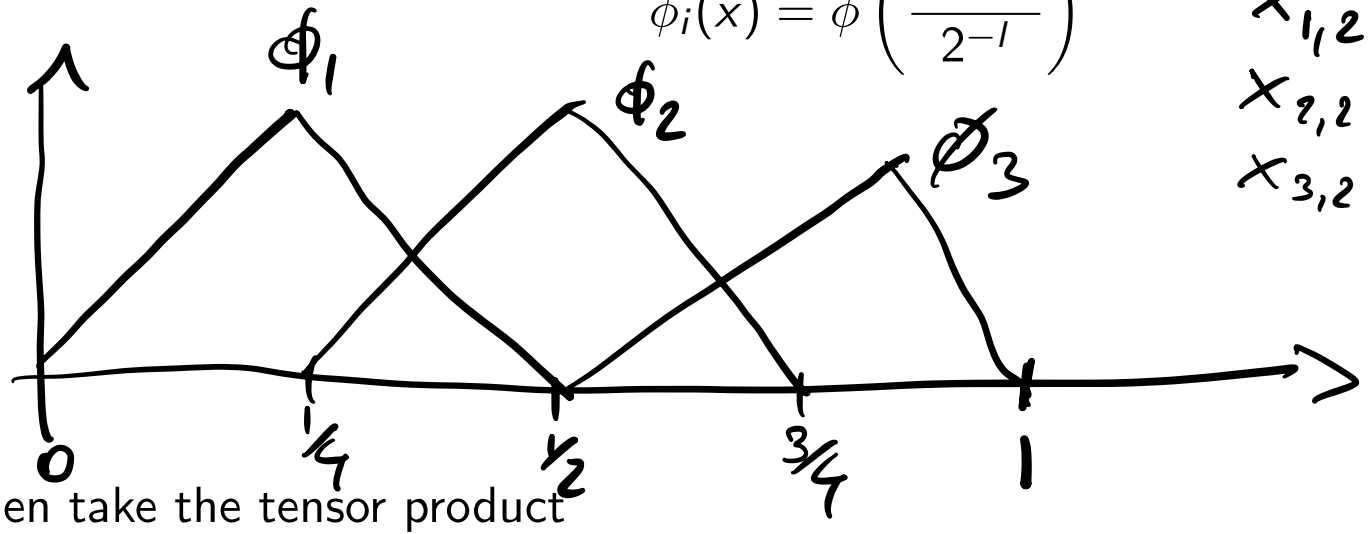
$$x_i = i2^{-l}\,, \qquad i = 1, \ldots, n-1$$

$$\phi_i(x) = \phi\left(\frac{x - x_i}{2^{-l}}\right)$$

$x_{1,2} = 1 \times 2^{-2} = 1/4$

$x_{2,2} = 2 \times 2^{-2} = 1/2$

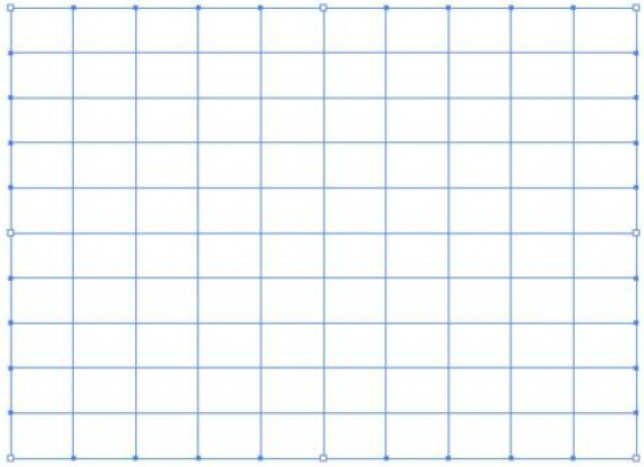
$x_{3,2} = 3 \times 2^{-2} = 3/4$

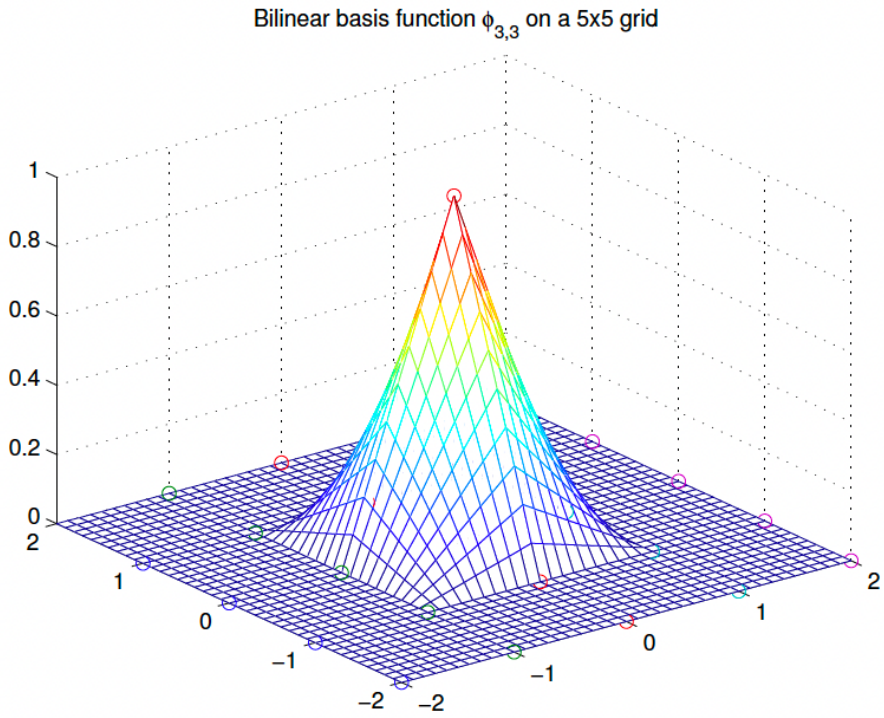


Then take the tensor product

$$\phi_{i,j}(x, y) = \phi_i(x)\phi_j(y)\,, \qquad i, j = 1, \ldots, n-1$$

$\rightsquigarrow$ lead to a basis of the piecewise bilinear functions in two dimensions





[Figure: A. Donev]

What type of basis is this?

What type of basis is this? The basis $\{\phi_{ij}\}$ is a nodal point basis for piecewise bilinear functions in $[0,1]^2 \rightsquigarrow$

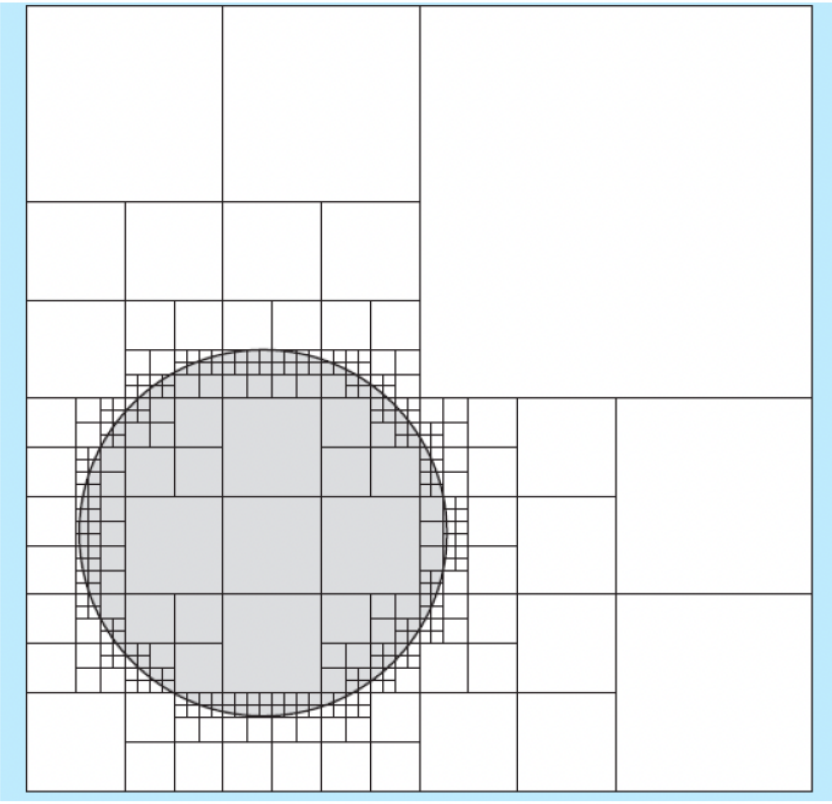
$$\hat{f}(x,y) = \sum_{i,j} f(x_i, y_j)\phi_{ij}(x,y)$$

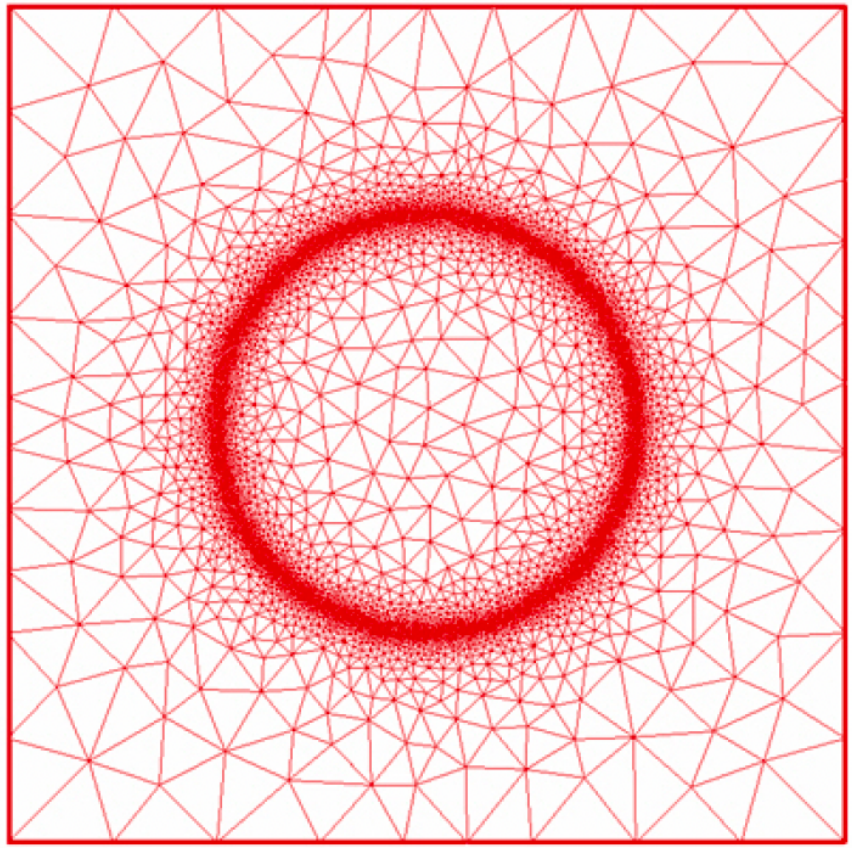
interpolates f at the nodes $\{(x_i, y_j)\}_{i,j=0}^{n}$

Integrate the bilinear basis functions $\{\phi_{ij}\}$ to obtain the weights λ_{ij} and then approximate

$$\hat{I}(f) = \sum_{i,j} f(x_i, y_i)\lambda_{ij}\,,$$

In multiple dimensions, adaptivity is essential to keep the computational costs manageable





[Figure: A. Donev]

Sparse grids

Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

- Consider a simple uniform grid over the domain $\Omega = [0,1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

- Consider a simple uniform grid over the domain $\Omega = [0, 1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

- The curse can be circumvented (to some extent) if?

Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

- Consider a simple uniform grid over the domain $\Omega = [0,1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

- The curse can be circumvented (to some extent) if? if we make stronger assumptions on the functions to approximate $\rightsquigarrow$ topic of today

Sparse grids*

*Follows lecture by H.-J. Bungartz. See also, Bungartz, Griebel, *Sparse grids*, Acta Numerica, 2004

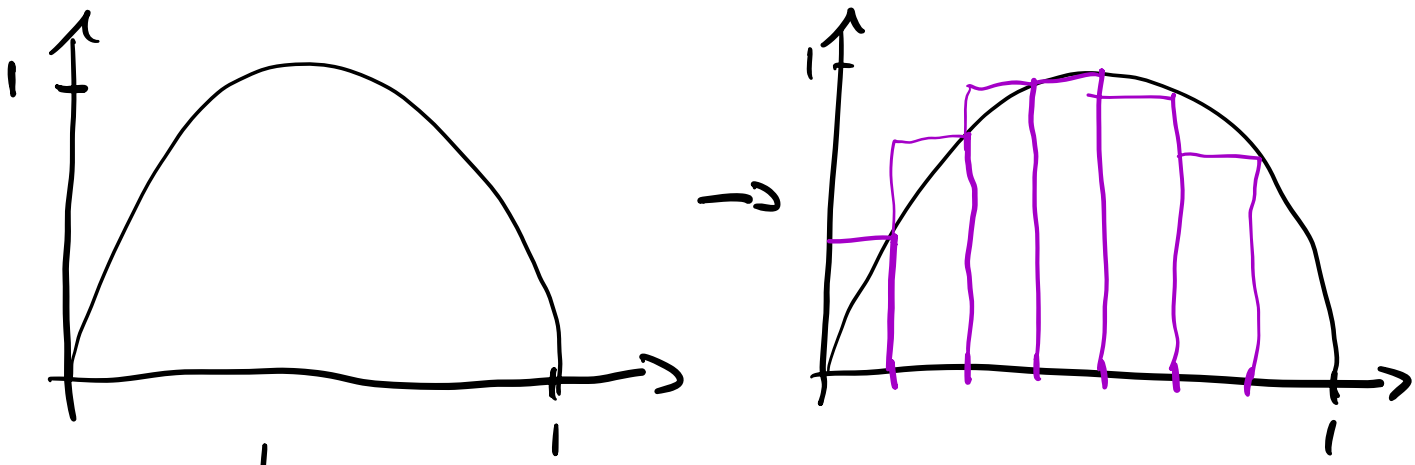
SG: Motivating example

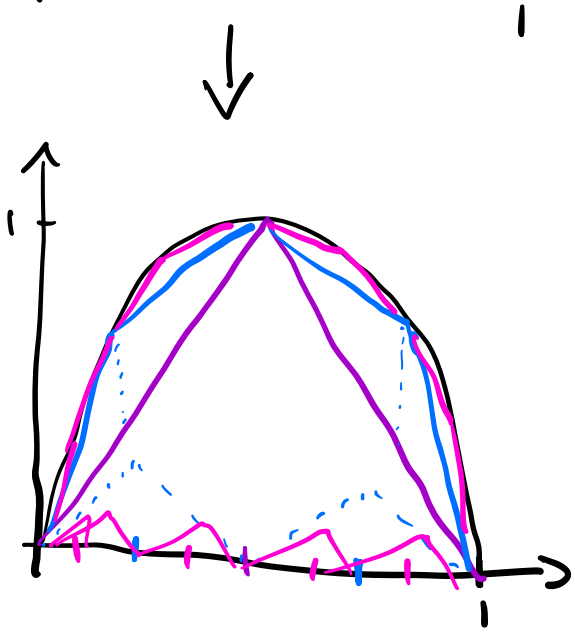
Approximate the integral

$$\int_0^1 4x(1-x)\mathrm{d}x = \frac{2}{3}$$

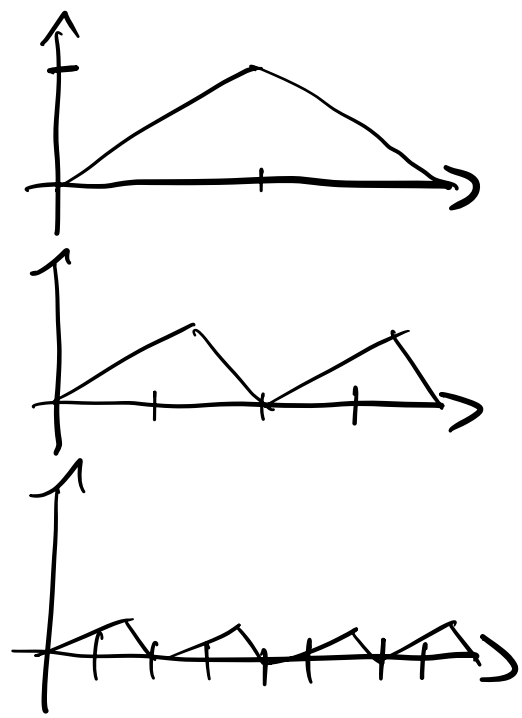
Board!

$$\int 4x(1-x)\,dx = \frac{2}{3}$$





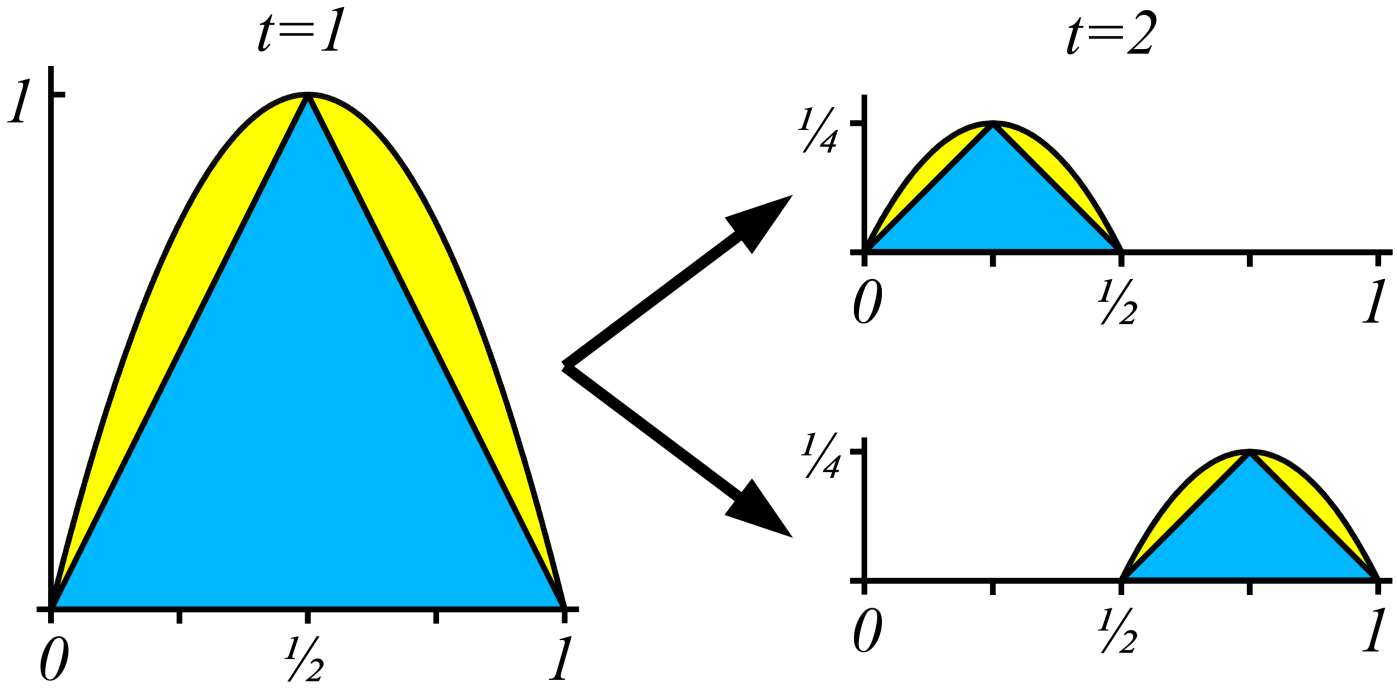
multi-level



SG: Hierarchical decomposition 1D

Archimedes Quadrature

$$\int_0^1 4x(1-x)\mathrm{d}x = \frac{2}{3}$$



depth	1	2	3	4	...	t
interval length h	1/2	1/4	1/8	1/16	...	2^{-t}
number of triangles	1	2	4	8	...	$\frac{1}{2}2^{t}$
surplus v	1	1/4	1/16	1/64	...	$4 \cdot 2^{-2t}$
triangle area D_1	1/2	1/16	1/128	1/1024	...	$4 \cdot 2^{-3t}$
sum (of this t)	1/2	1/8	1/32	1/128	...	$2 \cdot 2^{-2t}$
overall sum ($\leq t$)	1/2	5/8	21/32	85/128	...	$\frac{2}{3}\left(1 - 2^{-2t}\right)$
error	1/6	1/24	1/96	1/384	...	$\frac{2}{3}2^{-2t}$

What do we observe with respect to depth t?

depth	1	2	3	4	...	t
interval length h	1/2	1/4	1/8	1/16	...	2^{-t}
number of triangles	1	2	4	8	...	$\frac{1}{2}2^{t}$
surplus v	1	1/4	1/16	1/64	...	$4 \cdot 2^{-2t}$
triangle area D_1	1/2	1/16	1/128	1/1024	...	$4 \cdot 2^{-3t}$
sum (of this t)	1/2	1/8	1/32	1/128	...	$2 \cdot 2^{-2t}$
overall sum ($\leq t$)	1/2	5/8	21/32	85/128	...	$\frac{2}{3}\left(1 - 2^{-2t}\right)$
error	1/6	1/24	1/96	1/384	...	$\frac{2}{3}2^{-2t}$

What do we observe with respect to depth t? Contribution (e.g., "sum (of this t)") goes down with rate 2^{-2t}

SG: Approximation of functions

Analyze this approach in more general context.

Let $\phi_1, \ldots, \phi_n$ be basis functions and represent

$$u(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$$

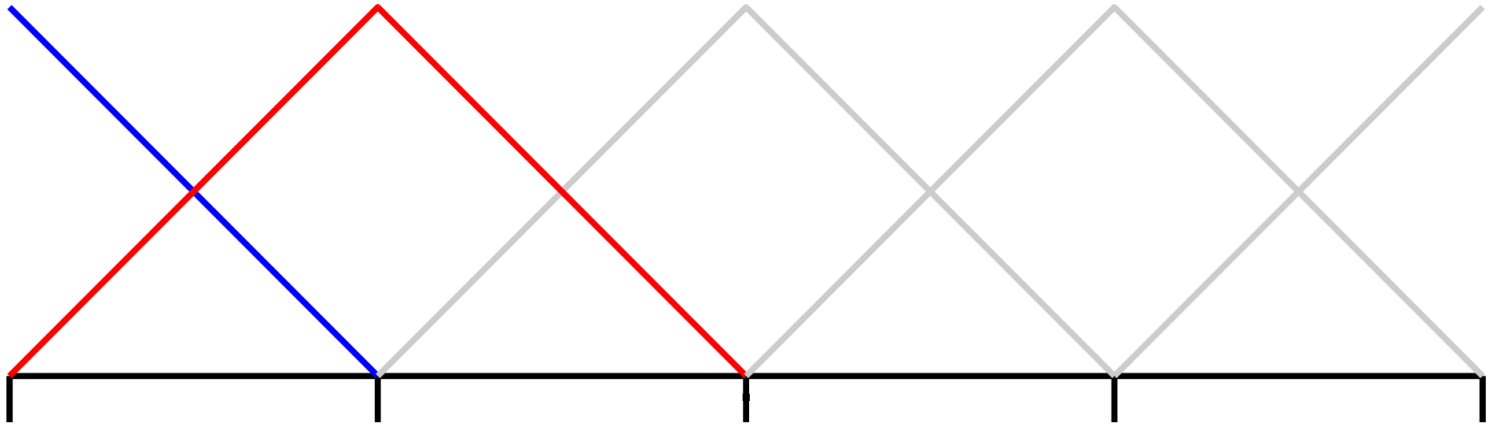
Obtain

$$\int_a^b u(x)\mathrm{d}x = \sum_{i=1}^{n} \alpha_i \int_a^b \phi_i(x)\mathrm{d}x\,,$$

i.e., a quadrature rule as a weighted sum of the coefficients $\alpha_1, \ldots, \alpha_n$

SG: Approximation of functions (cond't)

Represent a continuous, piecewise linear u in nodal point basis



The coefficient $\alpha_1, \ldots, \alpha_n$ are the function values of u at the nodal points

$$u(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$$

$\Rightarrow$ instead of nodal point basis, consider a **hierarchical** basis

SG: Piecewise linear functions

Consider only functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$ in the following.

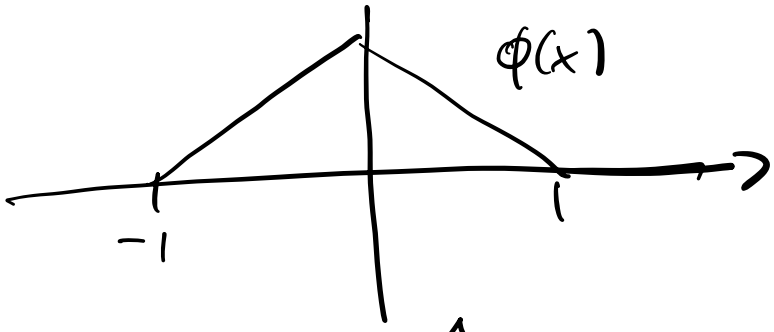
Need the following quantities

- mesh width $h_l = 2^{-l}$
- grid points $x_{l,i} = ih_l = i2^{-l}$
- Basis function

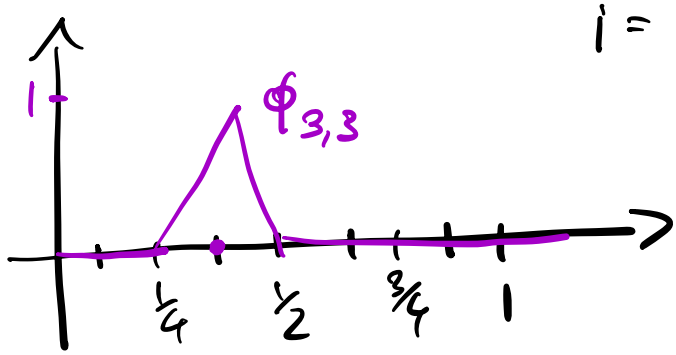
$$\phi_{l,i}(x) = \phi\left(\frac{x - x_{l,i}}{h_l}\right), \qquad \phi(x) = \max\{1 - |x|, 0\}$$

- Nodal point basis $\Phi_l = \{\phi_{l,i} \,:\, 1 \leq i < 2^l\}$

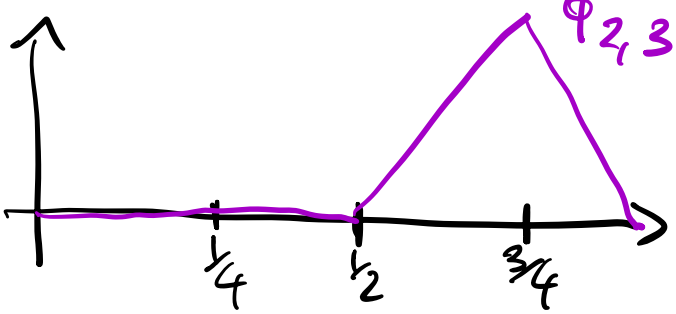
Visualize these basis functions on the board!



$\ell = 3$
$i = 3$



$\ell = 2$ $i = 3$



SG: Piecewise linear functions

Consider only functions $u : [0,1] \to \mathbb{R}$ with $u(0) = u(1) = 0$ in the following.

Need the following quantities

- mesh width $h_l = 2^{-l}$
- grid points $x_{l,i} = ih_l = i2^{-l}$
- Basis function

$$\phi_{l,i}(x) = \phi\left(\frac{x - x_{l,i}}{h_l}\right), \qquad \phi(x) = \max\{1 - |x|, 0\}$$

- Nodal point basis $\Phi_l = \{\phi_{l,i} \,:\, 1 \leq i < 2^l\}$

Visualize these basis functions on the board!

The space $V_l = \text{span}(\Phi_l)$ is the space of piecewise linear, continuous functions with respect to the grid points $x_{l,i}$ for $i = 1, \ldots, 2^l - 1$

SG: Hierarchical basis

For the hierarchical representation, we consider the hierarchical increment W_l, spanned by the basis functions $\phi_{l,i}$ such that

$$V_l = V_{l-1} \oplus W_l$$

is a direct sum (each $u_l \in V_l$ can be uniquely decomposed as $u_l = u_{l-1} + w_l$ with $u_{l-1} \in V_{l-1}$ and $w_l \in W_l \rightarrow$ remember the triangles in approach by Archimedes)

Because $\dim(V_l) = 2^l - 1$ and $\dim(V_l) = \dim(V_{l-1}) + \dim(W_l)$ we need $\dim(W_l) = 2^{l-1}$. These are given by $\phi_{l,i}$ with

$$i \in I_l = \{j \,:\, 1 \leq j < 2^l\,, \quad j \text{ odd}\}$$

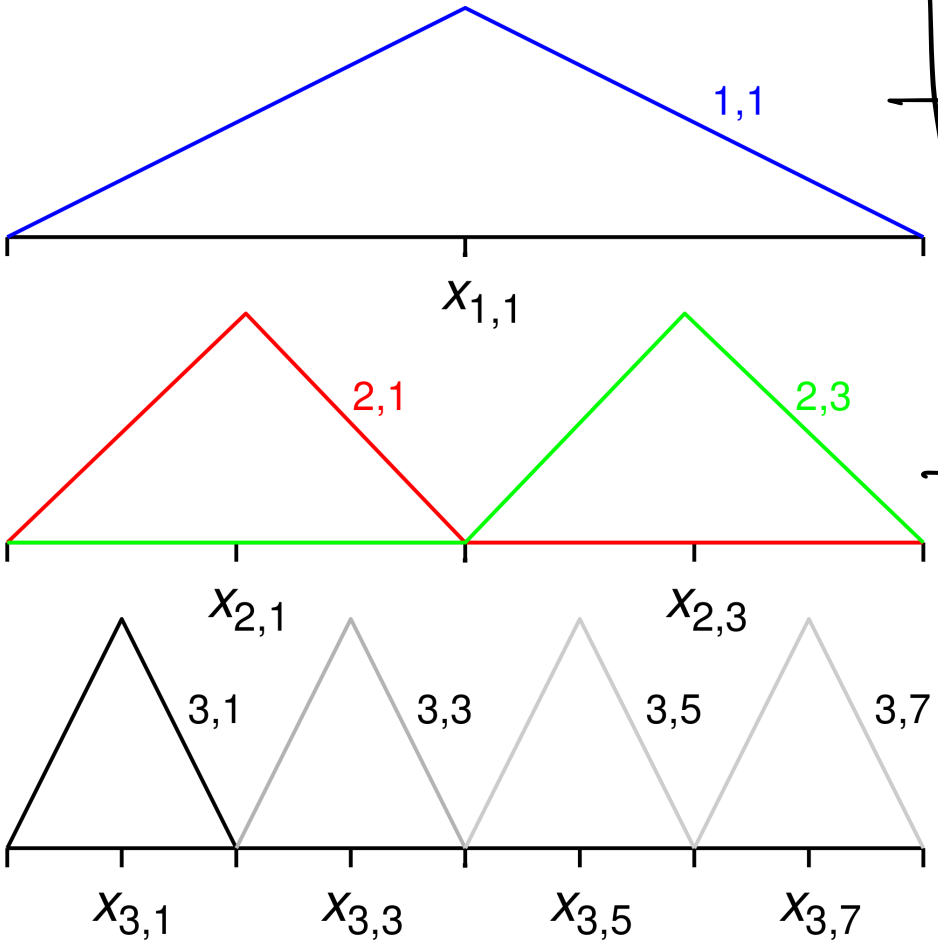
Then

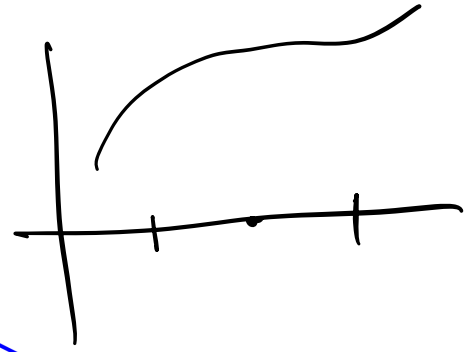
$$W_l = \text{span}\{\phi_{l,i}\,, i \in I_l\}$$

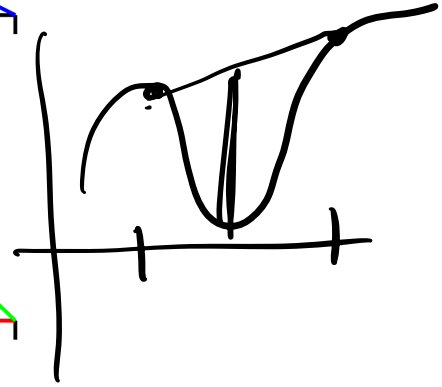
Note that $W_1 = V_1$

SG: Hierarchical basis cont'd

The bases for spaces W_1, W_2 and W_3







SG: Hierarchical basis cont'd

Obtain

$$V_n = \bigoplus_{l=1}^{n} W_l \,,$$

so that there is a unique representation for each $u \in V_n$ as

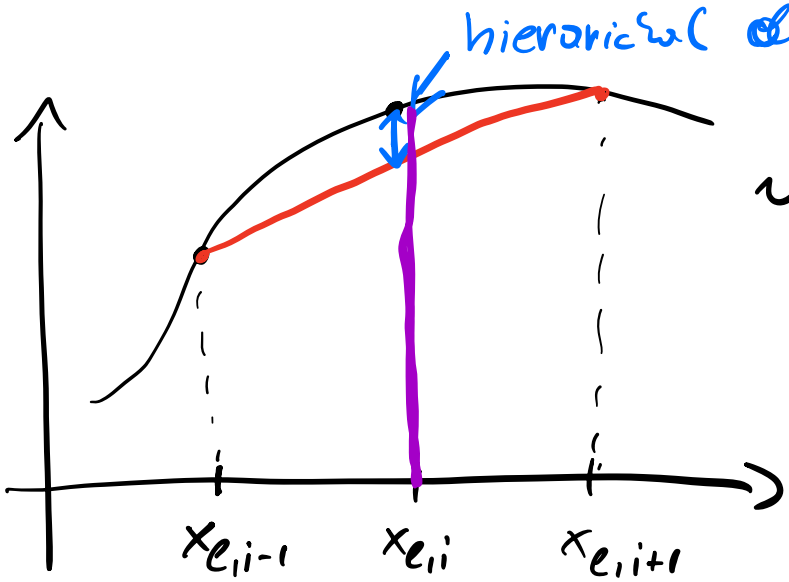
$$u = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

Coefficients $v_{l,i}$ in case of interpolation $\rightsquigarrow$ visualize on board

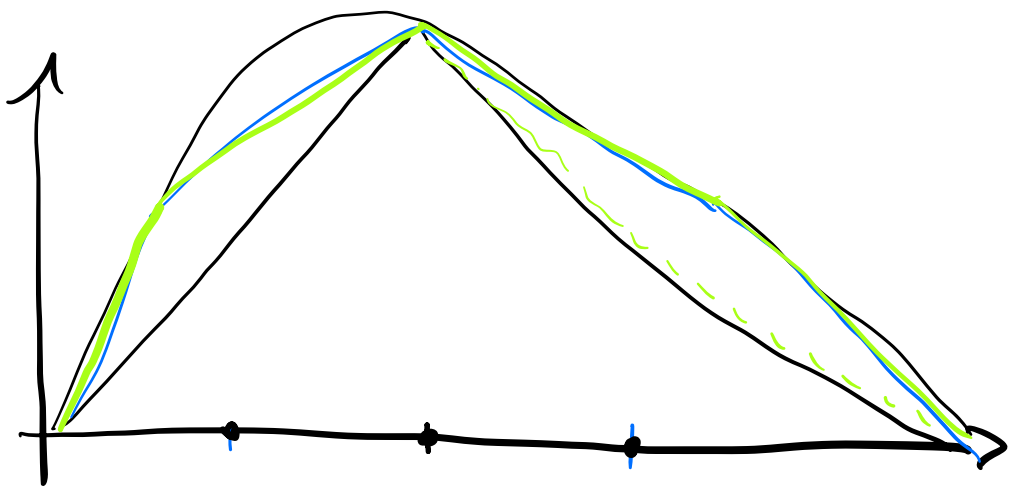
The coefficients $v_{l,i}$ are hierarchical differences

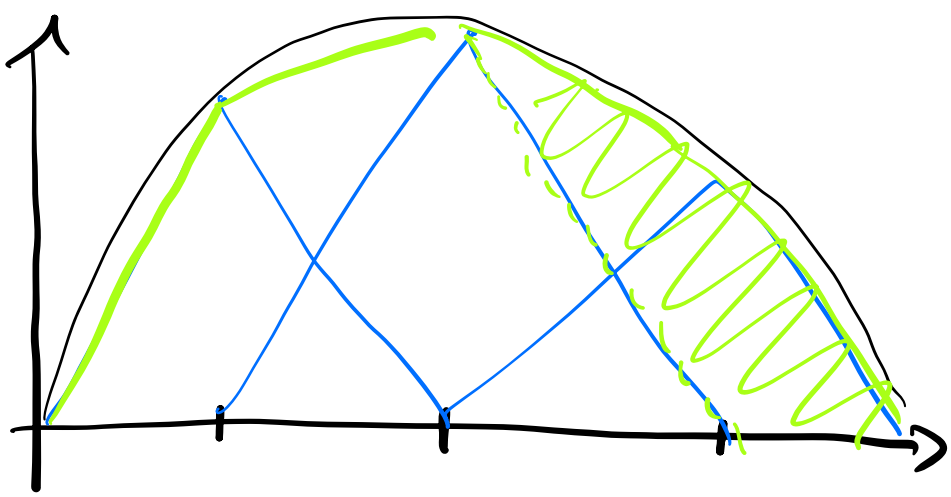
$$v_{l,i} = u^*(x_{l,i}) - \frac{u(x_{l,i-1}) + u(x_{l,i+1})}{2}$$

where u^* is the function to be interpolated



$$v_{e,i} = u^*(x_{e,i}) - \frac{u(x_{e,i-1}) + u(x_{e,i+1})}{2}$$





SG: Analysis of hierarchical decomposition

We now analyze the hierarchical decomposition

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

when interpolating a u. What would we like to obtain?

SG: Analysis of hierarchical decomposition

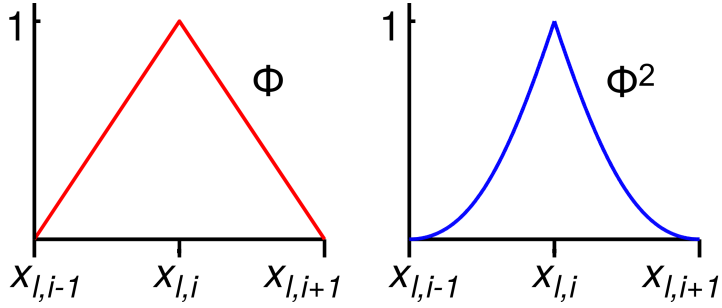
We now analyze the hierarchical decomposition

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

when interpolating a u. What would we like to obtain?

For our decomposition, we first calculate the norms of the basis functions

$$\|\phi_{l,i}\|_\infty = 1\,, \qquad \|\phi_{l,i}\|_2 = \sqrt{\frac{2h_l}{3}}$$



Show decay rate of

$$w_e = \sum_{i \in I_e} v_{e,i} \, \phi_{e,i}$$

Represent coefficients as [BG, Lemma 3.2]

$$v_{e,i} = \int_0^1 \varphi_{e,i}(x) \frac{\partial^2}{\partial x^2} u(x) \, dx$$

↳ $\varphi_{e,i} = -\frac{h_e}{2} \phi_{e,i}$

↳ requires twice differentiable u !!!

$\Rightarrow$ strong smoothness assumption

Now bound coefficients

$$\underline{|v_{e,i}|} \leq \frac{h_e}{2} \underbrace{\|\phi_{e,i}\|_2}_{\leq \sqrt{\frac{2}{3} h_e}} \left\| \frac{\partial^2}{\partial x^2} u_{|T_i} \right\|_2$$

$$\leq \sqrt{\frac{h_e^3}{6}} \left\| \frac{\partial^2}{\partial x^2} u_{|T_i} \right\|_2$$

where $u_{|T_i}$ is u restricted to support

$T_i = [x_{e,i-1}, x_{e,i+1}]$ of $\phi_{e,i}$

Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Today

Last time

- Quadrature in higher dimensions
- Sparse grids

Today

- Sparse grids

Announcements

- Homework 6 is due Mon, Dec 2 before class
- Wed, Nov 27 will be a Q&A session. No new material will be discussed. All notes will be posted online. Send me questions via email by Tue, Nov 26, 4.55pm.

Recap: Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

- Consider a simple uniform grid over the domain $\Omega = [0, 1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

Recap: Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

- Consider a simple uniform grid over the domain $\Omega = [0, 1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

- The curse can be circumvented (to some extent) if?

Recap: Curse of dimensionality

- The curse of dimensionality is a term coined by Bellmann (1961) that refers to an exponential increase of costs with the dimension of a problem

- For example, consider an approximation with a prescribed accuracy $\epsilon > 0$, let the costs of achieving this approximation scale as $\mathcal{O}(\epsilon^{-d})$ in d dimensions $\rightsquigarrow$ exponential increase of the costs as we increase d

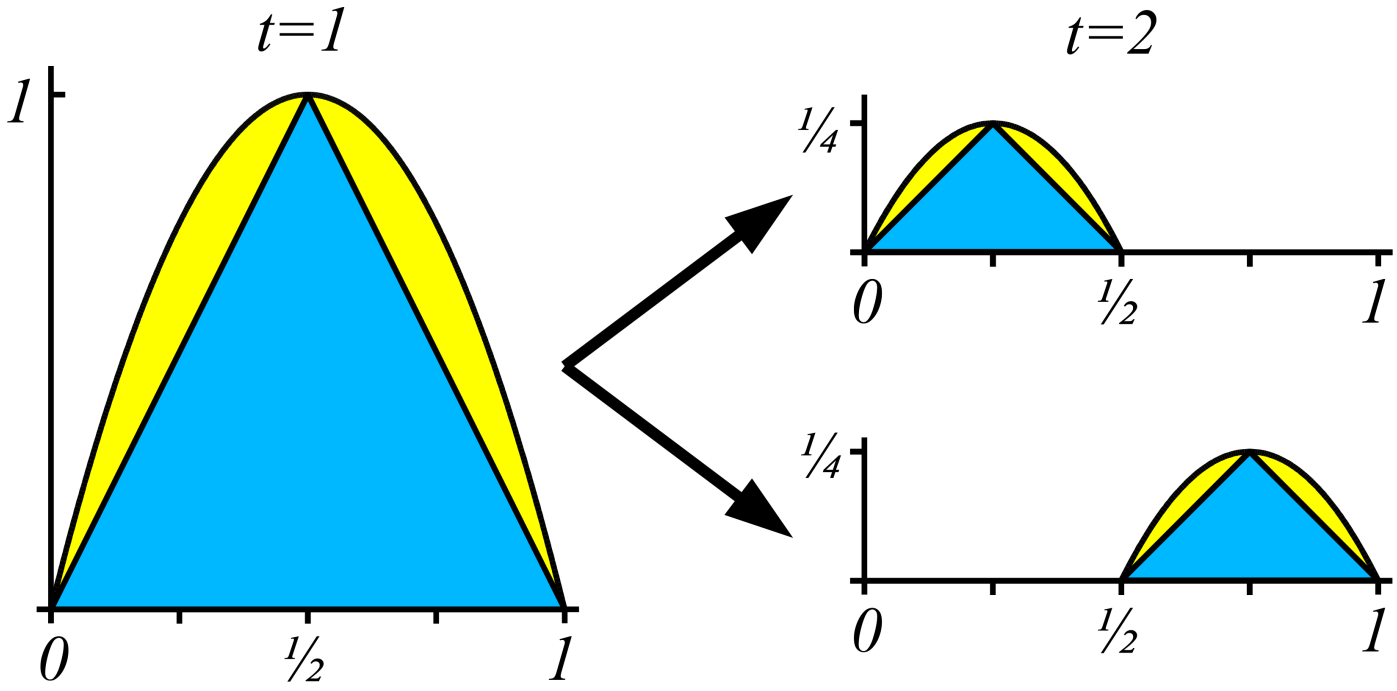
- Consider a simple uniform grid over the domain $\Omega = [0,1]^d$. To have a mesh with mesh width $h = 1/9$ in $d = 1$, we need $N = 10$ grid points. In $d = 2$ dimensions, need $N^2 = 100$ grid points. In $d = 5$ dimensions, need $N^5 = 10^5$ $\rightsquigarrow$ exponential growth of cost and storage requirements as we increase dimension d while keeping mesh width h ("accuracy") fixed

- The curse can be circumvented (to some extent) if? if we make stronger assumptions on the functions to approximate $\rightsquigarrow$ topic of today

Recap: SG: Hierarchical decomposition 1D

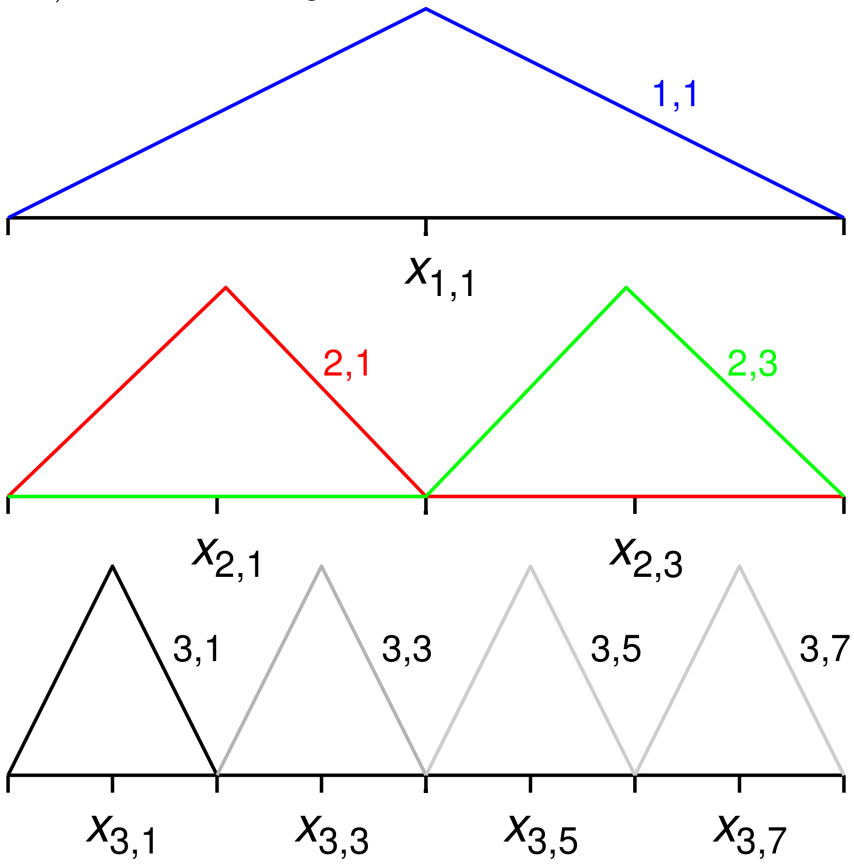
Archimedes Quadrature

$$\int_0^1 4x(1-x)\mathrm{d}x = \frac{2}{3}$$



SG: Hierarchical basis cont'd

The bases for spaces W_1, W_2 and W_3



SG: Hierarchical basis

For the hierarchical representation, we consider the hierarchical increment W_l, spanned by the basis functions $\phi_{l,i}$ such that

$$V_l = V_{l-1} \oplus W_l$$

is a direct sum (each $u_l \in V_l$ can be uniquely decomposed as $u_l = u_{l-1} + w_l$ with $u_{l-1} \in V_{l-1}$ and $w_l \in W_l \rightarrow$ remember the triangles in approach by Archimedes)

Because $\dim(V_l) = 2^l - 1$ and $\dim(V_l) = \dim(V_{l-1}) + \dim(W_l)$ we need $\dim(W_l) = 2^{l-1}$. These are given by $\phi_{l,i}$ with

$$i \in I_l = \{j \,:\, 1 \leq j < 2^l\,, \quad j \text{ odd}\}$$

Then

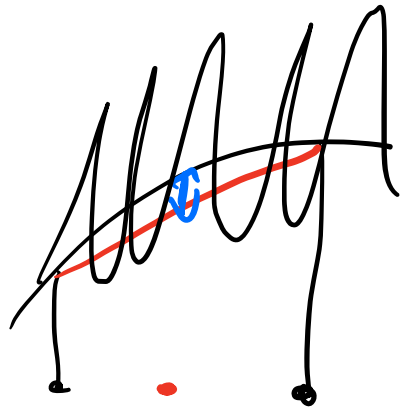
$$W_l = \text{span}\{\phi_{l,i}\,, i \in I_l\}$$

Note that $W_1 = V_1$

SG: Hierarchical basis cont'd

Obtain

$$V_n = \bigoplus_{l=1}^{n} W_l \,,$$



so that there is a unique representation for each $u \in V_n$ as

$$u = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

Coefficients $v_{l,i}$ in case of interpolation $\rightsquigarrow$ visualize on board

The coefficients $v_{l,i}$ are hierarchical differences

$$v_{l,i} = u^*(x_{l,i}) - \frac{u(x_{l,i-1}) + u(x_{l,i+1})}{2}$$

where u^* is the function to be interpolated

SG: Analysis of hierarchical decomposition

We now analyze the hierarchical decomposition

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

when interpolating a u. What would we like to obtain?

SG: Analysis of hierarchical decomposition

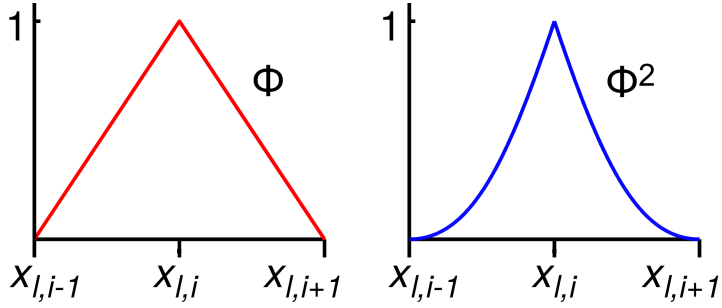
We now analyze the hierarchical decomposition

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

when interpolating a u. What would we like to obtain?

For our decomposition, we first calculate the norms of the basis functions

$$\|\phi_{l,i}\|_\infty = 1\,, \qquad \|\phi_{l,i}\|_2 = \sqrt{\frac{2h_l}{3}}$$



Show decay of

$$w_e = \sum_{i \in I_e} v_{e,i} \, \phi_{e,i}$$

Represent hierarchical coefficient as

$$\left.\begin{aligned} \underline{v_{e,i}} &= \int_0^1 \psi_{e,i}(x) \frac{\partial^2}{\partial x^2} u(x) \, dx \\ \psi_{e,i} &= -\frac{h_e}{2} \phi_{e,i} \end{aligned}\right\} \text{Lemma 3.2}$$

$\Rightarrow$ requires twice differentiable u

$\Rightarrow$ strong smoothness assumption

$$|v_{e,i}| \leq \frac{h_e}{2} \underline{\|\phi_{e,i}\|_2} \, \left\| \frac{\partial^2}{\partial x^2} u_{|T_i} \right\|_2 \qquad T_i = [x_{e,i-1}, x_{e,i+1}]$$

$$\leq \sqrt{\frac{h_e^3}{6}} \underbrace{\left\| \frac{\partial^2}{\partial x^2} u_{|T_i} \right\|_2}$$

$\Rightarrow$ requires bounded 2nd derivatives

define $\mu_2(u|T_i) = \left\| \frac{\partial^2}{\partial x^2} u_{|T_i} \right\|_2$

Now increment

$$w_e = \sum_{i \in I_e} v_{e,i} \, \phi_{e,i}$$

$$\|w_\ell\|_2^{②} = \int_0^1 w_\ell^2(x)\,dx =$$

$$= \int_0^1 \Big(\sum_{i\in I_\ell} v_{\ell,i}\,\varphi_{\ell,i}(x)\Big)^2 dx$$

$$\leq \sum_{i\in I_\ell} v_{\ell,i}^2\,\|\varphi_{\ell,i}\|_2^2$$

$$\leq \frac{h_\ell^3}{6}\,\frac{2h_\ell}{3} \sum_{i\in I_\ell} \mu_2(u(T_i))^2$$

$$= \frac{h_\ell^4}{9}\,\mu_2(u)^2$$

$\Rightarrow \|w_\ell\|_2 \in \mathcal{O}(h_\ell^2)$

$\Rightarrow$ norm of increments decay quadratically with level ℓ <u>if</u> u bounded 2nd-order derivatives

$\Rightarrow$ Represent u as series

$$u = \sum_{\ell=1}^{\infty} w_\ell$$

Recall

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

If u is twice differentiable, the $\| \cdot \|_2$ norm of the increments

$$w_l = \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

decays as $\mathcal{O}(h_l^2) \rightsquigarrow$ board

Recall

$$u_n = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

If u is twice differentiable, the $\|\cdot\|_2$ norm of the increments

$$w_l = \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

decays as $\mathcal{O}(h_l^2) \rightsquigarrow$ board

This means, we can write a twice differentiable u as a series

$$u = \sum_{l=1}^{\infty} w_l \,,$$

that converges because $\|w_l\|_2 \in \mathcal{O}(h_l^2)$. In particular, obtain

$$u - u_n = \sum_{l=n+1}^{\infty} w_l$$

$\rightsquigarrow$ decay of $\|w_l\|_2$ helps us to understand error $u - u_n$

In the 1-dimensional case, the benefit of the hierarchical basis is limited:

In the 1-dimensional case, the benefit of the hierarchical basis is limited:

- We can leave out nodes and still get a reasonable approximation in contrast to a nodal-point basis

In the 1-dimensional case, the benefit of the hierarchical basis is limited:

- We can leave out nodes and still get a reasonable approximation in contrast to a nodal-point basis
- Adding a new level requires us to compute the coefficients of the new level only but keeps the coefficients at the previous levels unchanged

What will "can remove nodes and still get reasonable results" amount to in higher dimension?

In the 1-dimensional case, the benefit of the hierarchical basis is limited:

- We can leave out nodes and still get a reasonable approximation in contrast to a nodal-point basis
- Adding a new level requires us to compute the coefficients of the new level only but keeps the coefficients at the previous levels unchanged

What will "can remove nodes and still get reasonable results" amount to in higher dimension? In the multi-dimensional case, we will now see that many of the hierarchical increments have high costs and low benefit in terms of error $\rightarrow$ we will then remove those hierarchical increments and obtain *sparse grids*

SG: Hierarchical basis in multivariate case

Let now $\boldsymbol{x} = [x_1, \ldots, x_d]$ with $d \in \mathbb{N}$ and $d > 1$.

Again consider the domain $\Omega = [0, 1]^d$ and functions $u|_{\partial\Omega} = 0$

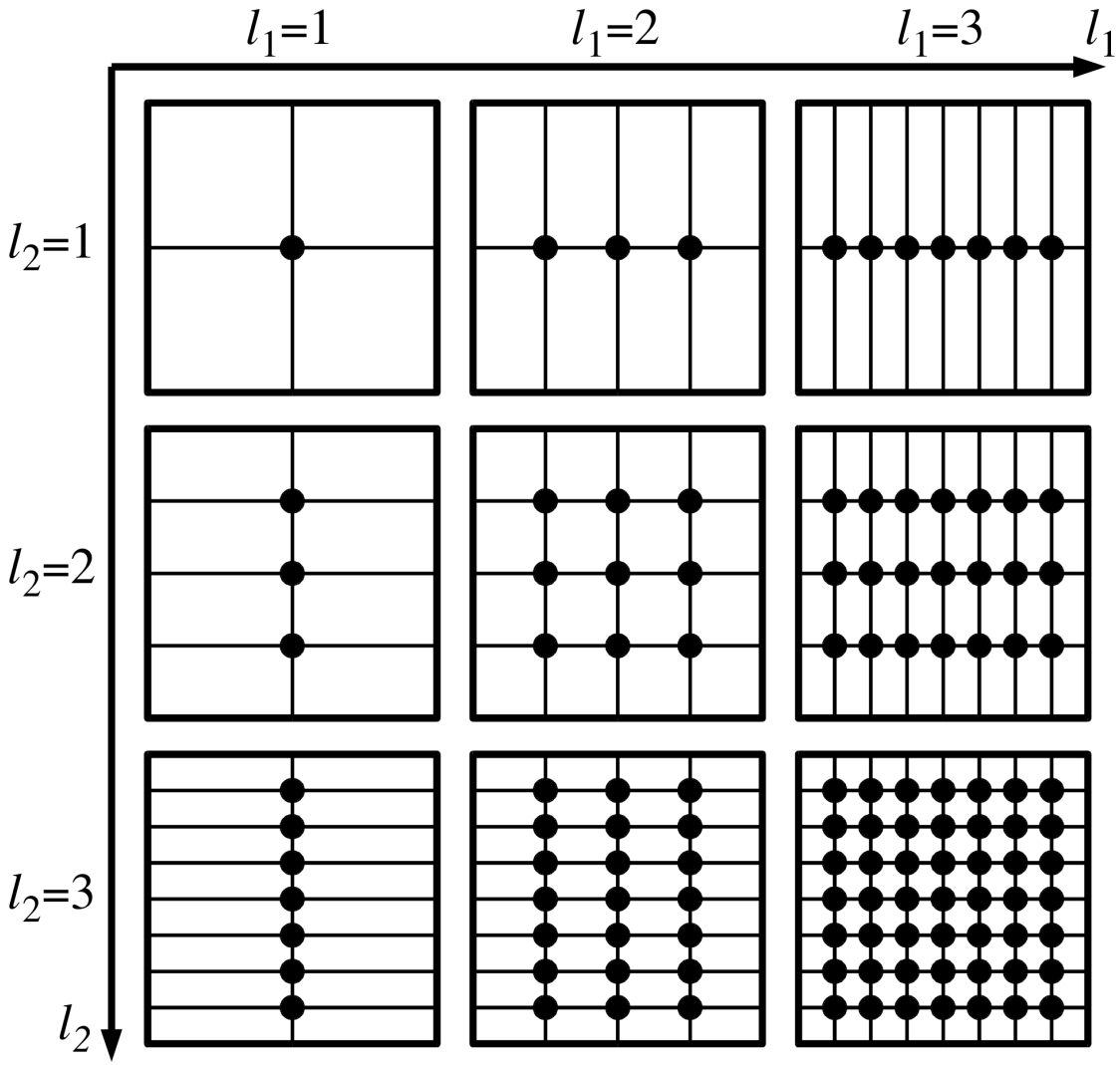
Multidimensional level $\boldsymbol{l} = [l_1, \ldots, l_d] \in \mathbb{N}^d$

Multidimensional mesh width $h_{\boldsymbol{l}} = [h_1, \ldots, h_d] = [2^{-l_1}, \ldots, 2^{-l_d}]$. Note that different mesh width in different dimensions allowed

Define $|\boldsymbol{l}|_1 = l_1 + \cdots + l_d$ and $|\boldsymbol{l}|_\infty = \max\{l_1, \ldots, l_d\}$

Grid points are $\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}} = [i_1 h_1, \ldots, i_d h_d]$

SG: Notation

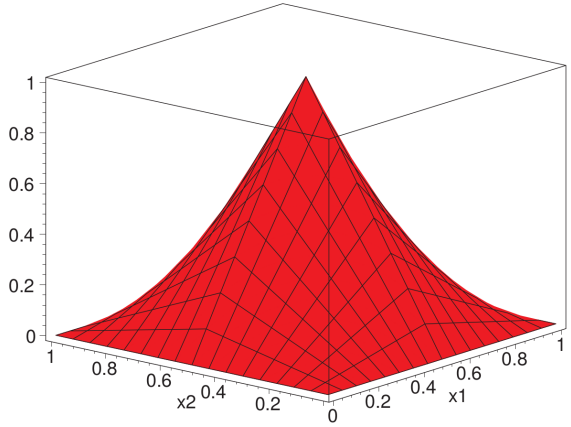


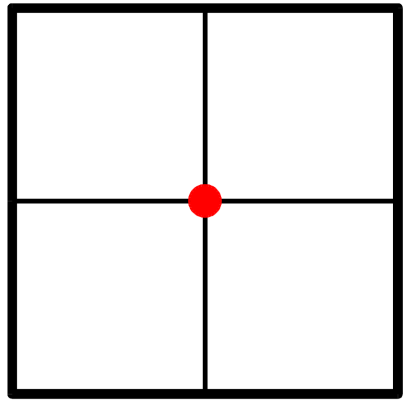
SG: Piecewise d-linear functions

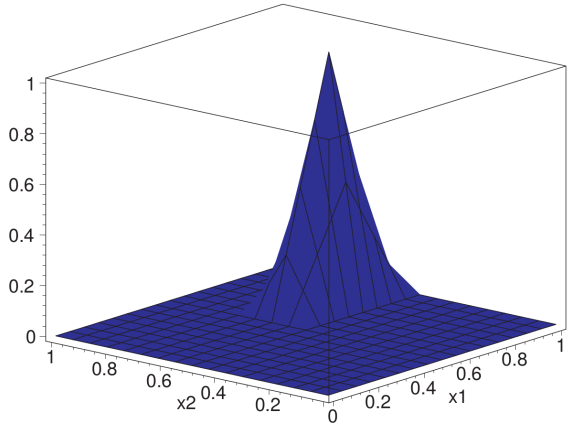
Generalize continuous, piecewise linear functions to continuous, piecewise d-linear functions with respect to $h_{\boldsymbol{l}}$:

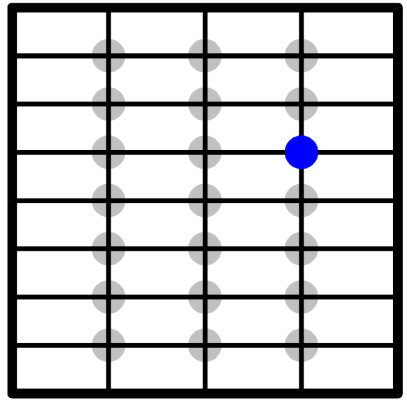
$$\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) = \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j)$$

For $d = 2$, the functions $\phi_{[1,1],[1,1]}$ and $\phi_{[2,3],[3,5]}$ are plotted on the right









SG: Spaces

Consider

$$\Phi_{\boldsymbol{l}} = \{\phi_{l,i} \,:\, 1 \leq \boldsymbol{i} < 2^{\boldsymbol{l}}\},$$

where the $\leq$ is to be read component-wise: each i_j must be at least 1 and at most $2^{l_j} - 1$

The space of piecewise d-linear functions is

$$V_{\boldsymbol{l}} = \text{span}\{\Phi_{\boldsymbol{l}}\}$$

with dimension

$$\dim(V_{\boldsymbol{l}}) = (2^{l_1} - 1) \cdots (2^{l_d} - 1) \in \mathcal{O}(2^{|\boldsymbol{l}|_1})$$

Special case $l_1 = \cdots = l_d$ set $V_n = V_{[n,\ldots,n]}$

SG: Hierarchical increments

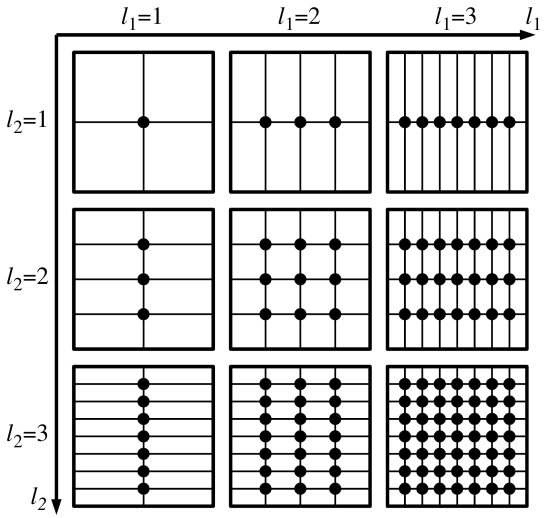
Define the hierarchical increment $W_{\boldsymbol{l}}$ as

$$W_{\boldsymbol{l}} = \text{span}\{\phi_{\boldsymbol{l},\boldsymbol{i}} \,:\, \boldsymbol{i} \in I_{\boldsymbol{l}}\}\,,$$

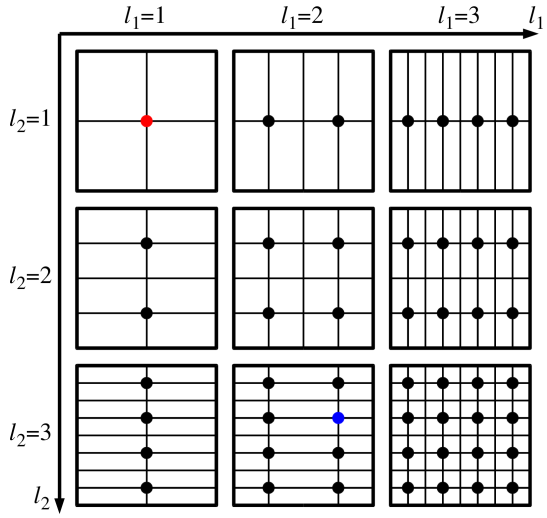
where $\boldsymbol{I}_{\boldsymbol{l}} = \{\boldsymbol{i} \,:\, 1 \leq \boldsymbol{i} < 2^{\boldsymbol{l}}\,,\ \text{all } i_j \text{ odd}\}$

Contains just those functions from $\boldsymbol{V}_{\boldsymbol{l}}$ that vanish at all points of coarser grid

full grid



hierarchical increments



SG: Hierarchical subspace decomposition

Obtain unique representation of $u_{\boldsymbol{l}} \in V_{\boldsymbol{l}}$ for $\boldsymbol{l} \in \mathbb{N}^d$ as

$$u_{\boldsymbol{l}} = \sum_{\boldsymbol{l}' \leq \boldsymbol{l}} w_{\boldsymbol{l}'} \,,$$

with $w_{\boldsymbol{l}'} \in W_{\boldsymbol{l}'}$

Now it will be worthwhile to estimate the norm of $w_{\boldsymbol{l}}$ to understand which contribute most to the accuracy of the representation

$\rightsquigarrow$ **board**

Multi-dimensional case:

Represent $u \in V_\ell$ $\quad \ell \in \mathbb{N}^d$ as

$$u = \sum_{\ell' \leq \ell} w_{\ell'} \quad , \quad w_{\ell'} \in W_{\ell'}$$

$\Rightarrow$ estimate norm of $w_{\ell'}$

Hierarchical coefficient

$$v_{\ell,i} = \int_{[0,1]^d} \varphi_{\ell,i} \, \partial^{2d} u \, dx \quad , \quad \varphi_{\ell,i} = 2^{-|\ell|_1 - d} \phi_{\ell,i}$$

$$\partial^{2d} u = \frac{\partial^{2d} u}{\partial_{x_1}^2 \cdots \partial_{x_d}^2}$$ mixed $2d$-fold derivative

$\Rightarrow$ STRONG assumption

Obtain

$$\underline{\|w_\ell\|_2} \leq 3^{-d} \underbrace{2^{-2|\ell|_1}}_{\text{decays } \underline{\text{fast}} \text{ with level } \ell} \|\partial^{2d} u\|_2$$

$\Rightarrow$ understand cost/benefit of each of these increments w_ℓ

costs: number of grid points of w_ℓ

$$c(\ell) = 2^{|\ell|_1 - d}$$

benefit: Let $L \subseteq \mathbb{N}^d$ be a set of levels that are selected

$$u_L = \sum_{\ell \in L} w_\ell$$

Then

$$u - u_L = \sum_{\ell \notin L} w_\ell$$

For each component have bound

$$\|w_\ell\|_2 \leq s(\ell)\,\mu(u)$$

$$\|u - u_L\|_2 \leq \sum_{\ell \notin L} \|w_\ell\|_2 \leq \Big(\sum_{\ell \notin L} s(\ell)\Big)\,\mu(u)$$

$$= \Big[\sum_{\ell \in \mathbb{N}^d} s(\ell) - \sum_{\ell \in L} s(\ell)\Big]\,\mu(u)$$

if select ℓ with $s(\ell)$ big, then error $\|u - u_L\|_2$ reduced

$\Rightarrow$ $s(\ell)$ is benefit of w_ℓ

With this new tool, let us analyze the cost/benefit of a "full grid",

$$L_n = \{ \ell : |\ell|_\infty \le n \}$$

Then calculations show

$$\sum_{\ell \in L_n} s(\ell) \geq \left(\tfrac{1}{3}\right)^d \left(1 - d\, 2^{-2n}\right)$$

for $n \to \infty$

$$\sum_{\ell \in \mathbb{N}^d} s(\ell) = \left(\tfrac{1}{3}\right)^d$$

$$\sum_{\ell \in \mathbb{N}^d} s(\ell) - \sum_{\ell \in L_n} s(\ell) \le \left(\tfrac{1}{3}\right)^d - \left(\tfrac{1}{3}\right)^d \left(1 - d\, 2^{-2n}\right)$$
$$\le \frac{1}{3^d}\, d\, 2^{-2n}$$

so that

$$\|u - u_L\|_2 \le \overset{\text{"}u\text{"}}{C} \sum_{\ell \notin L_n} s(\ell) \le C \frac{d}{3^d} 2^{-2n} \in \mathcal{O}(h_n^2)$$

Details in [Bungartz et al., 2004]: Hierarchical coefficient is now

$$v_{\boldsymbol{l},\boldsymbol{i}} = \int_{[0,1]^d} \psi_{\boldsymbol{l},\boldsymbol{i}} \partial^{2d} u \mathrm{d}\boldsymbol{x} \,,$$

which now depends on **mixed** 2d-fold derivative

$$\partial^{2d} u = \frac{\partial^{2d}}{\partial x_1^2 \cdots \partial x_d^2}$$

and $\psi_{\boldsymbol{l},\boldsymbol{i}} = 2^{-|\boldsymbol{l}|_1 - d} \phi_{\boldsymbol{l},\boldsymbol{i}}$

It is very important to note that the following holds *only* for functions with (L_2-)bounded $\partial^{2d} u \Rightarrow$ strong assumption on function (smoothness)

Obtain

$$\|w_{\boldsymbol{l}}\|_2 \leq 3^{-d} 2^{-2|\boldsymbol{l}|_1} \|\partial^{2d} u\|_2$$

Now we select those subspaces from the subspace scheme that minimize cost and maximize benefit for approximation function $u : [0,1]^d \to \mathbb{R}$ with sufficient smoothness

How should we measure costs?

Now we select those subspaces from the subspace scheme that minimize cost and maximize benefit for approximation function $u : [0,1]^d \to \mathbb{R}$ with sufficient smoothness

How should we measure costs? We measure cost via the number of grid points

$$c(\boldsymbol{l}) = 2^{|\boldsymbol{l}|_1 - d}$$

How should we measure benefit?

Now we select those subspaces from the subspace scheme that minimize cost and maximize benefit for approximation function $u : [0,1]^d \to \mathbb{R}$ with sufficient smoothness

How should we measure costs? We measure cost via the number of grid points

$$c(\boldsymbol{l}) = 2^{|\boldsymbol{l}|_1 - d}$$

How should we measure benefit? Measure benefit of subspace selection via introduced error if left out. Let $L \subset \mathbb{N}^d$ of levels that are selected, then obtain

$$u_L = \sum_{\boldsymbol{l} \in L} w_{\boldsymbol{l}}$$

and

$$u - u_L = \sum_{\boldsymbol{l} \notin L} w_{\boldsymbol{l}}$$

For each component $w_{\boldsymbol{l}}$ have derived bounds of the form

$$\|w_{\boldsymbol{l}}\| \leq s(\boldsymbol{l})\mu(u)$$

where $\mu(u)$ was typically $\|\partial^{2d} u\|_2$. Then obtain

$$\|u - u_L\| \leq \sum_{\boldsymbol{l} \notin L} \|w_{\boldsymbol{l}}\|_2 \leq \left(\sum_{\boldsymbol{l} \notin L} s(\boldsymbol{l}) \right) \mu(u)$$

$$= \left[\left(\sum_{\boldsymbol{l} \in \mathbb{N}^d} s(\boldsymbol{l}) \right) - \left(\sum_{\boldsymbol{l} \in L} s(\boldsymbol{l}) \right) \right] \mu(u)$$

Thus, if we select $\boldsymbol{l}$ with $s(\boldsymbol{l})$ big, then the error $\|u - u_L\|$ is reduced $\rightarrow$ use $s(\boldsymbol{l})$ as the benefit of subspace $W_{\boldsymbol{l}}$

SG: Quality of full-grid space

With this new tool in hand, let us analyze the cost/benefit of a "full grid", i.e., a grid corresponding to the selection

$$L_n = \{\boldsymbol{l} \,:\, |\boldsymbol{l}|_\infty \leq n\}$$

In the L_2 norm, we have bounds of the order

$$s(\boldsymbol{l}) = 2^{-2|\boldsymbol{l}|_1}$$

Then, calculations show that

$$\sum_{\boldsymbol{l}\in L_n} s(\boldsymbol{l}) \geq \left(\frac{1}{3}\right)^d (1 - d2^{-2n})$$

and for $n \to \infty$

$$\sum_{\boldsymbol{l}\in \mathbb{N}^d} s(\boldsymbol{l}) = \left(\frac{1}{3}\right)^d$$

Thus

$$\sum_{\boldsymbol{l}\in \mathbb{N}^d} s(\boldsymbol{l}) - \sum_{\boldsymbol{l}\in L_n} s(\boldsymbol{l}) \leq \left(\frac{1}{3}\right)^d - \left(\frac{1}{3}\right)^d (1 - d2^{-2n}) \leq \frac{d}{3^d} 2^{-2n}$$

SG: Approximation quality of full-grid space

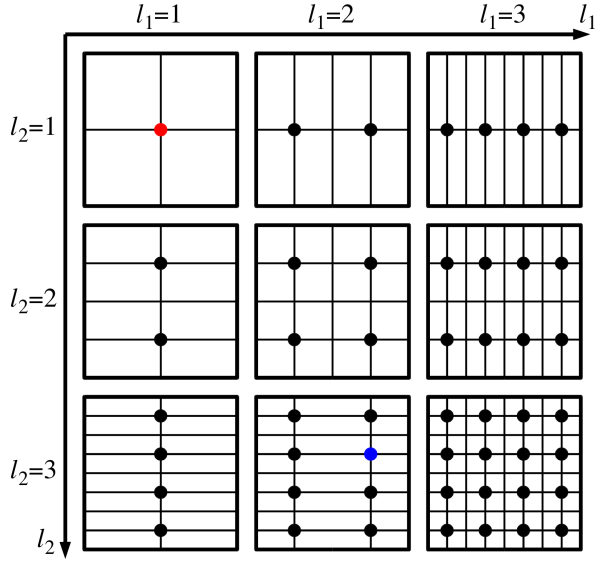
Benefit: $2^{-2|\ell|_1}$

Obtain

$$\|u - u_{L_n}\|_2 \leq C \sum_{\boldsymbol{l} \notin L_n} s(\boldsymbol{l}) \leq \frac{Cd}{3^d} 2^{-2n} \in \mathcal{O}(h_n^2)$$

This is what we expect from a piecewise linear approximation

What additional insights have we obtained?



SG: Approximation quality of full-grid space

Obtain

$$\|u - u_{L_n}\|_2 \leq C \sum_{\boldsymbol{l} \notin L_n} s(\boldsymbol{l}) \leq \frac{Cd}{3^d} 2^{-2n} \in \mathcal{O}(h_n^2)$$

This is what we expect from a piecewise linear approximation

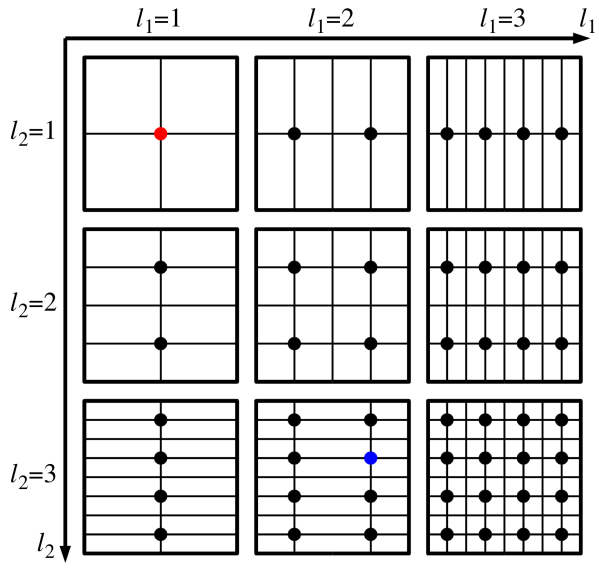
What additional insights have we obtained?

- The sum of local benefits

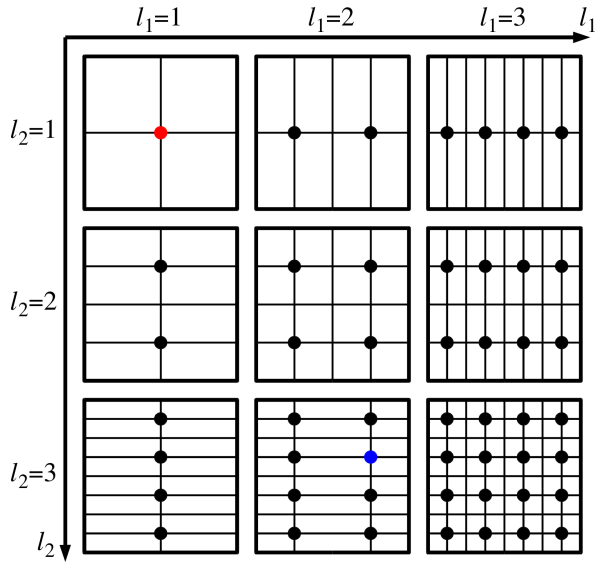
$$\sum_{\boldsymbol{l} \in L_n} 2^{-2|\boldsymbol{l}|_1},$$

  means that subspace on diagonals (i.e., with constant $|\boldsymbol{l}|_1$) have the same benefit.

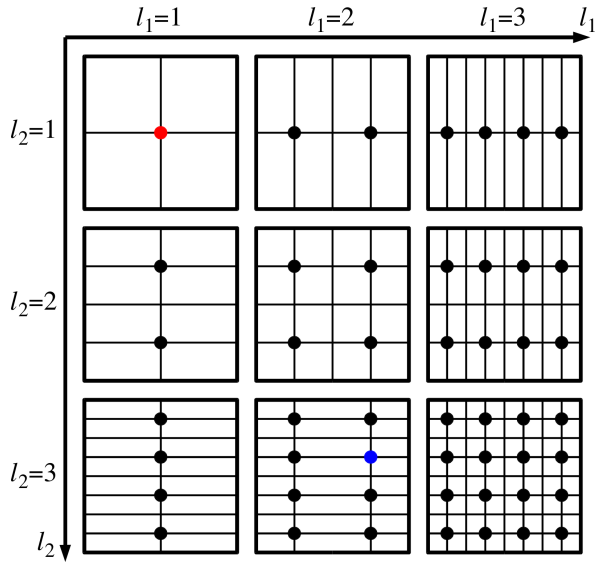
- Also, as we move further to the bottom right, the benefit gets less and less



If we now look at the costs $c(\boldsymbol{l}) = 2^{|\boldsymbol{l}|_1 - d}$ (which is the number of grid points), then we see that the costs are constant on diagonals as well

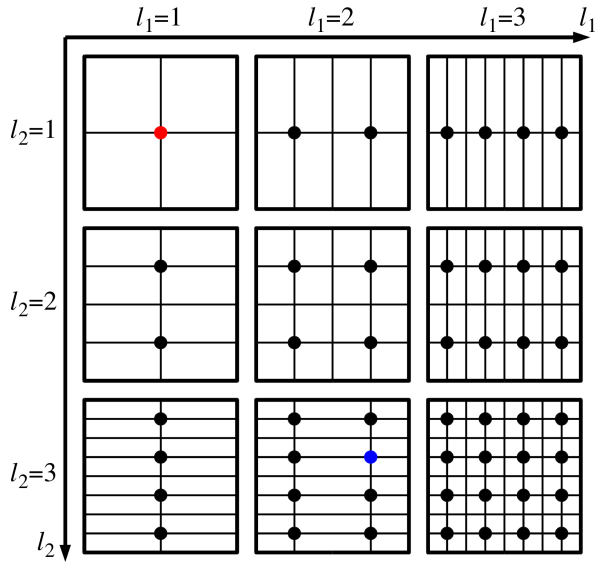


If we now look at the costs $c(\boldsymbol{l}) = 2^{|\boldsymbol{l}|_1 - d}$ (which is the number of grid points), then we see that the costs are constant on diagonals as well



Thus, the cost/benefit ratio $c(\boldsymbol{l})/s(\boldsymbol{l})$ is *constant* on diagonals. In particular, for lower-triangular diagonals, we add subspaces with worse and worse cost/benefit ratios $\Rightarrow$ what should we do with them?

If we now look at the costs $c(\boldsymbol{l}) = 2^{|\boldsymbol{l}|_1 - d}$ (which is the number of grid points), then we see that the costs are constant on diagonals as well

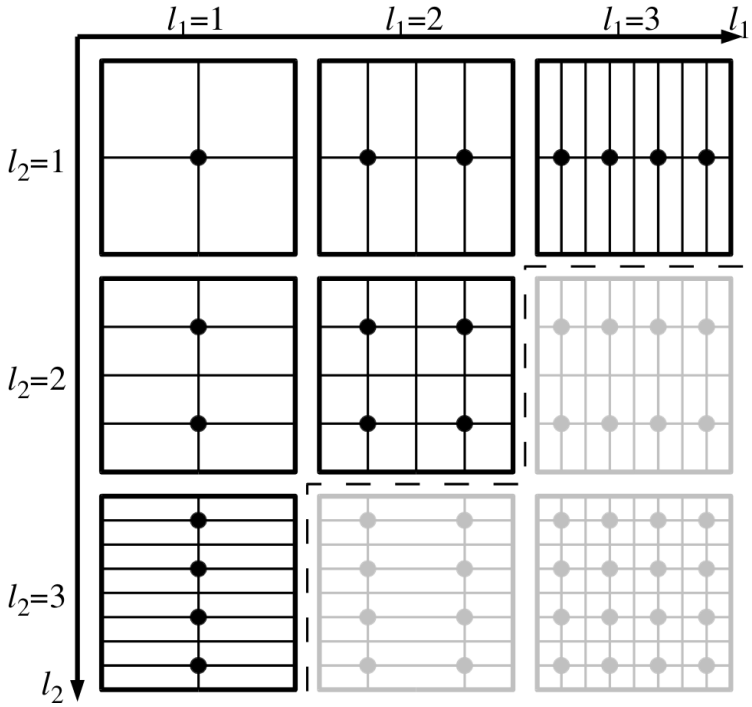


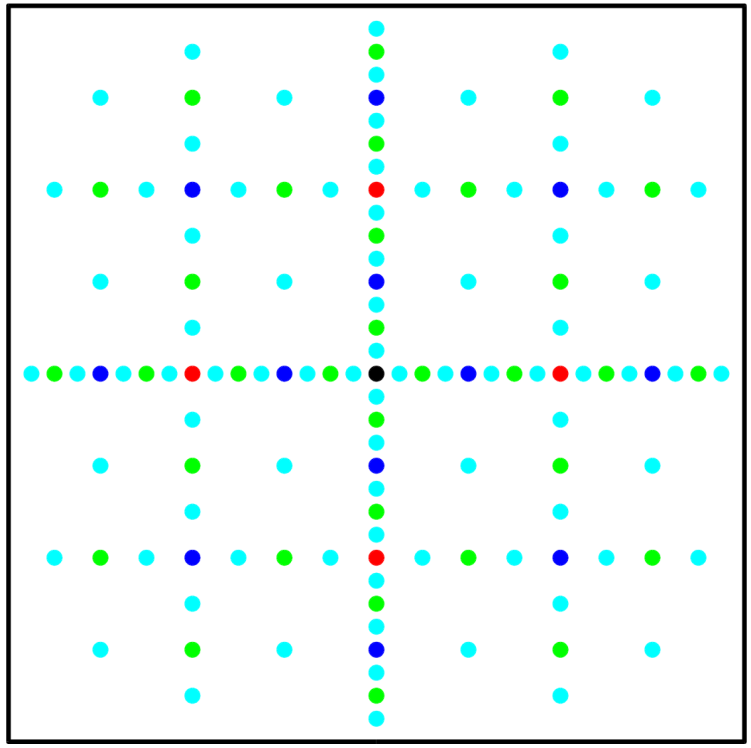
Thus, the cost/benefit ratio $c(\boldsymbol{l})/s(\boldsymbol{l})$ is *constant* on diagonals. In particular, for lower-triangular diagonals, we add subspaces with worse and worse cost/benefit ratios
$\Rightarrow$ what should we do with them? let's truncate them

Consider the diagonal cut $L_n^{(1)} = \{\boldsymbol{l} : |\boldsymbol{l}|_1 \leq n + d - 1\}$ and the sparse grid space

$$\boldsymbol{V}_n^{(1)} = \bigoplus_{|\boldsymbol{l}|_1 \leq n+d-1} W_{\boldsymbol{l}}$$

Here is an example of a sparse grid





SG: Properties of sparse grids

The number of grid points of a sparse grid grows as $\mathcal{O}(2^n n^{d-1})$ in contrast to $\mathcal{O}(2^{nd})$ of a full grid

If u has (L_2-)bounded mixed derivatives up to order $2d$, then

$$\|u - u_n^{(1)}\|_2 \in \mathcal{O}(2^{-2n} n^{d-1}),$$

whereas a full-grid space achieves

$$\|u - u_n\|_2 \in \mathcal{O}(2^{-2n})$$

Sparse-grid spaces achieve slightly worse error than full-grid space but drastically reduced points in higher dimensions d

Comparing the number of grid points corresponding to full-grid and sparse-grid spaces:

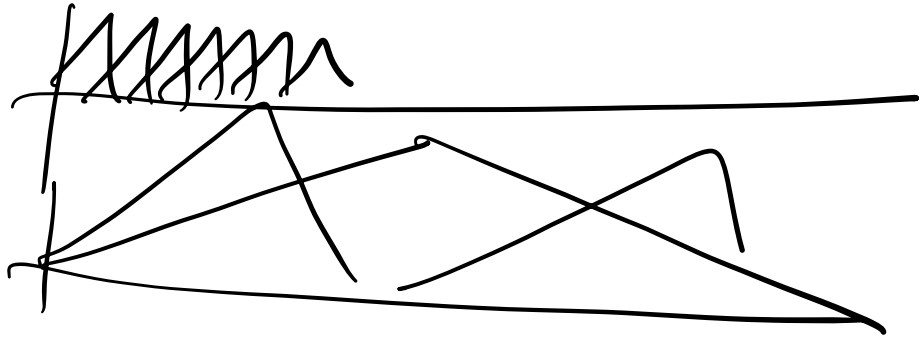
Dimension $d = 2$:

n	1	2	3	4	5	...	10
$\dim V_n = (2^n - 1)^2$	1	9	49	225	961	...	1 046 529
$\dim V_n^1 = 2^n(n-1) + 1$	1	5	17	49	129	...	9 217

Dimension $d = 3$:

n	1	2	3	4	...	10
$\dim V_n = (2^n - 1)^3$	1	27	343	3 375	...	1 070 590 167
$\dim V_n^1 = 2^n \left(\frac{n^2}{2} - \frac{n}{2} + 1\right) - 1$	1	7	31	111	...	47 103

Why does this work?



Why does this work?

Exploit the additional smoothness given by the assumption on the mixed derivatives of function u

The hierarchical basis is a key ingredient:

Why does this work?

Exploit the additional smoothness given by the assumption on the mixed derivatives of function u

The hierarchical basis is a key ingredient:

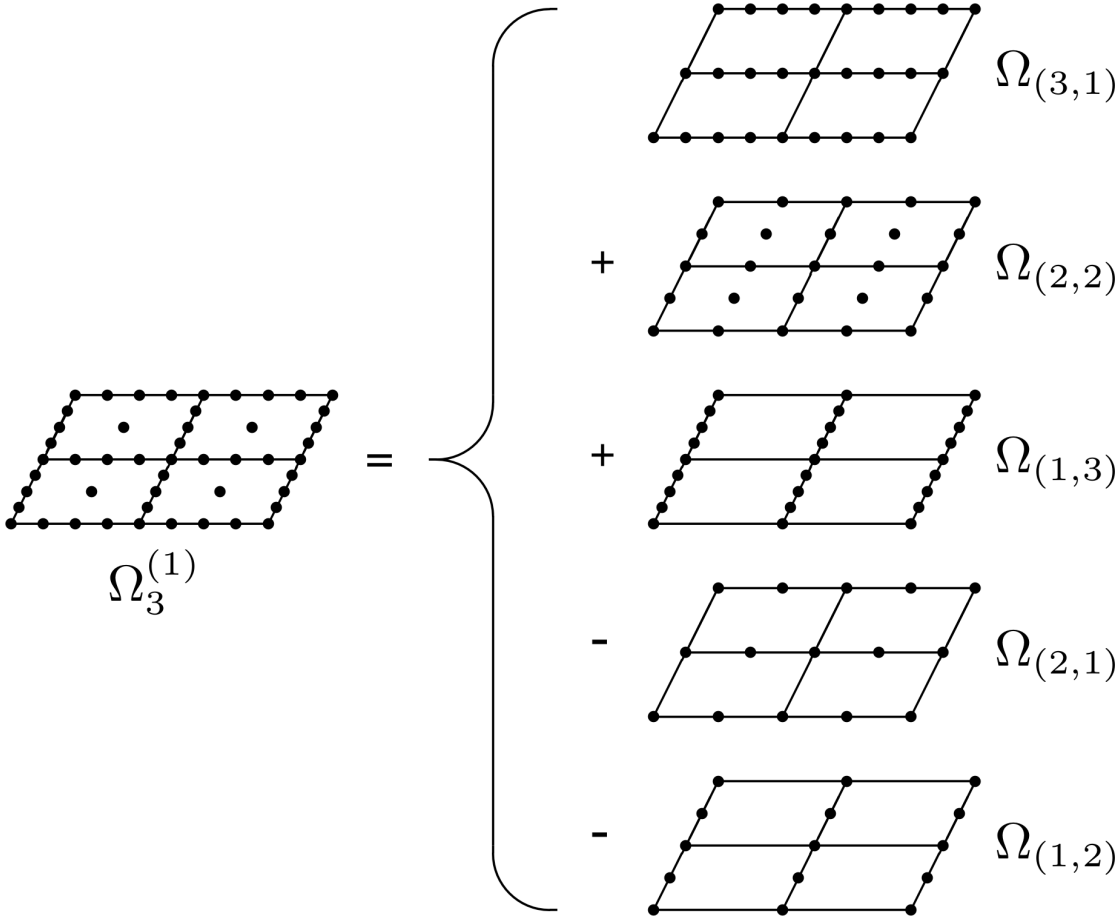
- Exploits smoothness by having "semi-global" support (i.e., function is smoother, so we can reach far over the domain and know it won't change too much),
- Introduced a hierarchy/multilevel and the coefficients in this hierarchy/multilevel basis decay fast ($\rightarrow$ multigrid, multilevel Monte Carlo)

(Logarithmic dependence can be avoided if measure error in energy norm)

Details: Bungartz, Griebel, Acta Numerica, 2004

SG: Combination technique

Formally, sparse grids are superpositions of coarser full grids



Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

- Equidistantly spaced nodes

$$h_i = h = \frac{b-a}{n}\,, \qquad t_i = a + ih\,, \qquad i = 0, \ldots, n$$

then quadrature formulas are called the *Newton-Cotes formulas* with weights

$$\lambda_{in} = \frac{1}{b-a}\int_a^b \prod_{i\neq j} \frac{t-t_i}{t_i-t_j}\,dt = \frac{1}{n}\int_0^n \prod_{i\neq j} \frac{s-j}{i-j}\,ds$$

These weights are independent of the interval boundaries a and b and can be pre-computed once and for all:

Table 9.1. Newton-Cotes weights λ_{in} for $n = 1, \ldots, 4$.

n	$\lambda_{0n}, \ldots, \lambda_{nn}$	Error	Name
1	$\frac{1}{2}$ $\frac{1}{2}$	$\frac{h^3}{12} f''(\tau)$	Trapezoidal rule
2	$\frac{1}{6}$ $\frac{4}{6}$ $\frac{1}{6}$	$\frac{h^5}{90} f^{(4)}(\tau)$	Simpson's rule, Kepler's barrel rule
3	$\frac{1}{8}$ $\frac{3}{8}$ $\frac{3}{8}$ $\frac{1}{8}$	$\frac{3h^5}{80} f^{(4)}(\tau)$	Newton's 3/8-rule
4	$\frac{7}{90}$ $\frac{32}{90}$ $\frac{12}{90}$ $\frac{32}{90}$ $\frac{7}{90}$	$\frac{8h^7}{945} f^{(6)}(\tau)$	Milne's rule

Generic rule

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(t_i)$$

Trapezoidal

$$\hat{I}_T(f) = \underline{(b-a)} \left(\underline{\tfrac{1}{2}} f(a) + \underline{\tfrac{1}{2}} f(b) \right)$$

Simpson

$$\hat{I}_S(f) = \underline{(b-a)} \left(\tfrac{1}{6} f(a) + \tfrac{4}{6} f\left(\tfrac{a+b}{2}\right) + \tfrac{1}{6} f(b) \right)$$

Lagrange polynomials: $x_0 = a$, $x_1 = b$

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

$$\ell_0(x) = \frac{(x-b)}{(a-b)} \qquad \int_a^b \ell_0(x)\,dx = \frac{b-a}{2}$$

$$\ell_1(x) = \frac{(x-a)}{(b-a)} \qquad \int_a^b \ell_1(x)\,dx = \frac{b-a}{2}$$

For $n+1$ pairwise distinct nodes $t_0, \dots, t_n$ there exists one and only one quadrature rule

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(t_i)$$

which is exact for $P \in \mathbb{P}_n$.

Have $\hat{I}$ that is exact for $P \in \mathbb{P}_n$

$\Rightarrow$ insert Lagrange polynomials for $t_0, \dots, t_n$

$$L_{in} \in \mathbb{P}_n$$

$$I(L_{in}) = \hat{I}(L_{in}) = (b-a) \sum_{j=0}^{n} \lambda_j L_{in}(t_j)$$

$$= (b-a) \lambda_i$$

$$\lambda_i = \frac{1}{(b-a)} I(L_{in})$$

Newton-Cotes formulas continued

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt$$

Gauss-Christoffel quadrature (cont'd)

Can we say something about the nodes $\tau_{0n}, \ldots, \tau_{nn}$?

Gauss-Christoffel quadrature (cont'd)

Can we say something about the nodes $\tau_{0n}, \ldots, \tau_{nn}$?

Theorem: For $n \in \mathbb{N}$, consider nodes $\tau_{0n}, \ldots, \tau_{nn}$. For an n, define $\hat{I}$ as

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_{in} f(\tau_{in})$$

and let it be exact for polynomials $p \in \mathbb{P}_{2n+1}$

$$\hat{I}_n(p) = \int_a^b \omega(t) p(t) \mathrm{d}t \,.$$

Then, the polynomials $\{P_n\}$ given by

$$P_{n+1}(t) = (t - \tau_{0n}) \cdot \ldots \cdot (t - \tau_{nn}) \in \mathbb{P}_{n+1}$$

are orthogonal with respect to the scalar product induced by $\omega(t)$

Gauss-Christoffel quadrature (cont'd)

Therefore, the nodes τ_{in} have to be roots of pairwise orthogonal polynomials $\{P_j\}$ of degree $\deg(P_j) = j$

For a given ω, we already know that the set of orthogonal polynomials $\{P_j\}$ is unique (if leading coefficient is 1)

We also know that the roots of these polynomials are real and have to lie in (a, b)
For each ω and $n \in \mathbb{N}$, this gives us a unique set of nodes, namely the roots of the corresponding orthogonal polynomial P_{n+1}

We thus have fixed $n + 1$ degrees of freedom so far...

What do we know about the weights for fixed $t_0, \ldots, t_n$ when we want to integrate exactly polynomials up to degree n?

What do we know about the weights for fixed $t_0, \ldots, t_n$ when we want to integrate exactly polynomials up to degree n?

Because we want to be able to exactly integrate polynomials up to degree $2n+1$, we already know that to even achieve exactness up to degree n, the weights are fixed for given nodes $t_0, \ldots, t_n$:

$$\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t) \mathrm{d}t \,, \qquad \text{Lagrange poly } L_{in}(\tau_{jn}) = \delta_{ij}$$

("For $n+1$ pairwise distinct nodes, the exists only one quadrature formula that exactly integrate polynomials up to degree n.")

Theorem: Let $\tau_{0n}, \ldots, \tau_{nn}$ be the roots of the $(n+1)$st orthogonal polynomial for the weight ω. Then any quadrature formula $\hat{I}$ is exact for polynomials up to order n if and only if it is exact up to order $2n+1$.

Proof

Let $\tau_{0n}, \dots, \tau_{nn}$ be the roots of the $(n+1)$-st orthog. polynomial with weight w. then any rule

$$\hat{I}_n(f) = \sum_{i=0}^{n} \lambda_i f(\lambda_i)$$

satisfies

$$\hat{I}_n \text{ exact on } \mathbb{P}_n \iff \hat{I}_n \text{ exact on } \mathbb{P}_{2n+1}$$

— "$\Rightarrow$":

suppose $P \in \mathbb{P}_{2n+1}$, then there exists $Q, R \in \mathbb{P}_n$

$$P = Q P_{n+1} + R$$

P_{n+1} orthogonal to $\mathbb{P}_n$, so that

$$\int_a^b w P = \underbrace{\int w Q P_{n+1}}_{=0} + \int w R = \int w R = \hat{I}(R) =$$

$$= \hat{I}(R) = \sum_{i=0}^{n} \lambda_{in} R(\tau_{in})$$

$$= \sum_{i=0}^{n} \lambda_{in} \left(Q(\tau_{in}) \overbrace{P_{n+1}(\tau_{in})}^{=0} + R(\tau_{in}) \right)$$

$$= \hat{I}(P)$$

Lagrange vs Newton

$$\ell_i(x) = \prod_{\substack{j=0 \\ i \neq j}}^{n} \frac{(x - x_j)}{(x_i - x_j)} \quad , \quad w_i(x) = \prod_{j=0}^{i-1} (x - x_j)$$

Recap: Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

Today

Last time

- Function approximation in higher dimensions
- Quadrature in higher dimensions

Today

- Monte Carlo

Announcements

- Homework 7 is posted and due Mon, Dec 9 before class (1 week)
- Next week, Mon Dec 9, recap of important topics—highly recommended!
- Email me by Wed, Dec 4 if you need special accommodations for the final exam

There are many opportunities for asking questions

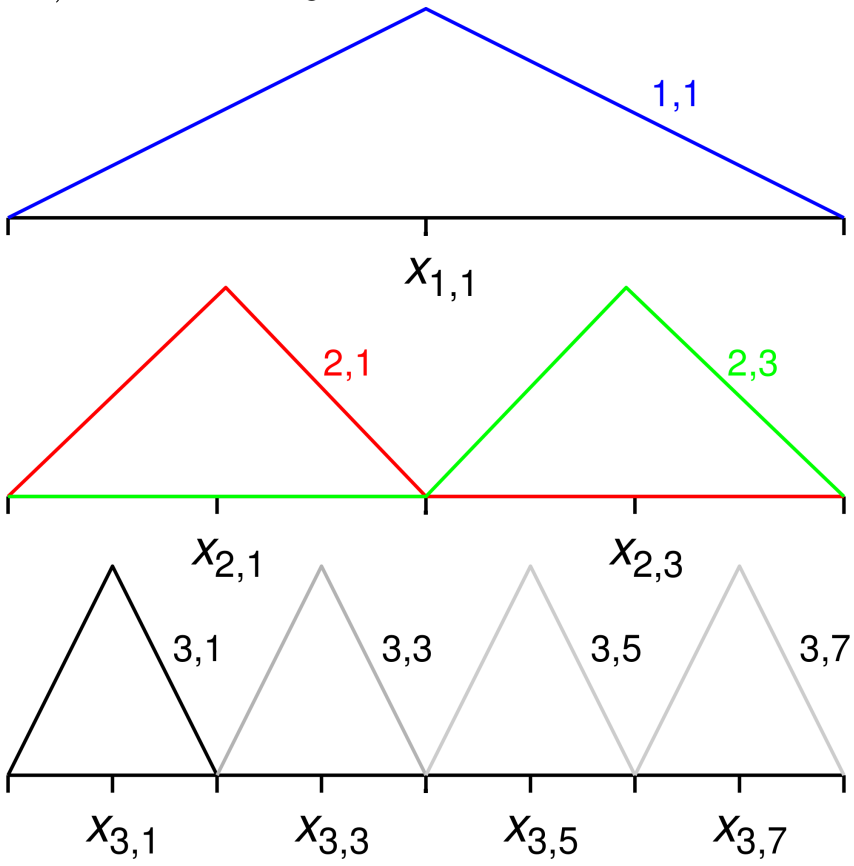
Mon, Dec 2	office hour
Wed, Dec 4	lecture (virtual) [not part of final exam]
Fri, Dec 6	office hour (grader)
Mon, Dec 9	recap and Q&A
Mon, Dec 9	office hour
Wed, Dec 11	extra office hour, 6.10pm ET (WWH 421) (all HWs graded; no HW re-grading after Thu, Dec 12)
Mon, Dec 16	final exam

- Double check that all homeworks are entered correctly in Brightspace
- Will go through example final exam problems next week
- Expect to write *some* code in the final exam
- Be in the room 10min early on the day of the final exam (most likely room changed to what is in Albert now)

High-dimensional interpolation and quadrature

SG: Hierarchical basis cont'd

The bases for spaces W_1, W_2 and W_3



SG: Hierarchical basis cont'd

Obtain

$$V_n = \bigoplus_{l=1}^{n} W_l \,,$$

so that there is a unique representation for each $u \in V_n$ as

$$u = \sum_{l=1}^{n} w_l = \sum_{l=1}^{n} \sum_{i \in I_l} v_{l,i} \phi_{l,i}$$

Coefficients $v_{l,i}$ in case of interpolation $\rightsquigarrow$ visualize on board

The coefficients $v_{l,i}$ are hierarchical differences

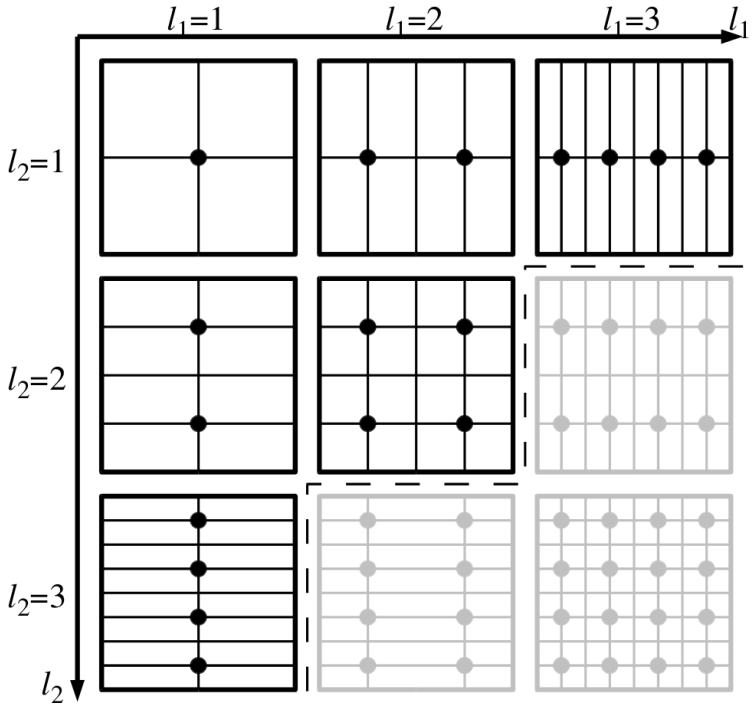
$$v_{l,i} = u^*(x_{l,i}) - \frac{u(x_{l,i-1}) + u(x_{l,i+1})}{2}$$

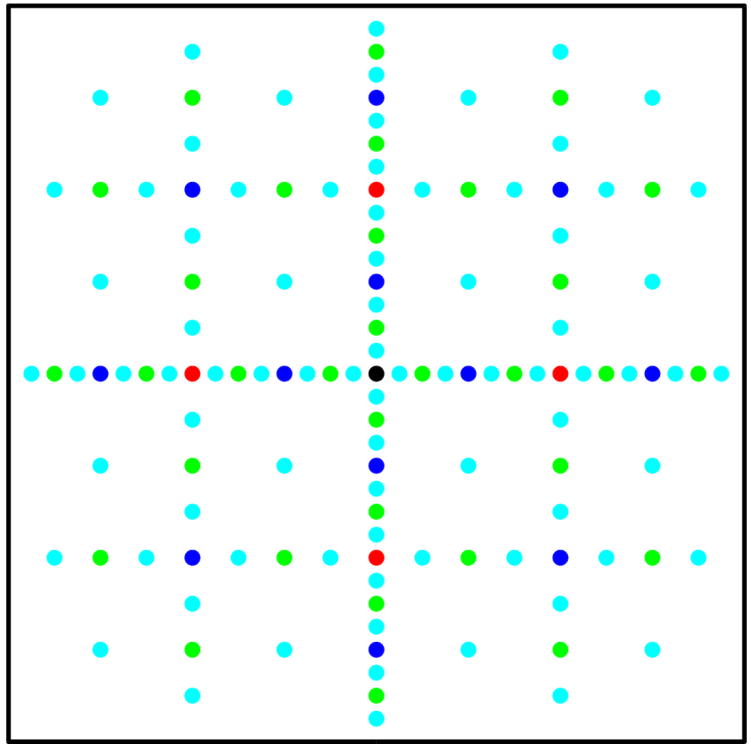
where u^* is the function to be interpolated

Consider the diagonal cut $L_n^{(1)} = \{\boldsymbol{l} : |\boldsymbol{l}|_1 \leq n + d - 1\}$ and the sparse grid space

$$\boldsymbol{V}_n^{(1)} = \bigoplus_{|\boldsymbol{l}|_1 \leq n+d-1} W_{\boldsymbol{l}}$$

Here is an example of a sparse grid





SG: Properties of sparse grids

The number of grid points of a sparse grid grows as $\mathcal{O}(2^n n^{d-1})$ in contrast to $\mathcal{O}(2^{nd})$ of a full grid

If u has (L_2-)bounded mixed derivatives up to order $2d$, then

$$\|u - u_n^{(1)}\|_2 \in \mathcal{O}(2^{-2n} n^{d-1}),$$

whereas a full-grid space achieves

$$\|u - u_n\|_2 \in \mathcal{O}(2^{-2n})$$

Sparse-grid spaces achieve slightly worse error than full-grid space but drastically reduced points in higher dimensions d

Comparing the number of grid points corresponding to full-grid and sparse-grid spaces:

Dimension $d = 2$:

n	1	2	3	4	5	...	10
$\dim V_n = (2^n - 1)^2$	1	9	49	225	961	...	1 046 529
$\dim V_n^1 = 2^n(n-1) + 1$	1	5	17	49	129	...	9 217

Dimension $d = 3$:

n	1	2	3	4	...	10
$\dim V_n = (2^n - 1)^3$	1	27	343	3 375	...	1 070 590 167
$\dim V_n^1 = 2^n \left(\frac{n^2}{2} - \frac{n}{2} + 1\right) - 1$	1	7	31	111	...	47 103

Why does this work?

Why does this work?

Exploit the additional smoothness given by the assumption on the mixed derivatives of function u

The hierarchical basis is a key ingredient:

Why does this work?

Exploit the additional smoothness given by the assumption on the mixed derivatives of function u

The hierarchical basis is a key ingredient:

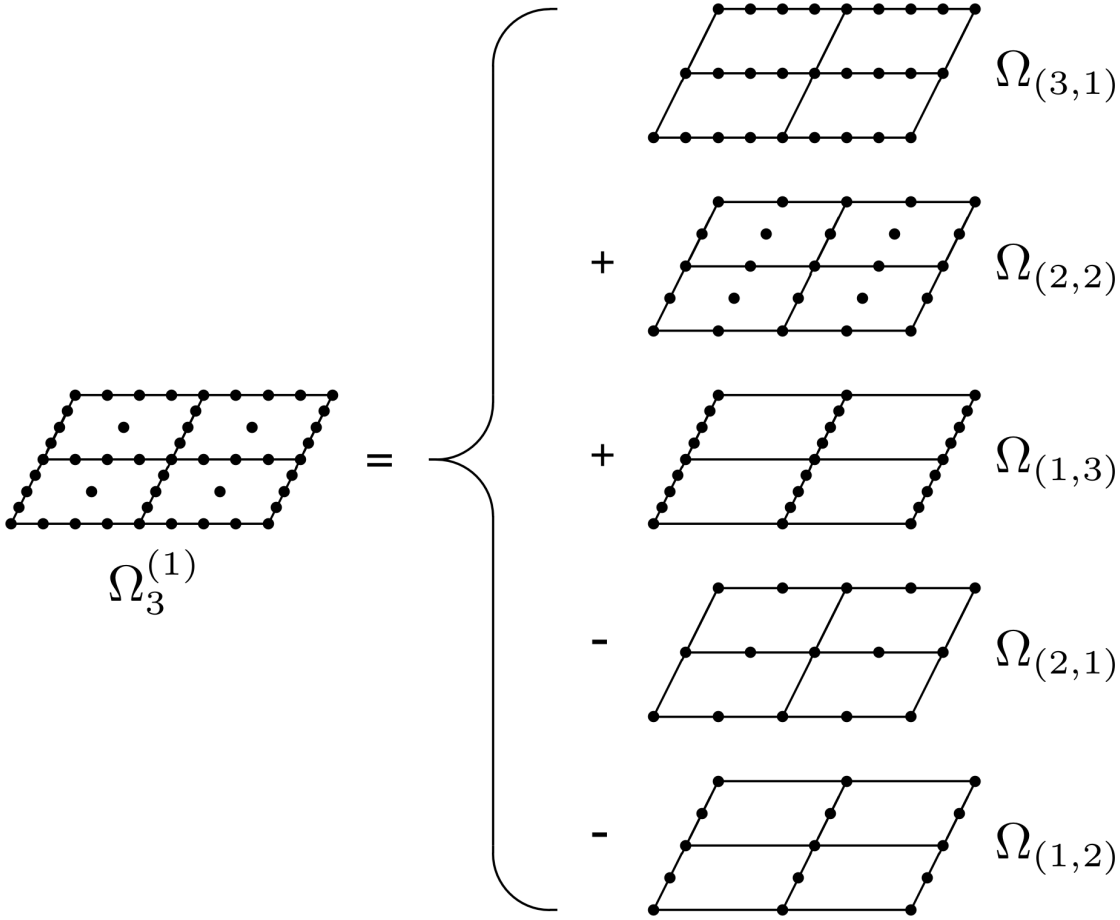
- Exploits smoothness by having "semi-global" support (i.e., function is smoother, so we can reach far over the domain and know it won't change too much),
- Introduced a hierarchy/multilevel and the coefficients in this hierarchy/multilevel basis decay fast ($\rightarrow$ multigrid, multilevel Monte Carlo)

(Logarithmic dependence can be avoided if measure error in energy norm)

Details: Bungartz, Griebel, Acta Numerica, 2004

SG: Combination technique

Formally, sparse grids are superpositions of coarser full grids



SG: Combination technique (cont'd)

This works for grids and also for functions (in certain situations)

- interpolation
- quadrature
- solutions of partial differential equations $\rightarrow$ limited

$\Rightarrow$ we are interested in quadrature

For quadrature, Smolyak has developed a related approach already in 1963

SG: Smolyak quadrature - 1D

Set $D = [-1, 1]$ and consider a one-dimensional function $f : D \to \mathbb{R}$ and we are interested in

$$If = \int_D f(x)\mathrm{d}x$$

One-dimensional quadrature rule

$$Q_l^1 f = \sum_{i=1}^{n_l} w_i f(x_i)$$

with weights $w_1, \ldots, w_{n_l}$ and points $x_1, \ldots, x_{n_l}$ and

$$X_l = \{x_i \,:\, 1 \leq i \leq n_l\}$$

Quadrature rules are nested if $X_l \subset X_{l+1}$

SG: Smolyak quadrature

Define the difference formula

$$\Delta_k f = (Q_k^1 - Q_{k-1}^1) f \,,$$

with $Q_0^1 f = 0$. What does the difference formula remind you of?

SG: Smolyak quadrature

Define the difference formula

$$\Delta_k f = (Q_k^1 - Q_{k-1}^1) f \,,$$

with $Q_0^1 f = 0$. What does the difference formula remind you of? hierarchical coefficients

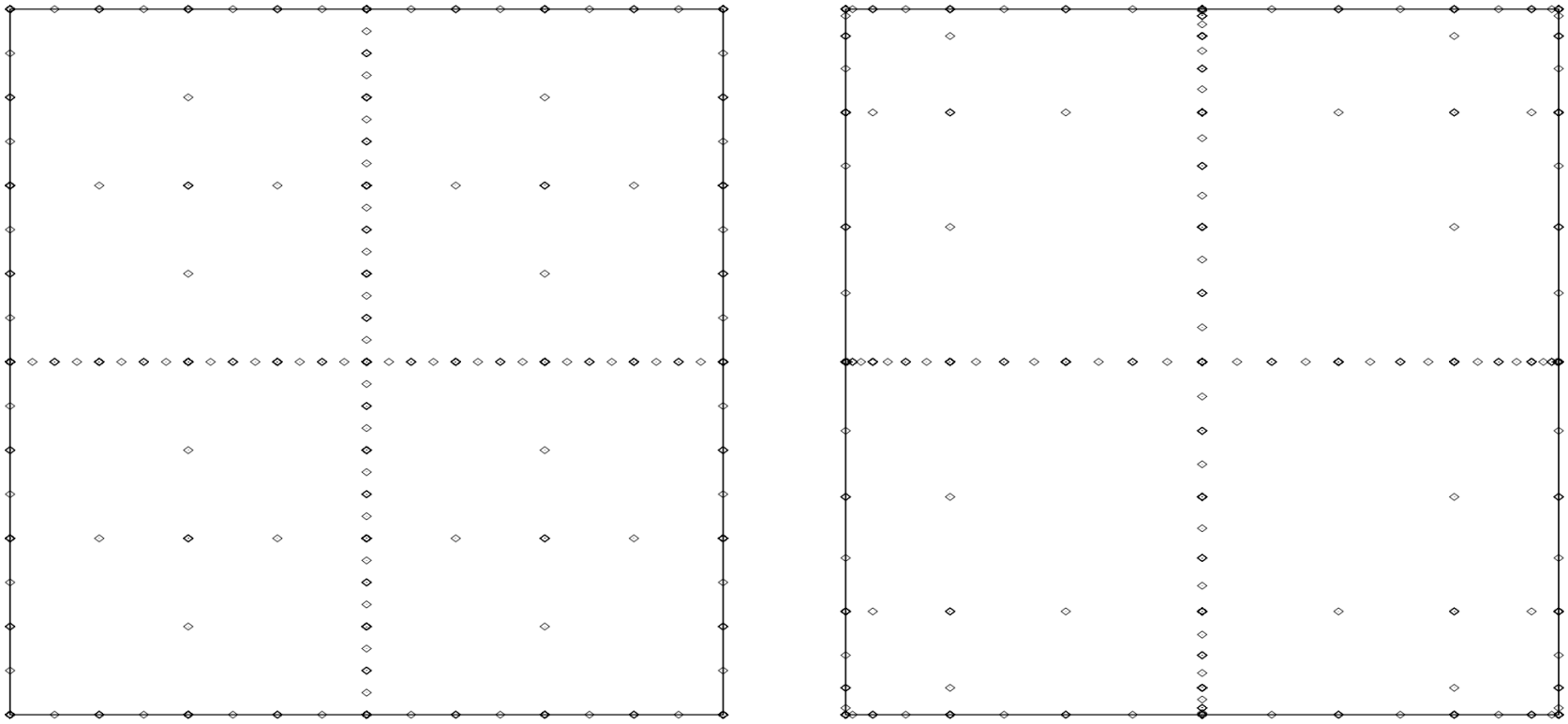
Smolyak's quadrature rule is

$$Q_l^d f = \sum_{|\boldsymbol{k}|_1 \leq n+d-1} (\Delta_{k_1}^1 \otimes \cdots \otimes \Delta_{k_d}^1) f \,,$$

where the tensor product of quadrature rules is

$$(\Delta_{k_1}^1 \otimes \cdots \otimes \Delta_{k_d}^1) f = \sum_{i_1=1}^{n_{k_1}} \cdots \sum_{i_d=1}^{n_{k_d}} w_{k_1,i_1} \cdots w_{k_d,i_d} f(x_{k_1,i_1}, \ldots, x_{k_d,i_d})$$

SG: Smolyak grid w.r.t. Clenshaw-Curtis rule



Left: sparse grid w.r.t. trapezoidal rule (piecewise constant), right: sparse grid obtained with Clenshaw-Curtis rule

SG: Alternative representations of Smolyak

A non-hierarchical representation is

$$Q_l^d f = \sum_{n \le |\boldsymbol{l}|_1 \le n+d-1} (-1)^{n+d-1-|\boldsymbol{l}|_1} \binom{d-1}{|\boldsymbol{l}|_1 - n} (Q_{l_1}^1 \otimes \cdots \otimes Q_{l_d}^1) f$$

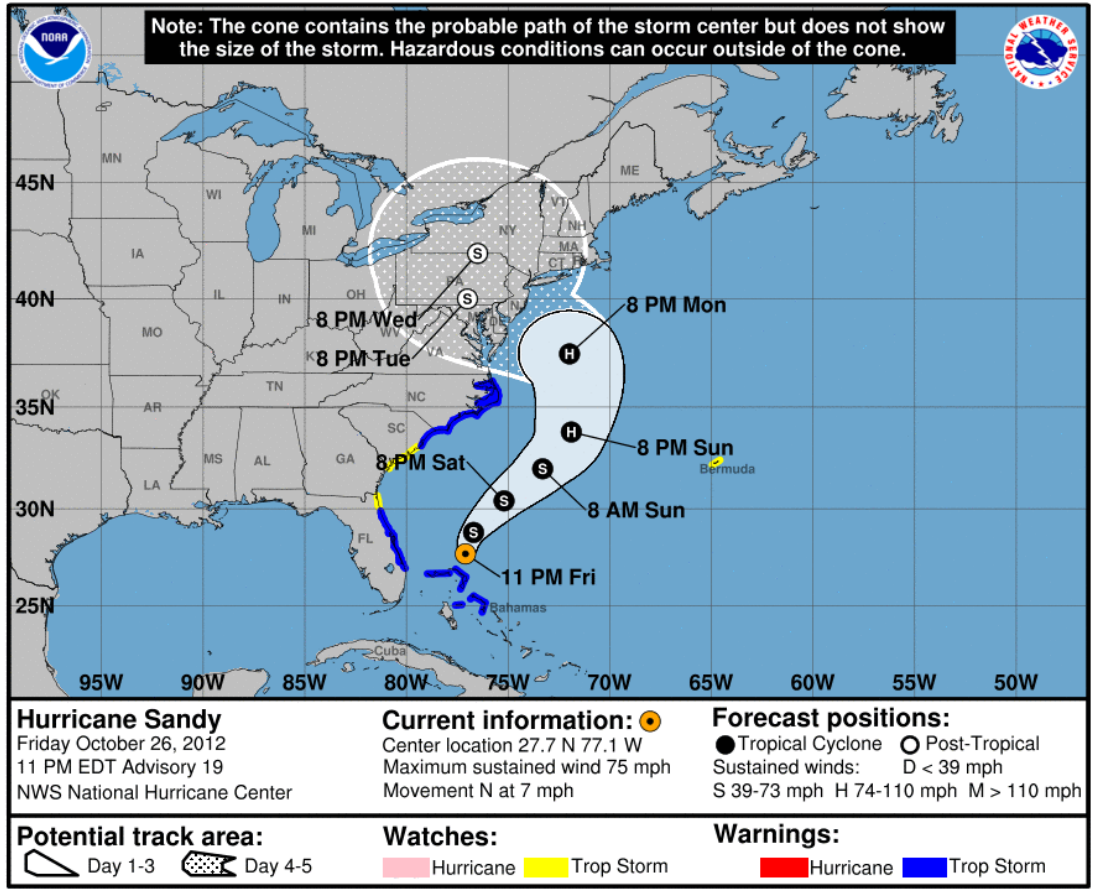
- Non-hierarchical: Can work on regular grids as with combination technique
- Simple to implement
- (Equivalence to hierarchical representation not obvious.)

Conclusions

- Computations in higher dimensions are typically affected by the curse of dimensionality, which means that computational costs become exponentially more expensive as the dimension is increased.
- We have two options in high dimension. The first option is using randomized methods and Monte Carlo which can circumvent the curse
- The other option is exploiting additional structure in the problem that can help to circumvent the curse to some extent.
- In case of sparse grids, we can circumvent the curse by assuming additional smoothness of the function to be interpolated.
- Sparse grids are very useful for quadrature in moderately high dimensions (10 up to few 100s of dimensions). Quadrature based on sparse grids is sometimes called Smolyak quadrature.

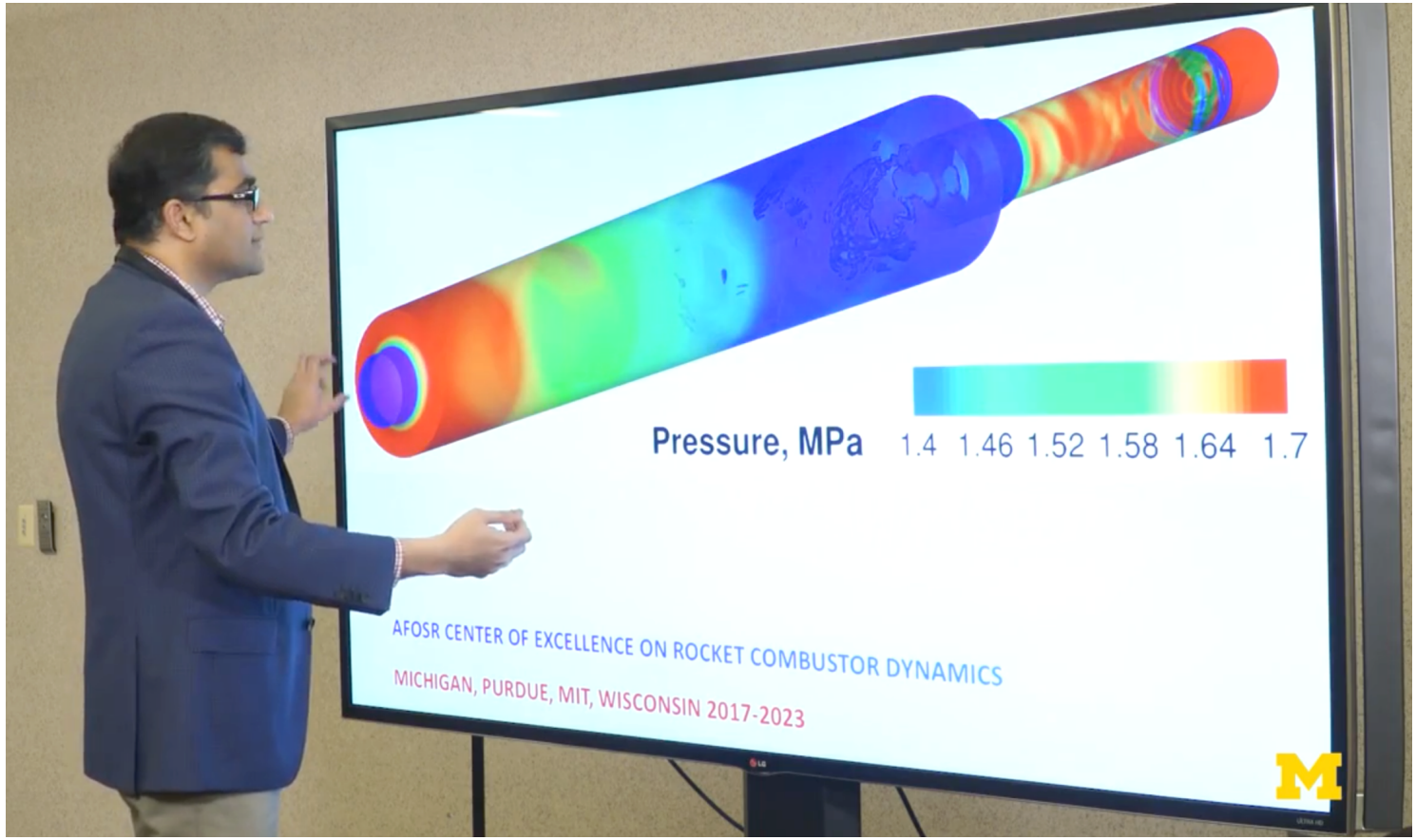
Monte Carlo methods and numerical methods

Known unknowns

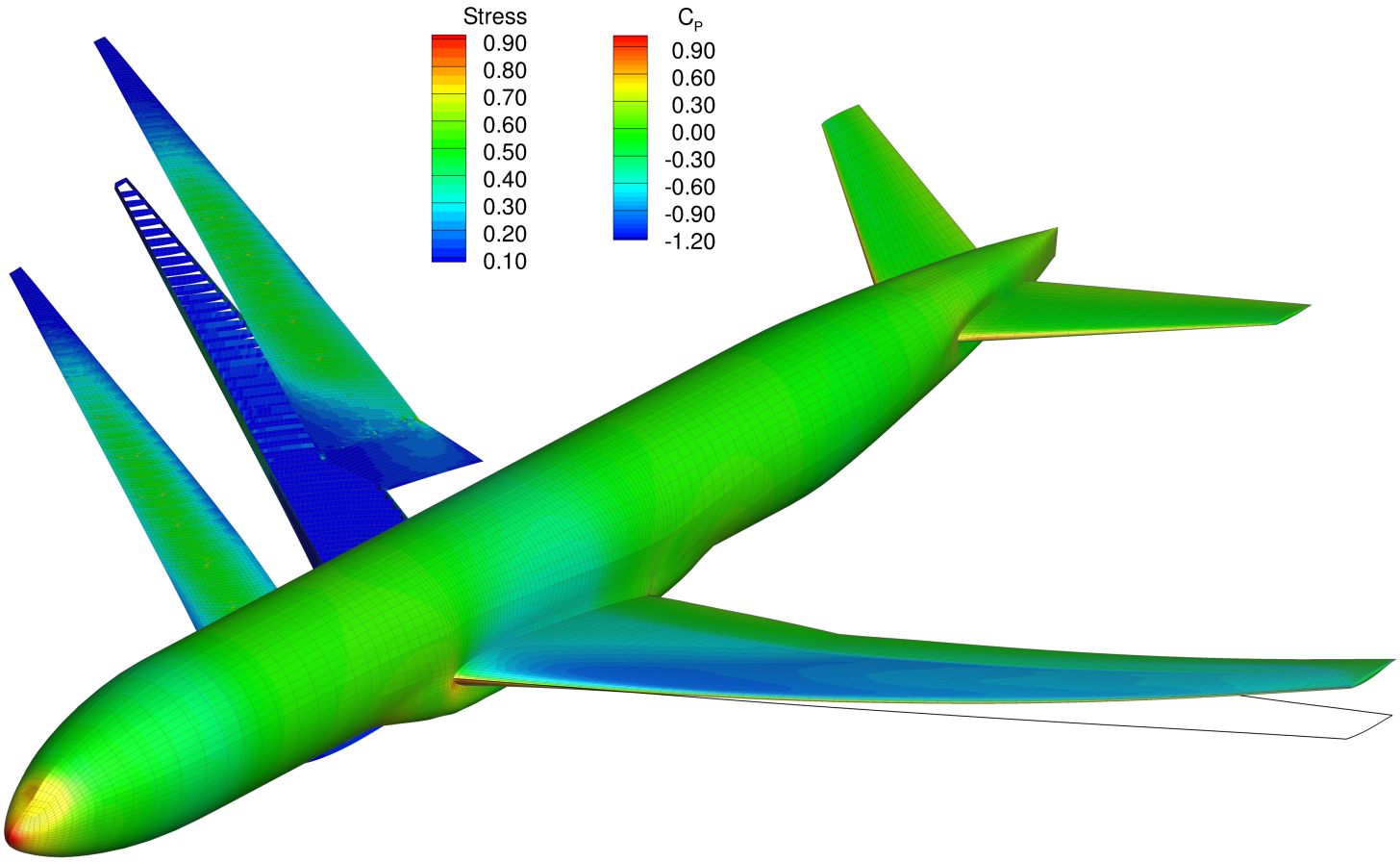


NOAA

No hope to model physics exhaustively

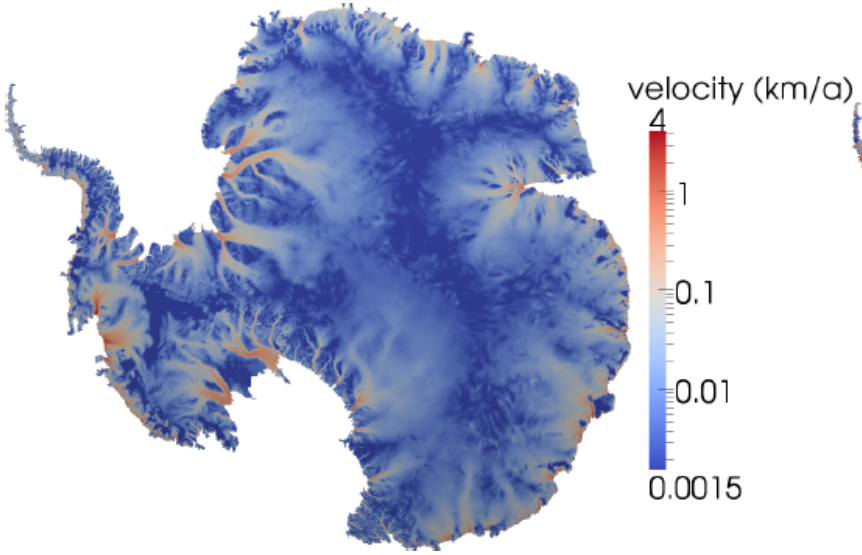


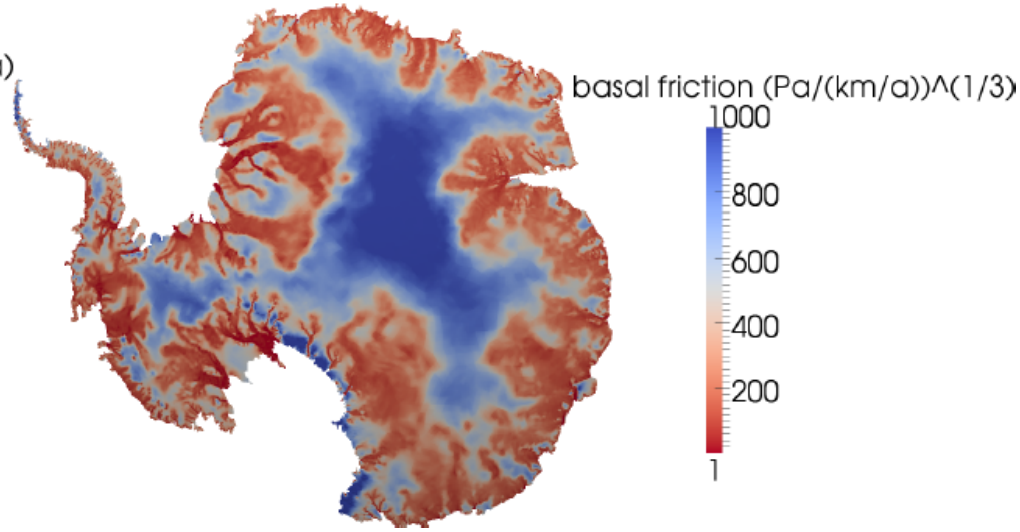
Rapidly changing dynamics



Kenway, G. K., Martins, J. R., & Kennedy, G. J. (2014). Aerostructural optimization of the Common Research Model configuration. Group (ADODG), 6(7), 8-9.

Uncertainties due to data



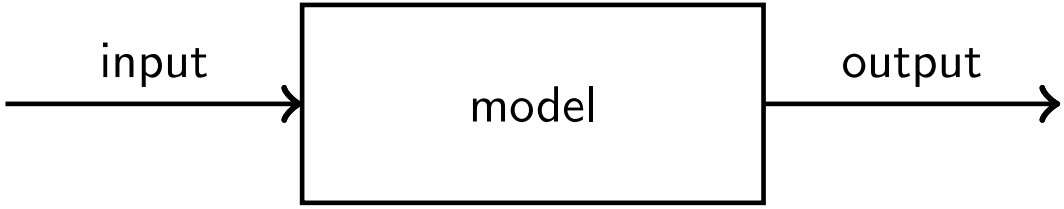


Figures: Petra, Ghattas, Isaac, Martin, Stadler, et al.

Intro: Model

Model of system of interest

- Model describes response of system to inputs, parameters, configurations
- Response typically is a quantity of interest
- Evaluating a model means numerically simulating the model
- Many models given in form of partial differential equations



Mathematical formulation

$$f : \mathcal{D} \to \mathcal{Y}$$

- Input domain $\mathcal{D}$ and output domain $\mathcal{Y}$
- Maps $\boldsymbol{z} \in \mathcal{D}$ input onto $\boldsymbol{y} \in \mathcal{Y}$ output (quantity of interest)

Intro: Model - Navier-Stokes equations

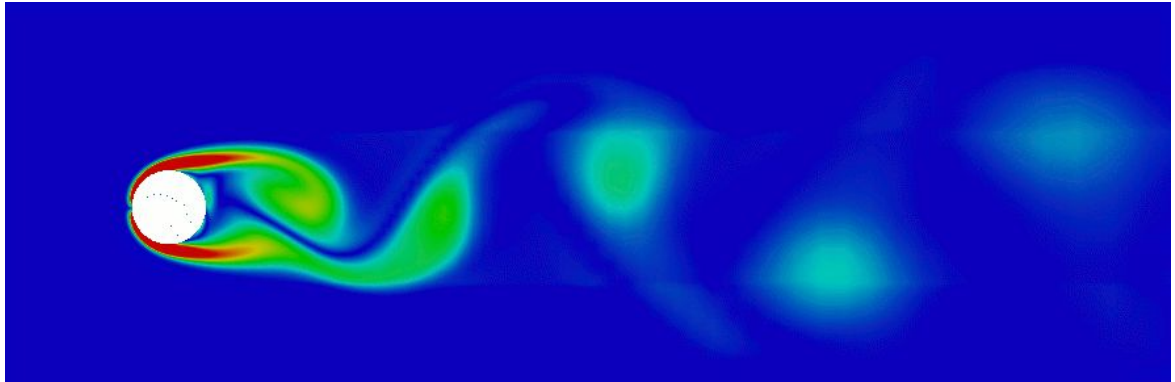
$$\rho\left(\frac{\partial u}{\partial t} + u \cdot \nabla u\right) = -\nabla p + \mu \Delta u + g$$

Examples of inputs

- Density ρ
- Dynamic viscosity μ

Examples of outputs (quantities of interest)

- Velocity at monitoring point
- Average pressure



Intro: Model - Diffusion-convection-reaction flow

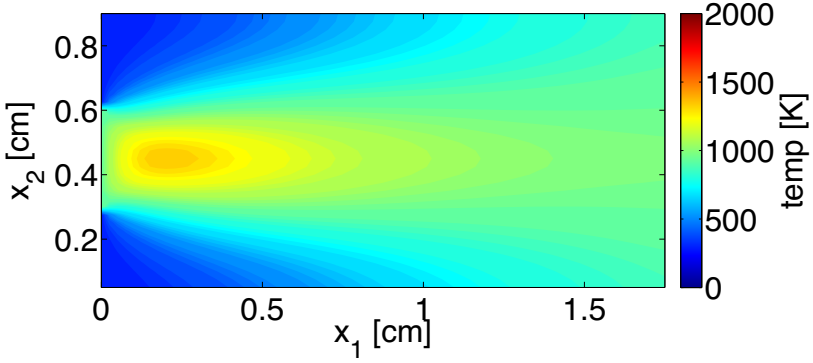
$$\frac{\partial u}{\partial t} = \Delta u - v \nabla u + g(u, \boldsymbol{\mu})$$

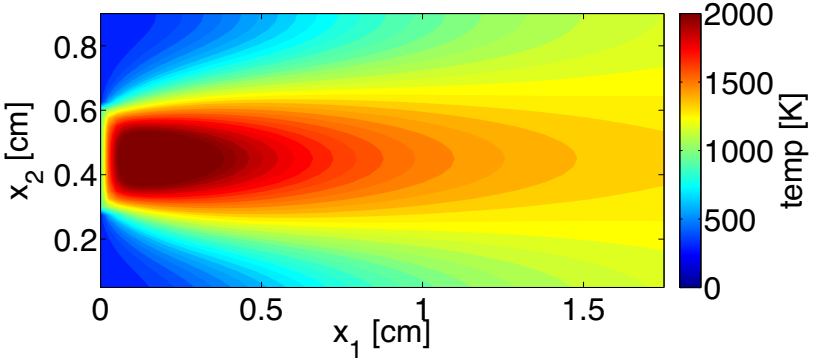
Examples of inputs

- Activation energy and pre-exponential factor (Arrhenius-type reaction)
- Temperature at boundary
- Ratio of fuel and oxidizer

Examples of outputs

- Average temperature in chamber





Intro: Uncertain inputs

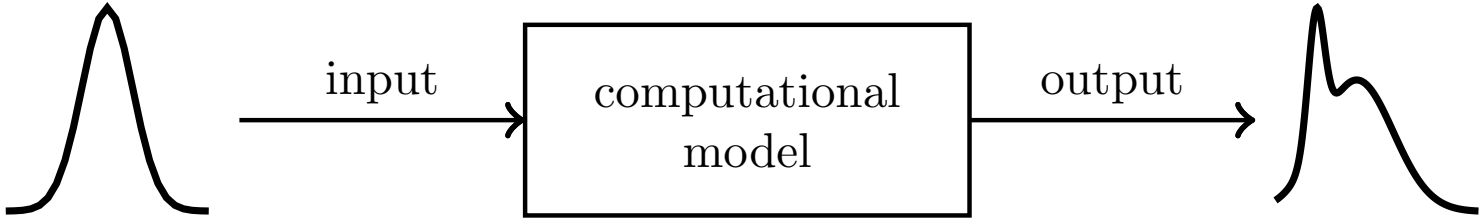
Inputs are uncertain

- Measurement errors in boundary conditions
- Manufacturing variations
- Model parameters determined by engineering judgment, etc.

Mathematically formulate uncertain inputs as random variables

$$Z : \Omega \to \mathcal{D}$$

Quantify effect of uncertainties in inputs on model outputs



Intro: General sampling-based approach

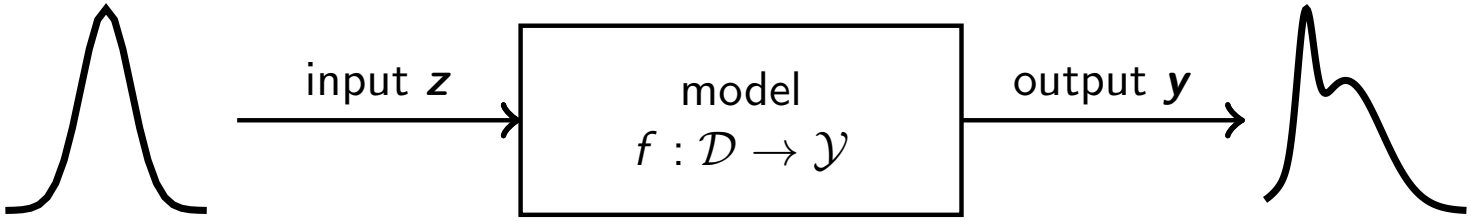
- Take many realizations of input random variable Z

$$\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n \in \mathcal{D}$$

- Evaluate model f at all $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ realizations

$$\boldsymbol{y}_1 = f(\boldsymbol{z}_1), \ldots, \boldsymbol{y}_n = f(\boldsymbol{z}_n)$$

- Estimate statistics from outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$



Intro: General sampling-based approach

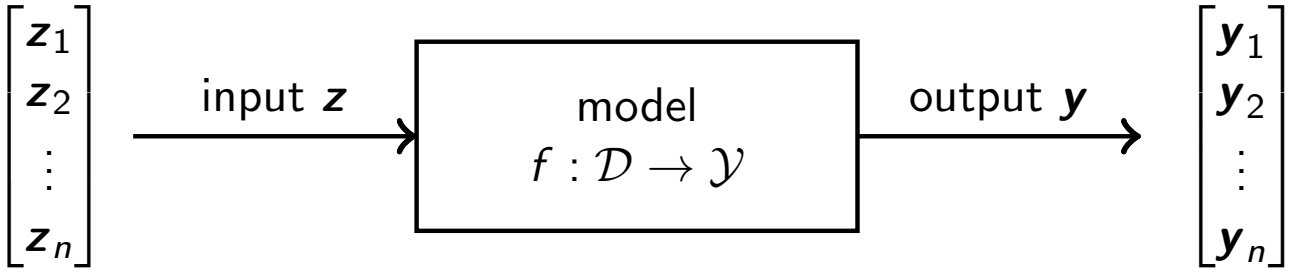
- Take many realizations of input random variable Z

$$\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n \in \mathcal{D}$$

- Evaluate model f at all $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ realizations

$$\boldsymbol{y}_1 = f(\boldsymbol{z}_1), \ldots, \boldsymbol{y}_n = f(\boldsymbol{z}_n)$$

- Estimate statistics from outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$



Monte Carlo methods

- Given $X_1, \ldots, X_n$ iid random variables that are distributed as X, the basic Monte Carlo estimator of $\mathbb{E}[X]$ is

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

- Note that each of the n $X_1, \ldots, X_n$ is a random variable, and thus $\bar{X}_n$ is a random variable too

- Thus, it makes sense to consider $\mathbb{E}[\bar{X}_n]$ and $\mathrm{Var}[\bar{X}_n]$

- Once the samples have been drawn/realized, the estimate $\bar{X}_n$ is a real number (here; clash of terminology)

Monte Carlo estimators

- Unbiasedness of Monte Carlo estimator $\mathbb{E}[\bar{X}_n] = \mathbb{E}[X]$
- Variance is $\mathrm{Var}[\bar{X}_n] = \sigma^2/n$
- With unbiasedness follows

$$\mathbb{E}\left[\left(\bar{X}_n - \mathbb{E}[X]\right)^2\right] = \mathrm{Var}\left[\bar{X}_n\right] = \frac{\sigma^2}{n}$$

The mean-squared error rate is $1/n$.

VR: Control variates

The constant in the MC rate is the variance $\mathrm{Var}[X]$ of the random variable from which MC samples are being drawn. By designing an equivalent MC approximation with lower variance, we can reduce the MSE

Auxiliary random variable

Let X be a random variable and Y be a another random variable that is correlated to X, i.e.,

$$|\rho(X, Y)| > 0\,.$$

The (Pearson) correlation coefficient $\rho(X, Y)$ is defined as

$$\rho(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}[X]\,\mathrm{Var}[Y]}}$$

with the covariance

$$\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

Control variates exploit the correlation of X and Y to reduce the MSE

Let X be a random variable and Y be an auxiliary random variable with known $\mathbb{E}[Y]$ and correlation coefficient $\rho = \rho(X, Y)$. Consider the control variate estimator

$$\theta = \bar{X}_m + \alpha(\mathbb{E}[Y] - \bar{Y}_m)\,.$$

For

$$\alpha^* = \rho\sqrt{\frac{\mathrm{Var}[X]}{\mathrm{Var}[Y]}}\,,$$

the MSE of the unbiased estimator θ of $\mathbb{E}[X]$ is

$$e(\theta) = (1 - \rho^2)\frac{\mathrm{Var}[X]}{m}\,.$$

Proof $\rightsquigarrow$ board

What is missing?

Which important point for us (numerical analysis!) does this analysis completely miss?

What is missing?

Which important point for us (numerical analysis!) does this analysis completely miss?

costs per sample/realization

Need to connect this analysis of Monte Carlo methods to our setting where each sample entails a numerical computation

VR: Cost complexity

Let $w_1 = c(X)$ be the costs of sampling X and let $w_2 = c(Y)$ be the costs of sampling the control variate. $\rightsquigarrow$ board

VR: Cost complexity

Let $w_1 = c(X)$ be the costs of sampling X and let $w_2 = c(Y)$ be the costs of sampling the control variate. $\rightsquigarrow$ board

Plain Monte Carlo achieves MSE $\leq \epsilon$ with costs

$$c(\theta^{\mathrm{MC}}) \leq \frac{\mathrm{Var}[X]}{\epsilon} w_1 \,.$$

Control variates estimator

$$c(\theta^{\mathrm{CV}}) \leq \frac{\mathrm{Var}[X]}{\epsilon}(1-\rho^2)(w_1+w_2)$$

The control variate estimator has lower costs than the plain Monte Carlo estimator if

$$1-\rho^2 < \frac{w_1}{w_1+w_2}$$

The inequality combines ρ and the costs $\rightarrow$ cost vs. accuracy trade-off that we know from numerical analysis

X is RV, Y is RV <u>with known $\mathbb{E}[Y]$</u>

correlation $\rho(X, Y)$

$$\Theta = \bar{X}_m + \alpha^* (\mathbb{E}[Y] - \bar{Y}_m)$$

unbiasedness

$$\mathbb{E}[\Theta] = \mathbb{E}[\bar{X}_m] + \alpha^* (\mathbb{E}[Y] - \mathbb{E}[\bar{Y}_m])$$

$$= \mathbb{E}[\bar{X}_m] = \mathbb{E}[X]$$

MSE

$$e(\Theta) = \mathrm{Var}[\Theta]$$

$$= \mathrm{Var}[\bar{X}_m] + \alpha^2 \mathrm{Var}[\mathbb{E}[Y] - \bar{Y}_m] + 2\,\mathrm{Cov}[\bar{X}_m, \alpha(\mathbb{E}[Y] - \bar{Y}_m)]$$

$$= \ldots = \frac{\mathrm{Var}[X]}{m} + \alpha^2 \frac{\mathrm{Var}[Y]}{m} - 2\alpha \frac{1}{m^2} \sum_{i,j=1}^{m} \mathrm{Cov}[X_i, Y_j]$$

$$= \ldots = \frac{\mathrm{Var}[X]}{m} + \alpha^2 \frac{\mathrm{Var}[Y]}{m} - 2\alpha \frac{1}{m} \rho \sqrt{\mathrm{V}[X]\mathrm{V}[Y]}$$

find α^* by minimizing $e(\Theta)$ w.r.t. α:

$$\partial_\alpha e(\Theta) = 2\alpha \frac{\mathrm{Var}[Y]}{m} - 2 \frac{1}{m} \rho \sqrt{\mathrm{V}[X]\mathrm{V}[Y]}$$

$$\overset{!}{=} 0 : \quad \alpha^* = \rho \sqrt{\frac{\mathrm{V}[X]}{\mathrm{V}[Y]}}$$

plug in: $\quad e(\Theta) = \frac{\mathrm{Var}[X]}{m} (1 - \rho^2)$

Example: Control variates (cont'd)

- $X \sim \mathcal{U}(0,1)$ and $f(x) = 10\sin(x)$
- Now consider $g(x) = 10(x - x^3/6)$ (Taylor approximation)
- Compute mean $\mathbb{E}[g(X)] = 10(1/2 - 1/24)$ analytically
- Set

$$F_i = f(X_i)\,, \qquad G_i = g(X_i)\,, \qquad i = 1, \ldots, m$$

- Set synthetic costs

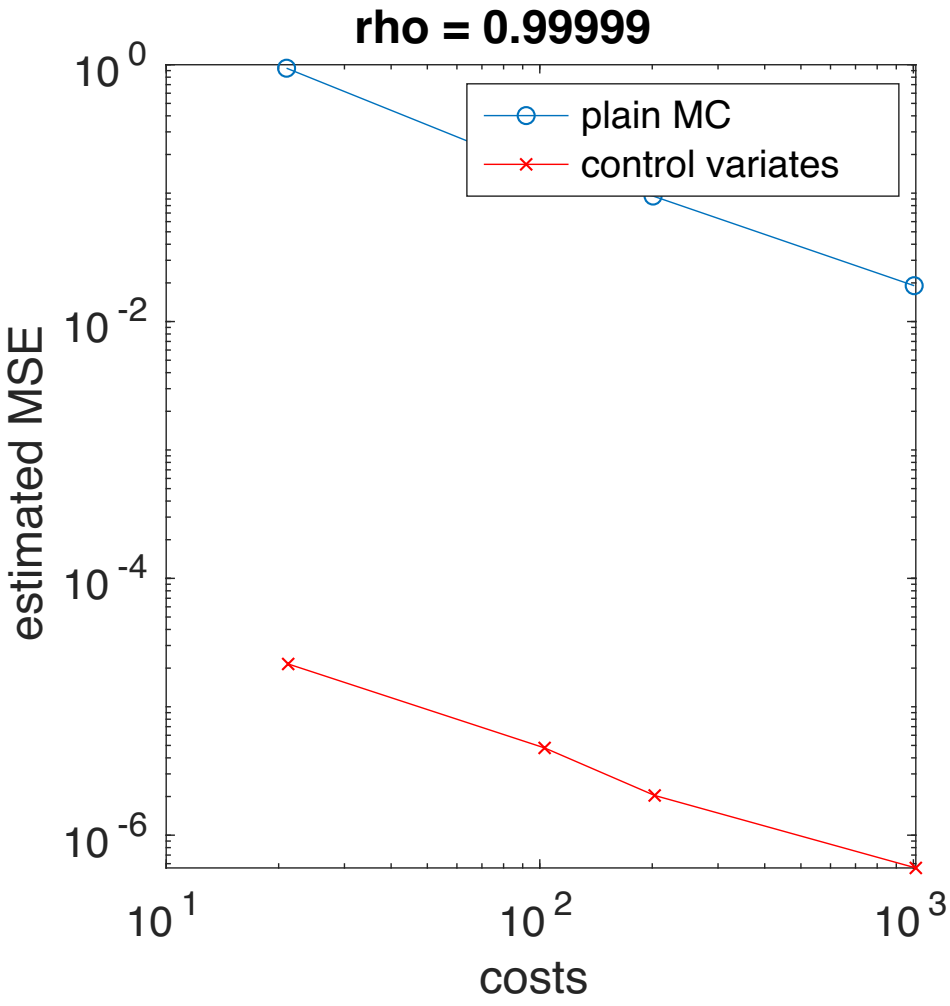
$$w_1 = 3\,, \qquad w_2 = w_1/100$$

- Measure $\rho(f(X), g(X)) \approx 9.999861330445534e - 01$ and $\mathrm{Var}[f(X)]$ and $\mathrm{Var}[g(X)]$
- Control variate estimator

$$\theta = \frac{1}{m}\sum_{i=1}^{m} F_i + \alpha^* \left(\mathbb{E}[g(X)] - \frac{1}{m}\sum_{i=1}^{m} G_i \right)$$

- Compare MSE of θ to plain vanilla Monte Carlo estimator

Example: Numerical example



VR: Control variates with unknown expectation

Assumed we know $\mathbb{E}[g(X)]$, i.e., the mean of the control variate. Typically unavailable; for example if g is an approximation of f. Instead, now $\mathbb{E}[g(X)]$ is unknown and needs to be estimated. We obtain the control variate estimator

$$\theta^{CV} = \bar{F}_{m_1} + \alpha(\underbrace{\bar{G}_{m_2}}_{\text{estimates } \mathbb{E}[G]} - \bar{G}_{m_1})$$

- There are two m's now: m_1 plays the same role as m before; additionally have m_2 which gives the number of samples used to estimate the mean $\mathbb{E}[G]$ of the control variate
- Is this an unbiased estimator of $\mathbb{E}[F]$?
- Optimal choice of α?
- Optimal choice of number of samples m_1 and m_2 to achieve MSE ϵ with minimal costs?

$\rightsquigarrow$ board

$$\Theta = \frac{1}{m_1^*} \sum_{i=1}^{m_1^*} f(X_i) + \alpha^* \left(\frac{1}{m_2^*} \sum_{i=1}^{m_2^*} g(X_i) - \frac{1}{m_1^*} \sum_{i=1}^{m_1^*} g(X_i) \right)$$

samples

$$X_1, \dots, X_{m_1^*}, X_{m_1^*+1}, \dots, X_{m_2^*} \qquad m_1^* \leq m_2^*$$

Unbiasedness:

$$\Theta = \bar{F}_{m_1} + \alpha (\bar{G}_{m_2} - \bar{G}_{m_1})$$

$$\mathbb{E}[\Theta] = \mathbb{E}[\bar{F}_{m_1}] = \mathbb{E}[f(X)]$$

MSE

$$\mathrm{Var}[\Theta] = V[\bar{F}_{m_1}] + \alpha^2 V[\bar{G}_{m_2} - \bar{G}_{m_1}] + 2\alpha \, \mathrm{Cov}[\bar{F}_{m_1}, \bar{G}_{m_2} - \bar{G}_{m_1}]$$

$$= \dots = V[\bar{F}_{m_1}] + \alpha^2 V[\bar{G}_{m_2}] + \alpha^2 V[\bar{G}_{m_1}] - 2\alpha^2 C[\bar{G}_{m_2}, \bar{G}_{m_1}] - 2\alpha C[\bar{F}_{m_1}, \bar{G}_{m_1}] + 2\alpha C[\bar{F}_{m_1}, \bar{G}_{m_2}]$$

$$\vdots$$

$$C[\bar{G}_{m_2}, \bar{G}_{m_1}] = \dots = \frac{1}{m_2} V[g(X)]$$

similar transformation:

$$\mathrm{Var}[\Theta] = \frac{1}{m_1}\left(V[f(x)] + \alpha^2 V[g(x)] - 2\alpha\rho\sqrt{V[f]V[g]}\right)$$
$$- \frac{1}{m_2}\left(\alpha^2 V[g(x)] - 2\alpha\rho\sqrt{V[f]V[g]}\right)$$

$$\left.\begin{array}{ll} \min\limits_{m_1, m_2, \alpha} & m_1 w_1 + m_2 w_2 \\ \text{such that} & V[\Theta_{m_1, m_2, \alpha}] \le \varepsilon \end{array}\right\} \text{this is the key step!}$$

Results: m_1^*, m_2^*, α^*

$$m_1^* = \frac{V[f(x)]}{\varepsilon}\left(1 - \left(1 - \frac{1}{r^*}\right)\rho^2\right)$$

$$m_2^* = m_1^* r^*$$

$$C_\varepsilon(\Theta) \le m_1^* w_1 + m_2^* w_2 = \frac{V[f(x)]}{\varepsilon}\left(1 - \left(1 - \frac{1}{r^*}\right)\rho^2\right)\left(w_1 + r^* w_2\right)$$

$$r^* = \sqrt{\frac{w_1 \rho^2}{w_2(1-\rho^2)}}$$

VR: Properties of control variate estimator

Consider two functions f and g with evaluation costs w_1 and w_2, respectively. Assume $w_1 > w_2$. Consider the random variable X and set $\rho = \rho(f(X), g(X))$ and let σ_F and σ_G be the standard deviation of $f(X)$ and $g(X)$, respectively. Set

$$\alpha^* = \rho \frac{\sigma_F}{\sigma_G}\,, \qquad r^* = \sqrt{\frac{w_1 \rho^2}{w_2(1-\rho^2)}}$$

The estimator

$$\theta = \frac{1}{m_1^*} \sum_{i=1}^{m_1^*} f(X_i) + \alpha^* \left(\frac{1}{m_2^*} \sum_{i=1}^{m_2^*} g(X_i) - \frac{1}{m_1^*} \sum_{i=1}^{m_1^*} g(X_i) \right)$$

is unbiased w.r.t. $\mathbb{E}[f(X)]$ if $X_1, \ldots, X_{m_1}, X_{m_1+1}, \ldots, X_{m_2} \sim X$ and achieves an MSE $e(\theta) \leq \epsilon$ with costs

$$c_\epsilon(\theta) \leq \frac{\sigma_F^2}{\epsilon} \left(1 - \left(1 - \frac{1}{r^*} \right) \rho^2 \right) (w_1 + r^* w_2)$$

if

$$m_1^* = \frac{\sigma_F^2}{\epsilon} \left(1 - \left(1 - \frac{1}{r^*} \right) \rho^2 \right)\,, m_2^* = m_1^* r_*\,.$$

The m_1^*, m_2^*, α^* are optimal in the sense that they minimize the costs $c_\epsilon(\theta)$.
Note that control variate reuses samples

VR: Comparison

Plain vanilla Monte Carlo

$$c_\epsilon(\theta) \leq \frac{\sigma_F^2}{\epsilon} w_1$$

Control variates if mean of control variate known

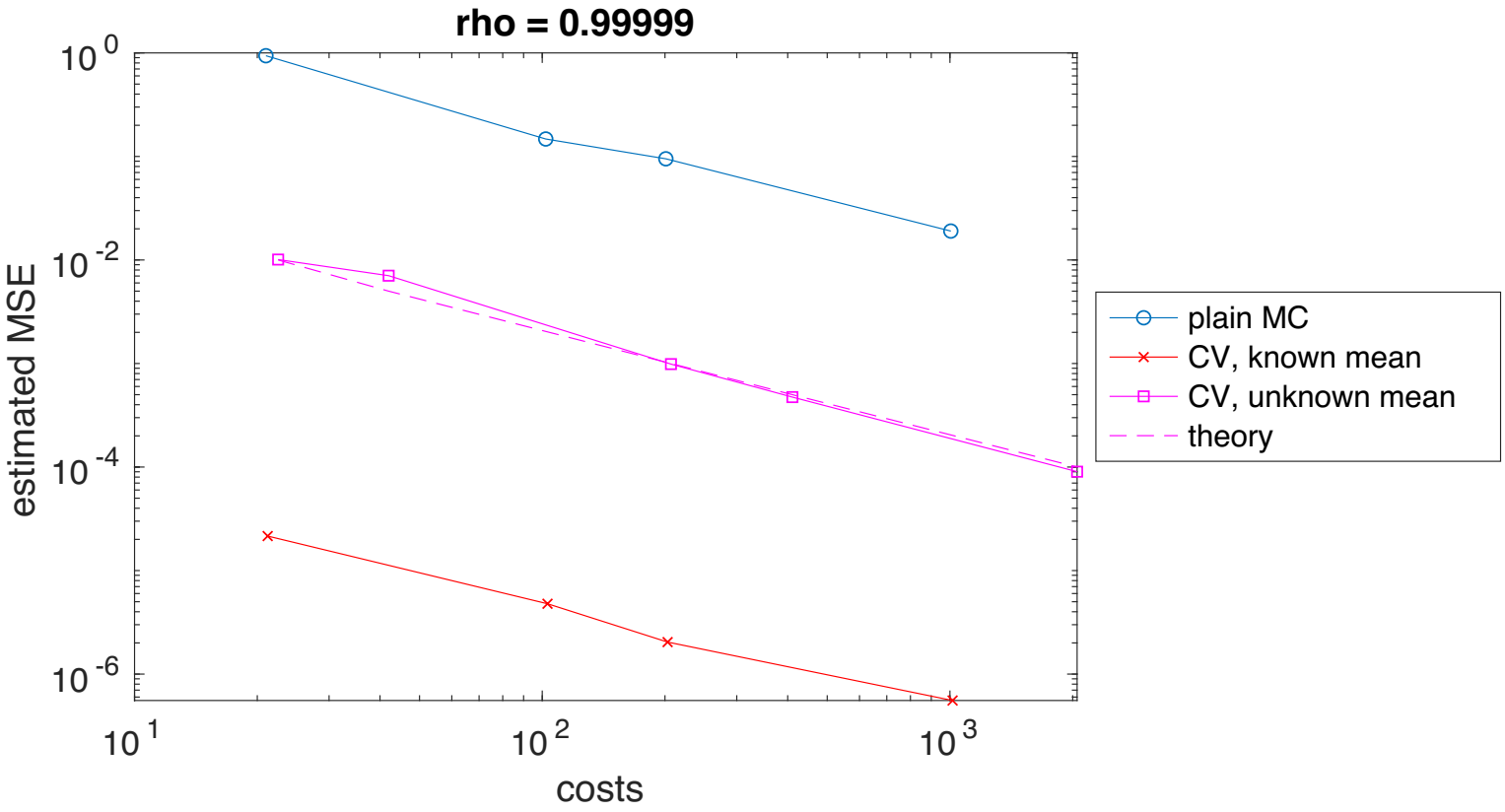
$$c_\epsilon(\theta) \leq \frac{\sigma_F^2}{\epsilon}(1-\rho^2)(w_1 + w_2)$$

Control variates if mean of control variate unknown

$$c_\epsilon(\theta) \leq \frac{\sigma_F^2}{\epsilon}\left(1 - \left(1 - \frac{1}{r^*}\right)\rho^2\right)(w_1 + r^* w_2)$$

Notice that the rate with respect to ϵ is the same but the constants change

VR: Example (cont'd)



Today

Last time

- Monte Carlo

Today

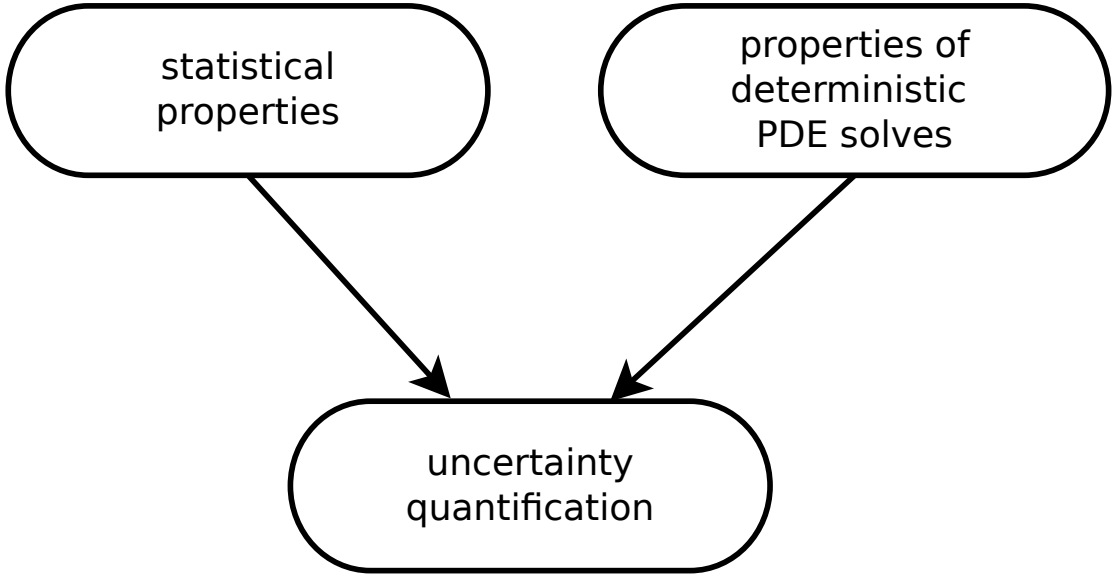
- Multi-level Monte Carlo

Announcements

- Homework 7 is posted and due Mon, Dec 9 before class (1 week)
- Next week, Mon Dec 9, recap of important topics—highly recommended!

What follows won't be on the final exam

Monte Carlo + Numerical Analysis

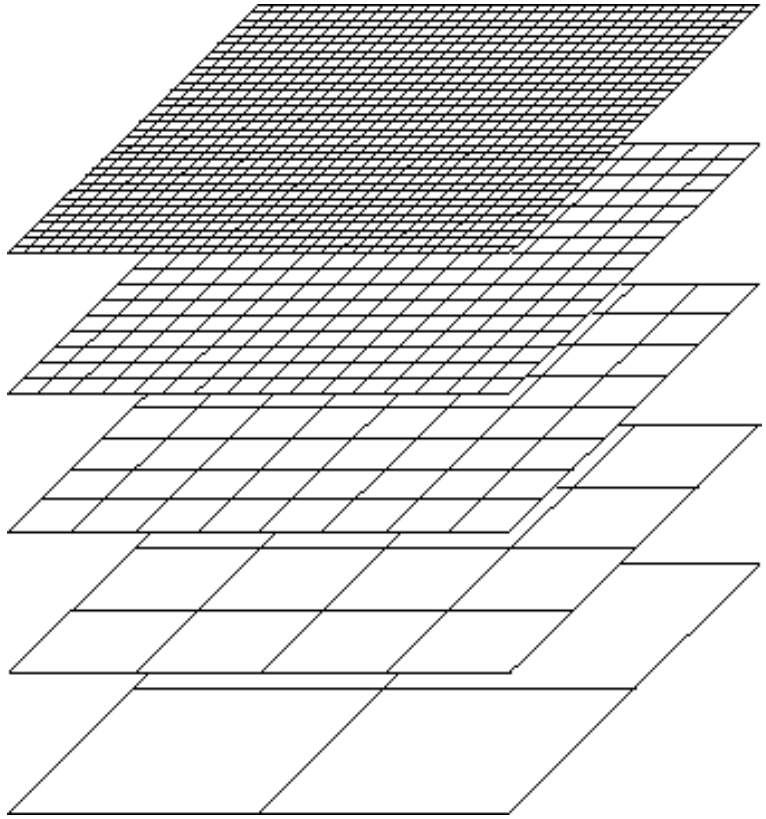


- Bounding **costs** instead of, e.g., number of samples
- Need to take into account numerical PDE solver costs

Level

Instead of describing the discretization with the number of degrees of freedom N, we describe it by the level $\ell \in \mathbb{N}$

- Level $\ell \in \mathbb{N}$, mesh width $h = 2^{-\ell}$
- Number of degrees of freedom in 1-dimensional spatial domain $N = 2^\ell + 1 \in \mathcal{O}(2^\ell)$
- In 2-dimensional spatial domain $N = (2^\ell + 1)^2 \in \mathcal{O}(4^\ell)$
- In 3-dimensional spatial domain $N = (2^\ell + 1)^3 \in \mathcal{O}(8^\ell)$
- General $N \in \mathcal{O}(s^\ell)$ with $s \in \mathbb{N}$



A first analysis of MC in the context of UQ

The following summary follows the analysis in [Cliffe et al., 2011].

To estimate the expectation $\mathbb{E}[Q]$ of a (random) quantity of interest Q, assume only approximations $Q_\ell \approx Q$ are computable, where $\ell \in \mathbb{N}$ is a discretization level such that

$$\lim_{\ell \to \infty} \mathbb{E}[Q_\ell] = \mathbb{E}[Q] \,.$$

More precisely, we assume the error converges in mean (in distribution) with a rate $\alpha > 0$

$$|\mathbb{E}[Q - Q_\ell]| \lesssim s^{-\ell\alpha} \,,$$

where $\lesssim$ means up to constants independent of ℓ and α.

With increasing ℓ, the costs of solving the corresponding system may increase

$$c(Q_\ell) \lesssim s^{\ell\gamma} \,, \qquad \gamma > 0$$

A first analysis of MC (cont'd)

- Let $\bar{Q}_\ell$ be an unbiased estimator of $\mathbb{E}[Q_\ell]$
- Summary:
  - Q is the QoI that is unavailable ("continuous solution of PDE solution")
  - Q_ℓ is the approximation of Q ("numerical approximation")
  - $\mathbb{E}[Q]$ is what we want to estimate ("no numerical error; no statistical error")
  - $\mathbb{E}[Q_\ell]$ is mean of Q_ℓ ("no statistical error")
  - $\bar{Q}_\ell$ is an unbiased estimator of $\mathbb{E}[Q_\ell]$ ("Monte Carlo estimator of $\mathbb{E}[Q_\ell]$")
- The mean-squared error (MSE) of $\bar{Q}_\ell$ is

$$\mathbb{E}[(\bar{Q}_\ell - \mathbb{E}[Q])^2] = \underbrace{\mathrm{Var}[\bar{Q}_\ell]}_{\text{variance}} + \underbrace{\mathbb{E}[Q_\ell - Q]^2}_{\text{bias}}$$

  - Measure bias w.r.t. $\mathbb{E}[Q]$
  - Bias w.r.t. $\mathbb{E}[Q_\ell]$ is zero because $\bar{Q}_\ell$ unbiased by assumption
- How can we control the variance and bias?

A first analysis of MC (cont'd)

- Let $\bar{Q}_\ell$ be an unbiased estimator of $\mathbb{E}[Q_\ell]$
- Summary:
  - Q is the QoI that is unavailable ("continuous solution of PDE solution")
  - Q_ℓ is the approximation of Q ("numerical approximation")
  - $\mathbb{E}[Q]$ is what we want to estimate ("no numerical error; no statistical error")
  - $\mathbb{E}[Q_\ell]$ is mean of Q_ℓ ("no statistical error")
  - $\bar{Q}_\ell$ is an unbiased estimator of $\mathbb{E}[Q_\ell]$ ("Monte Carlo estimator of $\mathbb{E}[Q_\ell]$")
- The mean-squared error (MSE) of $\bar{Q}_\ell$ is

$$\mathbb{E}[(\bar{Q}_\ell - \mathbb{E}[Q])^2] = \underbrace{\mathrm{Var}[\bar{Q}_\ell]}_{\text{variance}} + \underbrace{\mathbb{E}[Q_\ell - Q]^2}_{\text{bias}}$$

  - Measure bias w.r.t. $\mathbb{E}[Q]$
  - Bias w.r.t. $\mathbb{E}[Q_\ell]$ is zero because $\bar{Q}_\ell$ unbiased by assumption
- How can we control the variance and bias?
  - Variance: Improve estimator $\bar{Q}_\ell$ of $\mathbb{E}[Q_\ell] \rightarrow$ statistics
  - Bias: Improve accuracy of Q_ℓ w.r.t. $Q \rightarrow$ deterministic solver

Cost complexity of basic Monte Carlo

Assume that

- (A0) the variance $\mathrm{Var}[Q_\ell]$ is constant w.r.t. ℓ,
- (A1) $|\mathbb{E}[Q - Q_\ell]| \lesssim s^{-\ell\alpha}\,, \qquad \alpha > 0,$
- (A2) $c(Q_\ell) \lesssim s^{\ell\gamma}\,, \qquad \gamma > 0\,.$

Consider the Monte Carlo estimator $\bar{Q}^{\mathrm{MC}}_{\ell,m}$ $(=\bar{Q}_{\ell,m})$ of $\mathbb{E}[Q_\ell]$ with $m \in \mathbb{N}$ iid copies $Q_\ell^{(1)}, \ldots, Q_\ell^{(m)}$ of Q_ℓ. For $\epsilon > 0$, the Monte Carlo estimator $\bar{Q}^{\mathrm{MC}}_{\ell,m}$ achieves the mean-squared error (MSE)

$$\mathbb{E}[(\bar{Q}^{\mathrm{MC}}_{\ell,m} - \mathbb{E}[Q])^2] \leq \epsilon$$

with costs

$$c_\epsilon(\bar{Q}^{\mathrm{MC}}_{\ell,m}) \lesssim \epsilon^{-1-\gamma/(2\alpha)}\,.$$

In terms of the root-mean-squared error (RMSE), the costs are bounded by $\epsilon^{-2-\gamma/\alpha}$.

know that

$$\mathrm{Var}[\bar{Q}^{MC}_{\underline{\ell},\underline{m}}] = \frac{\mathrm{Var}[Q_\ell]}{m}$$

and thus

$$\mathbb{E}[(\bar{Q}^{MC}_{\ell,m} - \mathbb{E}[Q])^2] = \underbrace{\frac{\mathrm{Var}[Q_\ell]}{m}}_{\text{depends on } m \text{ only (A0)}} + \underbrace{\mathbb{E}[Q_\ell - Q]^2}_{\text{only depends on the level}}$$

want to find ℓ and m such that

$$\underline{\mathbb{E}[(\bar{Q}^{MC}_{\ell,m} - \mathbb{E}[Q])^2] \leq \varepsilon}$$

$\Rightarrow$ balance bias and variance

(1) Variance

$$\frac{\mathrm{Var}[Q_\ell]}{m} \leq \frac{\varepsilon}{2} \iff \underline{2\,\mathrm{Var}[Q_\ell]\,\varepsilon^{-1} \leq m} \quad (A0)$$

(2) Bias

$$\mathbb{E}[Q - Q_\ell]^2 \overset{(A1)}{\leq} (c_1 s^{-\ell\alpha})^2 \overset{!}{\leq} \frac{\varepsilon}{2}$$

$$\iff \underline{\varepsilon^{-\frac{1}{2\alpha}} (\sqrt{2}\, c_1)^{1/\alpha} \leq s^\ell}$$

Invoke (A2)

(A2)

$$c(\bar{Q}^{MC}_{\ell,m}) \le \underbrace{m}_{\#\text{samples}} \; \underbrace{c_2 s^{\ell\gamma}}_{\text{cost per sample}}$$

to achieve $\le \varepsilon$, set $m = \underline{2\,\mathrm{Var}[Q_\ell]\,\varepsilon^{-1}}$ and set level ℓ such that

$$\underline{s^\ell = \varepsilon^{-1/2\alpha}(\sqrt{2}\,c_1)^{1/\alpha}}$$

$$c_\varepsilon(\bar{Q}^{MC}_{\ell,m}) \le c_2 \underbrace{2\,\mathrm{Var}[Q_\ell]\,\varepsilon^{-1}}_{=\underline{m}} \underbrace{\varepsilon^{-\gamma/2\alpha}(\sqrt{2}\,c_1)^{\gamma/\alpha}}_{=\underline{s}^{\ell\gamma}}$$

$$\lesssim \varepsilon^{-1-\gamma/2\alpha}$$

Improving plain Monte Carlo: Variance reduction

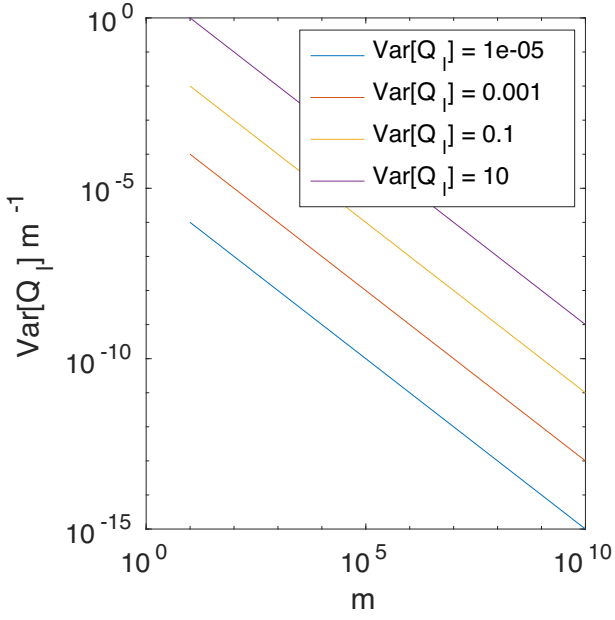
Convergence of plain vanilla Monte Carlo

$$\mathbb{E}[(\bar{Q}^{\mathrm{MC}}_{\ell,m} - \mathbb{E}[Q])^2] = \mathrm{Var}[\bar{Q}^{\mathrm{MC}}_{\ell,m}] + (\mathbb{E}[\bar{Q}^{\mathrm{MC}}_{\ell,m} - \mathbb{E}[Q])^2$$

$$= \underbrace{\frac{\mathrm{Var}[Q_\ell]}{m}}_{\text{sampling error (variance)}} + \underbrace{(\mathbb{E}[Q_\ell - Q])^2}_{\text{model error (bias)}}$$

- Convergence rate of plain vanilla Monte Carlo is $\mathcal{O}(m^{-1})$ in MSE
- The constant $\mathrm{Var}[Q_\ell]$ plays a critical role

$\Rightarrow$ it is worthwhile trying to reduce *constants* $\Rightarrow$ variance reduction



MLMC: Multilevel Monte Carlo

- Multilevel Monte Carlo (MLMC) uses control variates in a judicious way
- First ideas for high-dimensional quadrature by Heinrich, 2000

  Heinrich S. (2001) Multilevel Monte Carlo Methods. In: Margenov S., Waśniewski J., Yalamov P. (eds) Large-Scale Scientific Computing. LSSC 2001. Lecture Notes in Computer Science, vol 2179. Springer, Berlin, Heidelberg

- Independently discovered and popularized by Giles, 2007 in the context of stochastic differ-
  ential equations in mathematical finance (one of the most influential papers in UQ!)

  Michael B. Giles, Multilevel Monte Carlo Path Simulation, Operations Research 2008 56:3, 607-617

- First papers in the context of UQ
  - Cliffe, K.A., Giles, M.B., Scheichl, R. et al. Comput. Visual Sci. (2011) 14: 3. https://doi.org/10.1007/s00791-011-0160-x
  - Barth, A., Schwab, C. & Zollinger, N. Numer. Math. (2011) 119: 123. https://doi.org/10.1007/s00211-011-0377-0

MLMC: Multilevel Monte Carlo

- Multilevel Monte Carlo (MLMC) uses control variates in a judicious way
- First ideas for high-dimensional quadrature by Heinrich, 2000

  Heinrich S. (2001) Multilevel Monte Carlo Methods. In: Margenov S., Waśniewski J., Yalamov P. (eds) Large-Scale Scientific Computing. LSSC 2001. Lecture Notes in Computer Science, vol 2179. Springer, Berlin, Heidelberg

- Independently discovered and popularized by Giles, 2007 in the context of stochastic differ-
  ential equations in mathematical finance (one of the most influential papers in UQ!)

  Michael B. Giles, Multilevel Monte Carlo Path Simulation, Operations Research 2008 56:3, 607-617

- First papers in the context of UQ
  - Cliffe, K.A., Giles, M.B., Scheichl, R. et al. Comput. Visual Sci. (2011) 14: 3. https://doi.org/10.1007/s00791-011-0160-x
  - Barth, A., Schwab, C. & Zollinger, N. Numer. Math. (2011) 119: 123. https://doi.org/10.1007/s00211-011-0377-0

MLMC: Multilevel estimator

- **Key idea:** use realizations of Q_ℓ on a hierarchy of different levels, i.e., for different values $\ell = 1, \dots, L$ of the discretization parameter ($\rightarrow$ recall sparse grids)
- Make the following decomposition (telescoping sum)

$$\mathbb{E}[Q_{N_L}] = \mathbb{E}[Q_{N_0}] + \sum_{\ell=1}^{L} \mathbb{E}[Q_{N_\ell} - Q_{N_{\ell-1}}] = \sum_{\ell=0}^{L} \mathbb{E}[Y_\ell]\,,$$

where $N_0 \in \mathbb{N}$ and $N_\ell = sN_{\ell-1}$ for $\ell = 1, \dots, L$ and $s \in \mathbb{N} \setminus \{1\}$ and

$$Y_0 = Q_{N_0}\,, \qquad Y_\ell = Q_{N_\ell} - Q_{N_{\ell-1}}$$

- Given (unbiased) estimators $\bar{Y}_{\ell,m_\ell}$ for $\mathbb{E}[Y_\ell]$, the estimator

$$\bar{Q}^{\mathrm{ML}}_{L,\boldsymbol{m}} = \sum_{\ell=0}^{L} \bar{Y}_{\ell,m_\ell}$$

is a multilevel estimator of Q with $\boldsymbol{m} = [m_0, \dots, m_L]$

MLMC: Multilevel Monte Carlo estimator

- All estimators $\bar{Y}_{\ell,m_\ell}$ sampled independently, then

$$\mathrm{Var}[\bar{Q}^{\mathrm{ML}}_{L,\boldsymbol{m}}] = \sum_{\ell=0}^{L} \mathrm{Var}[\bar{Y}_{\ell,m_\ell}]$$

- If each $\bar{Y}_\ell$ is a plain Monte Carlo estimator

$$\bar{Y}_{0,m_0} = \frac{1}{m_0} \sum_{i=1}^{m_0} Q^{(i)}_{N_0}\,, \qquad Q^{(i)}_{N_0} \sim Q_{N_0}\,,$$

and

$$\bar{Y}_{\ell,m_\ell} = \frac{1}{m_\ell} \sum_{i=1}^{m_\ell} Q^{(i)}_{N_\ell} - Q^{(i)}_{N_{\ell-1}}\,,$$

one obtains a multilevel Monte Carlo estimator

- The MSE of the multilevel Monte Carlo estimator is

$$\mathbb{E}\left[\left(\bar{Q}^{\mathrm{MLMC}}_{L,\boldsymbol{m}} - \mathbb{E}[Q]\right)^2\right] = \sum_{\ell=0}^{L} \frac{\mathrm{Var}[Y_\ell]}{m_\ell} + \mathbb{E}[Q_L - Q]^2$$

MLMC: Variance reduction

$$\mathbb{E}\left[\left(\bar{Q}_{L,\boldsymbol{m}}^{\mathrm{MLMC}} - \mathbb{E}[Q]\right)^2\right] = \sum_{\ell=0}^{L} \frac{\mathrm{Var}[Y_\ell]}{m_\ell} + \mathbb{E}[Q_L - Q]^2$$

Why do we have hope for variance reduction (=lower cost for same variance)?

MLMC: Variance reduction

$$\mathbb{E}\left[\left(\bar{Q}_{L,\boldsymbol{m}}^{\mathrm{MLMC}} - \mathbb{E}[Q]\right)^2\right] = \sum_{\ell=0}^{L} \frac{\mathrm{Var}[Y_\ell]}{m_\ell} + \mathbb{E}[Q_L - Q]^2$$

Why do we have hope for variance reduction (=lower cost for same variance)?

- As we coarsen the problem, the cost per sample decays rapidly from level to level $\rightarrow$ observed this already in the control variates example $\rightarrow$ sampling gets cheaper and cheaper

MLMC: Variance reduction

$$\mathbb{E}\left[\left(\bar{Q}_{L,\boldsymbol{m}}^{\mathrm{MLMC}} - \mathbb{E}[Q]\right)^2\right] = \sum_{\ell=0}^{L} \frac{\mathrm{Var}[Y_\ell]}{m_\ell} + \mathbb{E}[Q_L - Q]^2$$

Why do we have hope for variance reduction (=lower cost for same variance)?

- As we coarsen the problem, the cost per sample decays rapidly from level to level $\rightarrow$ observed this already in the control variates example $\rightarrow$ sampling gets cheaper and cheaper

- Since $Q_{N_\ell} \rightarrow Q$, we have $\mathrm{Var}[Y_\ell] = \mathrm{Var}[Q_{N_\ell} - Q_{N_{\ell-1}}] \rightarrow 0$ going to 0 fast for $\ell \rightarrow \infty$, allowing for smaller and smaller sample sizes m_ℓ to estimate the *difference* $Q_{N_\ell} - Q_{N_{\ell-1}}$ on finer and finer (more and more expensive) levels.

MLMC: Cost complexity

Theorem (Giles 2008; Cliffe, Giles, Scheichl, Teckentrup 2011)

Let $\bar{Y}_\ell = \bar{Y}^{MC}_{\ell,m_\ell}$ *and suppose that there are positive constants* $\alpha, \beta, \gamma > 0$ *such that* $\alpha \geq \frac{1}{2}\min(\beta, \gamma)$ *and*

- *(A1)* $|\mathbb{E}[Q_{N_\ell} - Q]| \lesssim N_\ell^{-\alpha}$
- *(A2)* $\mathrm{Var}[Y_\ell] \lesssim N_\ell^{-\beta}$
- *(A3)* $w_\ell \lesssim N_\ell^{\gamma}$.

Then, for any $\sqrt{\epsilon} < \mathrm{e}^{-1}$*, there exist a value* $L \in \mathbb{N}$ *and* $\boldsymbol{m} = [m_0, \ldots, m_L]$ *such that*

$$\mathbb{E}\left[\left(\bar{Q}^{MLMC}_{L,\boldsymbol{m}} - \mathbb{E}[Q]\right)\right] < \epsilon\,,$$

and

$$c_\epsilon(\bar{Q}^{MLMC}_{L,\boldsymbol{m}}) \lesssim \begin{cases} \epsilon^{-1} & \text{if } \beta > \gamma,\ , \\ \epsilon^{-1}(\log \epsilon^{1/2})^2\,, & \text{if } \beta = \gamma\,, \\ \epsilon^{-1-(\gamma-\beta)/(2\alpha)}\,, & \text{if } \beta < \gamma\,. \end{cases}$$

(A1) $|\mathbb{E}[Q_{N_\ell} - Q]| \lesssim c_1 N_\ell^{-\alpha}$

(A2) $\mathrm{Var}[Y_\ell] \lesssim c_2 N_\ell^{-\beta}$

(A3) $w_\ell \lesssim c_3 N_\ell^{\gamma}$

$$\mathbb{E}\left[\left(\bar{Q}^{MLMC}_{L,m} - \mathbb{E}[Q]\right)^2\right] < \varepsilon$$

and costs

$$c_\varepsilon\left(\bar{Q}^{MLMC}_{L,m}\right) \lesssim \begin{cases} \varepsilon^{-1}, & \text{if } \beta > \gamma \\ \vdots \end{cases}$$

Recall $N_\ell = s N_{\ell-1}$, $\ell = 1, \dots, L$, $N_0 = 1$

Approach: for each case, bound bias by $\frac{\varepsilon}{2}$ and variance by $\frac{\varepsilon}{2}$ and then bound costs

Bias: Set

hidden constant

$$L = \lceil \alpha^{-1} \log_s(\varepsilon^{-1/2} \sqrt{2}\, c_1) \rceil$$

$$< \alpha^{-1} \log_s(\varepsilon^{-1/2} \sqrt{2}\, c_1) + 1$$

$$\mathbb{E}[Q_L - Q]^2 \leq \frac{\varepsilon}{2}$$

Variance

case $\underline{\beta > \gamma}$: Set

$$\underline{m_\ell} = \left\lceil 2\,\varepsilon^{-1} c_2 \left(1 - s^{-(\beta-\gamma)/2}\right)^{-1} \underline{s^{-(\beta+\gamma)\ell/2}} \right\rceil$$

$$\underline{\sum_{\ell=0}^{L} \mathrm{Var}[\bar{Y}_{\ell,m}]} = \sum_{\ell=0}^{L} \underbrace{\underline{m_\ell^{-1}}\ \underline{\mathrm{Var}[Y_\ell]}}_{\text{balance}} \le \sum_{\ell=0}^{L} m_\ell^{-1} c_2 s^{-\beta\ell}$$

$$\le \frac{1}{2}\varepsilon\left(1 - s^{-(\beta-\gamma)/2}\right) \sum_{\ell=0}^{L} c_2 \underbrace{s^{-\beta\ell}\, s^{-(\beta+\gamma)\ell/2}}_{s^{-(\beta-\gamma)\ell/2}}$$

$$= \frac{1}{2}\varepsilon\left(\cancel{1 - s^{-(\beta-\gamma)/2}}\right) \underbrace{\sum_{\ell=0}^{L} \left(s^{-(\beta-\gamma)/2}\right)^{\ell}}_{\dfrac{1 - s^{-(\beta-\gamma)/2(L+1)}}{\cancel{1 - s^{-(\beta-\gamma)/2}}}}$$

$$\le \underline{\frac{1}{2}\varepsilon}$$

Costs:

$$c\left(\bar{Q}_{L,m}^{\mathrm{MLMC}}\right) = \sum_{\ell=0}^{L} \underline{w_\ell}\, m_\ell$$

$$\leq \sum_{\ell=0}^{L} c_3\, s^{\ell\gamma} \left(2\varepsilon^{-1} c_2 (1 - s^{-(\beta-\gamma)/2})^{-1} s^{-(\beta+\gamma)\ell/2} + 1\right)$$

$$\leq c_3\, 2\varepsilon^{-1} c_2 (1 - s^{-(\beta-\gamma)/2})^{-1} \underbrace{\sum_{\ell=0}^{L} \underbrace{s^{-(\beta+\gamma)\ell/2}\, s^{\ell\gamma}}_{\underbrace{s^{-(\beta-\gamma)\ell/2}}_{\text{geo. series}}}}_{\frac{1 - s^{-(\beta-\gamma)/2(L+1)}}{(1 - s^{-(\beta-\gamma)/2})}} + c_3 \underbrace{\sum_{\ell=0}^{L} s^{\ell\gamma}}$$

$$= c_3\, 2\varepsilon^{-1} c_2 (1 - s^{-(\beta-\gamma)/2})^{-2} \underbrace{\left(1 - \underbrace{s^{-(\beta-\gamma)/2(L+1)}}_{<1}\right)}_{>0} + c_3 \varepsilon \ldots$$

$$\leq C\varepsilon^{-1} + \underbrace{c_3 \sum_{\ell=0}^{L} s^{\ell\gamma}}_{\text{because of rounding } \lceil\ \rceil} \leq C\varepsilon^{-1} \lesssim \varepsilon^{-1}$$

$\rightarrow$ technical lemma

for the other cases: $\beta > \gamma$, $\beta = \gamma$

MLMC: Interpreting the cost complexity

For idealized flows through a porous medium (Darcy flow):

- Numerically observed deterministic error $\Rightarrow$ rate $\alpha \approx 3/8$
- Numerically observed cost/sample $\Rightarrow$ rate $\gamma \approx 1$

Consideration generalization of the problem in $d = 1, 2, 3$ dimensional spatial domain. It has been observed $\alpha \approx 3/4d^{-1}, \gamma \approx 1, \beta \approx 2\alpha$.

Using the complexity theorems of MC and MLMC and the costs per sample $\epsilon^{-\gamma/(2\alpha)}$ obtain

d	MC	MLMC	per sample on finest level
1	$\mathcal{O}(\epsilon^{-5/3})$	$\mathcal{O}(\epsilon^{-1})$	$\mathcal{O}(\epsilon^{-2/3})$
2	$\mathcal{O}(\epsilon^{-7/3})$	$\mathcal{O}(\epsilon^{-4/3})$	$\mathcal{O}(\epsilon^{-4/3})$
3	$\mathcal{O}(\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$

MLMC: Interpreting the cost complexity

For idealized flows through a porous medium (Darcy flow):

- Numerically observed deterministic error $\Rightarrow$ rate $\alpha \approx 3/8$
- Numerically observed cost/sample $\Rightarrow$ rate $\gamma \approx 1$

Consideration generalization of the problem in $d = 1, 2, 3$ dimensional spatial domain. It has been observed $\alpha \approx 3/4d^{-1}, \gamma \approx 1, \beta \approx 2\alpha$.

Using the complexity theorems of MC and MLMC and the costs per sample $\epsilon^{-\gamma/(2\alpha)}$ obtain

d	MC	MLMC	per sample on finest level
1	$\mathcal{O}(\epsilon^{-5/3})$	$\mathcal{O}(\epsilon^{-1})$	$\mathcal{O}(\epsilon^{-2/3})$
2	$\mathcal{O}(\epsilon^{-7/3})$	$\mathcal{O}(\epsilon^{-4/3})$	$\mathcal{O}(\epsilon^{-4/3})$
3	$\mathcal{O}(\epsilon^{-3})$	$\mathcal{O}(\epsilon^{-2})$	$\mathcal{O}(\epsilon^{-2})$

MLMC costs asymptotically the same as **one** deterministic solve to accuracy ϵ for $d > 1 \rightarrow$ "UQ is for free"

Today

Today

- Review of *some* important topics

Announcements

- Be 10min earlier in the room for the final exam
- Mind that the room has been updated; check Albert
- Double check that all homeworks are entered correctly in brightspace

There are many opportunities for asking questions

Mon, Dec 2	office hour
Wed, Dec 4	lecture (virtual) [not part of final exam]
Fri, Dec 6	office hour (grader)
Mon, Dec 9	recap and Q&A
Mon, Dec 9	office hour
Wed, Dec 11	extra office hour, 6.10pm ET (WWH 421) (all HWs graded; no HW re-grading after Thu, Dec 12)
Mon, Dec 16	final exam

- Condition and stability
- Linear systems and linear least-squares problems
- Eigenproblems
- Iterative methods for linear systems and nonlinear systems
- Interpolation
- Quadrature

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation?

Condition of a problem

- Consider a generic problem: given F and data/input x, find output y such that

$$F(x, y) = 0$$

- Let's assume there is a unique solution so that we can write

$$y = f(x)\,,$$

  for a function f in the following
- Well-posed: Unique solution + If we perturb the input x a little bit, the solution y gets perturbed by a small amount.
- Otherwise, the problem is ill-posed; no numerical method can help with that. (What should we do in such a situation? $\rightsquigarrow$ change the problem)

Condition of a problem (cont'd)

- Absolute condition number at x is

$$\kappa_{\text{abs}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\|}{\|x - \hat{x}\|}$$

- Relative condition number at x is

$$\kappa_{\text{rel}} = \lim_{\delta \to 0} \sup_{\|x - \hat{x}\| \le \delta} \frac{\|f(x) - f(\hat{x})\| / \|f(x)\|}{\|x - \hat{x}\| / \|x\|}$$

- If f is differentiable in x, then

$$\kappa_{\text{abs}} = \|f'(x)\| \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\| \,,$$

where $\|f'(x)\|$ is the norm of the Jacobian $f'(x)$ in the operator norm

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|$$

Revisiting stability

An algorithm $\tilde{f}$ for a problem f is backward stable if for each $x \in X$ we have $\tilde{f}(x) = f(\tilde{x})$ for an $\tilde{x}$ with

$$\frac{\|\tilde{x} - x\|}{\|x\|} \in \mathcal{O}(u)\,,$$

where u is the roundoff unit

- Recall that, loosely speaking, the symbol $\mathcal{O}(u)$ means "on the order of the roundoff unit."
- By allowing $u \to 0$ (which is implied here by the $\mathcal{O}$), we consider an idealization of a computer (in practice, u is fixed). So what we mean is that the error should decrease in proportion to u or faster.

Stability + condition

Suppose a backward stable algorithm is applied to solve a problem $f : X \to Y$ with relative condition number κ. Then, the relative errors satisfy

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \in \mathcal{O}(\kappa(x)u)\,.$$

Condition of solving system of linear equations

Recall that we derived the condition number $\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|$ of a matrix $\boldsymbol{A}$

Problem 1: fix A, consider $b \mapsto A^{-1}b$

$f: \mathbb{R}^n \to \mathbb{R}^n$, $f(b) = A^{-1}b$

$k_{abs} = \|A^{-1}\|$

$k_{rel} = \frac{\|b\|}{\|A^{-1}b\|}\|A^{-1}\| \leq \|A\|\,\|A^{-1}\| = k(A)$

Problem 2: fix b, $A \mapsto A^{-1}b$

$k_{rel} \leq \|A\|\,\|A^{-1}\| = k(A)$

Problem 1: Show that $\rho(A) \leq \|A\|$ for any induced matrix norm, where $\rho(A)$ is the spectral radius of A (the largest absolute eigenvalue).

$$\|A\| = \sup_{x \neq} \frac{\|Ax\|}{\|x\|} \geq \frac{\|Ax_1\|}{\|x_1\|} = \frac{\|\rho(A)x_1\|}{\|x_1\|} = \rho(A)$$

Solving systems of linear equations

Gauss elimination and LU factorization

multiply first row by ℓ_{i1} and subtract

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \begin{array}{l} \\ \leftarrow \boxed{\ell_{21}} = \dfrac{a_{21}^{(1)}}{a_{11}^{(1)}} \\ \leftarrow \boxed{\ell_{31}} = \dfrac{a_{31}^{(1)}}{a_{11}} \end{array}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} - \ell_{21} a_{11}^{(1)} = a_{21}^{(1)} - \frac{a_{21}^{(1)}}{a_{11}^{(1)}} a_{11}^{(1)} = 0 & a_{22}^{(1)} - \ell_{21} a_{12}^{(1)} & a_{23}^{(1)} - \ell_{21} a_{13}^{(1)} \\ 0 & a_{32}^{(1)} - \ell_{31} a_{12}^{(1)} & a_{33}^{(1)} - \ell_{31} a_{13}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} - \ell_{21} b_1^{(1)} \\ b_3^{(1)} - \ell_{31} b_1^{(1)} \end{bmatrix}$$

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures existance of

$$\boldsymbol{PA} = \boldsymbol{LU}$$

  where $\boldsymbol{P}$ is a permutation matrix

- Furthermore, pivoting (w.r.t. $\max |a_{ij}|$) leads to a backward stable algorithm.

- Once an LU factorization is available, solving a linear system is cheap:

$$\boldsymbol{Ax} = \boldsymbol{LUx} = \boldsymbol{L}(\boldsymbol{Ux}) = \boldsymbol{Ly} = \boldsymbol{b}$$

  or

$$\boldsymbol{PAx} = \boldsymbol{LUx} = \boldsymbol{L}(\boldsymbol{Ux}) = \boldsymbol{Ly} = \boldsymbol{Pb}$$

  - Solve for $\boldsymbol{y}$ using *forward substitution*
  - Solve for $\boldsymbol{x}$ by using *backward substitution* $\boldsymbol{Ux} = \boldsymbol{y}$

Recap: Costs

For forward [backward] substitution at step k there are $\approx k$ $[(n-k)]$ multiplications and subtractions plus a few divisions. The total over all n steps is

$$\sum_{k=1}^{n} k \in \mathcal{O}(n^2)$$

$\rightsquigarrow$ the number of floating-point operations (FLOPs) scales as $\mathcal{O}(n^2)$

For Gaussian elimination, at step k, there are $\approx (n-k)^2$ operations. Thus, the total scales as

$$\sum_{k=1}^{n} (n-k)^2 \in \mathcal{O}(n^3)$$

Summary:

- Directly applying Gaussian elimination (=LU + fwd/bwd) scales as $\mathcal{O}(n^3)$
- Computing LU decomposition scales as $\mathcal{O}(n^3)$
- Forward/backward substitution scales as $\mathcal{O}(n^2)$
- LU + forward/backward scales as $\mathcal{O}(n^3)$ $\rightsquigarrow$ can *reuse LU* for other b

Linear least-squares

Consider non-square matrices $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ and rank(A) $= n$. Then the system

$$Ax = b$$

does, in general, not have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_x \|Ax - b\|^2 = \min_x \Phi(\boldsymbol{x})$$

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$

geometric sketch

$$\bar{x} = \arg\min_x \|Ax - b\|_2^2$$

$$\Updownarrow$$

$$A\bar{x} - b \perp R(A)$$

column space of A

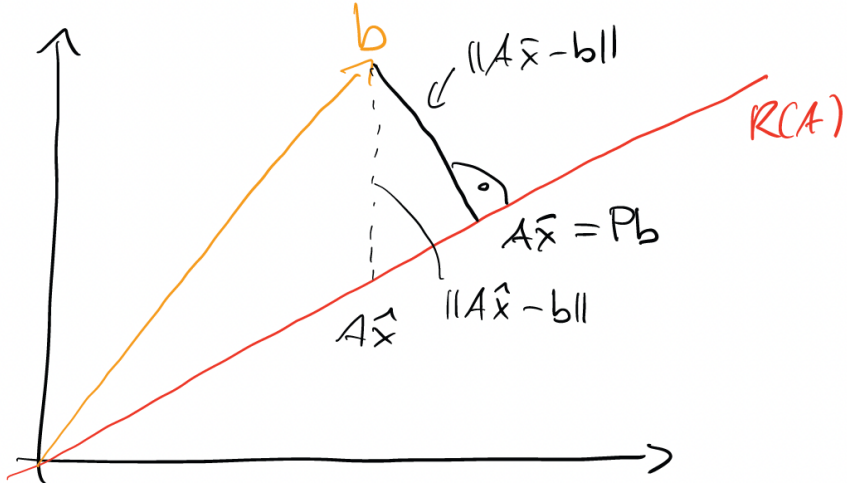
$m \geq n$

$$Ax = b$$

idea: project b onto the range of A

$$Pb$$

$$Ax = Pb$$



Linear least-squares problems

Now for the least-squares problem $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. The relative condition number κ in the Euclidean norm is bounded by

- With respect to perturbations in $\boldsymbol{b}$:

$$\kappa \leq \frac{\kappa_2(A)}{\cos(\theta)}$$

- With respect to perturbations in $\boldsymbol{A}$:

$$\kappa \leq \kappa_2(A) + \kappa_2(A)^2 \tan(\theta)$$

Small residual problems, small angle θ $\cos(\theta) \approx 1$, $\tan(\theta) \approx 0$: behavior similar to linear system.
Large residual problems, large angle θ $\cos(\theta) \ll 1$, $\tan(\theta) \approx 1$: behavior very different from linear system because $\kappa_2(A)^2$ shows up

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \left[Q_1, Q_2\right] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

One would like to avoid the multiplication $A^T A$ (normal equations) and use a suitable factorization of A that avoids solving the normal equation directly:

$$A = QR = \left[Q_1, Q_2\right] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

where $Q \in \mathbb{R}^{m \times m}$ is an orthonormal matrix ($QQ^T = I$), and $R \in \mathbb{R}^{m \times n}$ consists of an upper triangular matrix and a block of zeros.

How can the QR factorization be used to solve the least-squares problem?

$$\begin{aligned} \min_x \|Ax - b\|^2 &= \min_x \|Q^T(Ax - b)\|^2 \qquad &= \min_x \left\| \begin{bmatrix} b_1 - R_1 x \\ b_2 \end{bmatrix} \right\|^2, \\ &= \min_x \|b_1 - R_1 x\|^2 + \|b_2\|^2 \end{aligned}$$

where $Q^T b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$.

Thus, the least squares solution is $x = R^{-1} b_1$ and the residual is $\|b_2\|$.

Problem 2: What is the following algorithm doing when applied to an $m \times n$ matrix A and is it a good idea to use it?

for $j = 1, \ldots, n$

- $v_j = A_{:,j}$
- $R_{1:j-1,j} = Q^T_{:,1:j-1} A_{:,j}$
- $v_j = v_j - Q_{:,1:j-1} R_{1:j-1,j}$
- $R_{jj} = \|v_j\|_2$
- $Q_{:,j} = v_j / R_{jj}$

Problem 2: What is the following algorithm doing when applied to an $m \times n$ matrix A and is it a good idea to use it?

for $j = 1, \ldots, n$

- $v_j = A_{:,j}$
- $R_{1:j-1,j} = Q_{:,1:j-1}^T A_{:,j}$
- $v_j = v_j - Q_{:,1:j-1} R_{1:j-1,j}$
- $R_{jj} = \|v_j\|_2$
- $Q_{:,j} = v_j / R_{jj}$

Computes the reduced QR decomposition with Gram-Schmidt. This is not modified Gram-Schmidt, so the columns in Q can be far from orthogonal.

Instead of directly computing

$$\boldsymbol{v}_j = \boldsymbol{a}_j - (\boldsymbol{q}_1^T \boldsymbol{a}_j)\boldsymbol{q}_1 - (\boldsymbol{q}_2^T \boldsymbol{a}_j)\boldsymbol{q}_2 - \cdots - (\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)\boldsymbol{q}_{j-1}$$

based on $\boldsymbol{a}_j$, the *modified* Gram-Schmidt procedure computes $\boldsymbol{v}_j$ iteratively

$$\begin{aligned}
\boldsymbol{v}_j^{(1)} &= \boldsymbol{a}_j\,, \\
\boldsymbol{v}_j^{(2)} &= \boldsymbol{v}_j^{(1)} - \boldsymbol{q}_1\boldsymbol{q}_1^T \boldsymbol{v}_j^{(1)}\,, && \text{"subtract from } \boldsymbol{v}_j^{(1)} \text{ what is already in } \boldsymbol{q}_1\text{"} \\
\boldsymbol{v}_j^{(3)} &= \boldsymbol{v}_j^{(2)} - \boldsymbol{q}_2\boldsymbol{q}_2^T \boldsymbol{v}_j^{(2)}\,, && \text{"subtract from } \boldsymbol{v}_j^{(2)} \text{ what is already in } \boldsymbol{q}_2\text{"} \\
&\vdots \\
\boldsymbol{v}_j &= \boldsymbol{v}_j^{(j)} = \boldsymbol{v}_j^{(j-1)} - \boldsymbol{q}_{j-1}\boldsymbol{q}_{j-1}^T \boldsymbol{v}_j^{(j-1)}
\end{aligned}$$

Computing a QR factorization with the modified Gram-Schmidt procedure is stabler than with the classical Gram-Schmidt procedure. However, even the modified Gram-Schmidt procedure can lead to vectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$ that are far from orthogonal if the condition number of $\boldsymbol{A}$ is large (see, Golub et al., Matrix Computations, Section 5.2.9)

Eigenproblems

For a matrix $A \in \mathbb{C}^{n \times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Eigenproblems

For a matrix $A \in \mathbb{C}^{n\times n}$ (potentially real), we want to find $\lambda \in \mathbb{C}$ and $\boldsymbol{x} \neq 0$ such that

$$A\boldsymbol{x} = \lambda \boldsymbol{x}.$$

Let $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ be diagonalizable matrix and λ_1 be a simple eigenvalue with

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Let $\boldsymbol{x}_0$ be an initial guess that is not orthogonal to the eigenspace of λ_1, then $\boldsymbol{x}_k$ obtained via the iterations

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \tag{1}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{z}_{k+1}/\|\boldsymbol{z}_{k+1}\|_2 \tag{2}$$

will converge to the normalized eigenvector of $\boldsymbol{A}$ corresponding to λ_1 for $k \to \infty$.

This process is called the power method. How did we proof convergence?

Problem 3: Computing eigenvectors

Problem 3: Starting with $p_0 = q_0 = 1$, define the iteration

$$\begin{aligned} p_{n+1} &= p_n + q_n \\ q_{n+1} &= p_{n+1} + p_n \end{aligned}$$

for $n = 0, 1, 2, \ldots$. The ratio q_n/p_n converges to $\sqrt{2}$ as $n \to \infty$.

Prove convergence of this algorithm using the power method. I.e., write the problem as an eigenvalue problem and show that the power method for this eigenvalue problem converges to an eigenvector and deduce from this eigenvector that $q_n/p_n \to \sqrt{2}$.
(*Hint:* $\det(A) = ad - bc$ for a 2×2 matrix with $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.)

$q_0 = p_0 = 1$

$$p_{n+1} = p_n + q_n$$
$$q_{n+1} = p_{n+1} + p_n$$

Write as matrix

$$\begin{bmatrix} p_{n+1} \\ q_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} p_n \\ q_n \end{bmatrix}$$

Compute eigenvalues of A

$$\det(A - \lambda I) = \det\begin{pmatrix} 1-\lambda & 1 \\ 2 & 1-\lambda \end{pmatrix}$$

$$= (1-\lambda)(1-\lambda) - 1 \cdot 2$$
$$= \lambda^2 - 2\lambda - 1$$

$$\lambda_{1/2} = \dots = 1 \pm \sqrt{2}$$

$\Rightarrow$ dominant EV: $|1+\sqrt{2}| > |1-\sqrt{2}|$

Compute EV for $1+\sqrt{2} = \lambda_1$

$$0 = (A - \lambda_1 I)\bar{v}_1 = \begin{bmatrix} 1-(1+\sqrt{2}) & 1 \\ 2 & 1-(1+\sqrt{2}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$= \begin{bmatrix} -\sqrt{2} & 1 \\ 2 & -\sqrt{2} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$v_1 = \frac{1}{\sqrt{2}} v_2 \quad (*)$$

obtain $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1 \end{bmatrix}$ as an EV

starting point $\begin{bmatrix} p_0 \\ q_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, <u>not</u> orthogonal to $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

power method converges to an eigenvector

For all eigenvectors have $(*)$

$$\frac{q_\infty}{p_\infty} = \frac{v_2}{v_1} = \frac{v_2}{\frac{1}{\sqrt{2}} v_2} = \sqrt{2}$$

Iterative methods for linear systems

Iterative solution of linear systems

Target problems: very large ($n = 10^5, 10^6, \ldots$), A is usually sparse and has specific properties.

To solve

$$A\boldsymbol{x} = \boldsymbol{b}$$

we construct a sequence

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

of iterates that converges fast to the solution $\boldsymbol{x}$, where $\boldsymbol{x}_{k+1}$ can be cheaply computed from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ (e.g., one matrix-vector multiplication).

Thought experiment: If we can compute one iteration with cost $\mathcal{O}(n)$ (e.g., one matrix-vector multiplication with a sparse matrix) and need a constant $\mathcal{O}(1)$ number of iterations to reach desired precision, then we solve $Ax = b$ with costs $\mathcal{O}(n)$. Intuitively, we cannot do better than that because we solve for n quantities and thus need to touch each at least once.

Let Q be invertible, then

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow Q^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - Q^{-1}A)\boldsymbol{x} + Q^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow G\boldsymbol{x} + \boldsymbol{c} = \boldsymbol{x} \end{aligned}$$

Leads to fixed-point iteration

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c}$$

and with $\boldsymbol{G}$ invertible obtain that $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$ is a stationary point

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (need it to be invertible)?

Extreme cases for selecting $\boldsymbol{Q}$

What are two extreme cases for selecting $\boldsymbol{Q}$ (need it to be invertible)?

Choose $Q = A^{-1}$, then our iteration becomes

$$\begin{aligned} A\boldsymbol{x} = \boldsymbol{b} &\Leftrightarrow A^{-1}(\boldsymbol{b} - A\boldsymbol{x}) = 0 \\ &\Leftrightarrow (I - A^{-1}A)\boldsymbol{x} + A^{-1}\boldsymbol{b} = \boldsymbol{x} \\ &\Leftrightarrow \boldsymbol{0} + \boldsymbol{x} = \boldsymbol{x} \end{aligned}$$

and we are done in just a single step

$$x_{k+1} = x$$

Thus, if we "know the solution" (in form of having the inverse A^{-1}) then no further work is needed here because we already did all the work when finding A^{-1}

The other extreme is setting $Q = I$, this leads to the Richardson method

$$\boldsymbol{x}_{k+1} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{b}$$

We have invested zero costs in finding Q and so we intuitively expect that $Q = I$ will require high costs in terms of number of iterations to converge in general, if it converges at all

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

$$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

  Jacobi uses $Q = D$ and Gauss-Seidel uses $Q = L + D$, which both can be computed quickly from A.
- **What property of A critically influences how quickly these relaxation methods converge?**

Problem 3: Answer the following questions with 1-2 sentence explanations:

- **What is the problem with the Richardson method and what can we do about it?**

  Converges only for limited set of matrices. Use linear combination between new and previous iterate:

  $$\boldsymbol{x}_{k+1} = \omega \underbrace{(G\boldsymbol{x}_k + c)}_{\boldsymbol{x}'_{k+1}} + (1-\omega)\boldsymbol{x}_k = G_\omega \boldsymbol{x}_k + \omega c,$$

  where $\omega > 0$ is a damping/relaxation parameter. Goal is to choose ω such that $\rho(G_\omega)$ is minimal.
- **What are other reasonable choices for Q?**

  Jacobi uses $Q = D$ and Gauss-Seidel uses $Q = L + D$, which both can be computed quickly from A.
- **What property of A critically influences how quickly these relaxation methods converge?**The spectral radius $\rho(A)$ of A

In the following A is symmetric positive definite.

Formulate solving $Ax = b$ as an optimization problem: Define

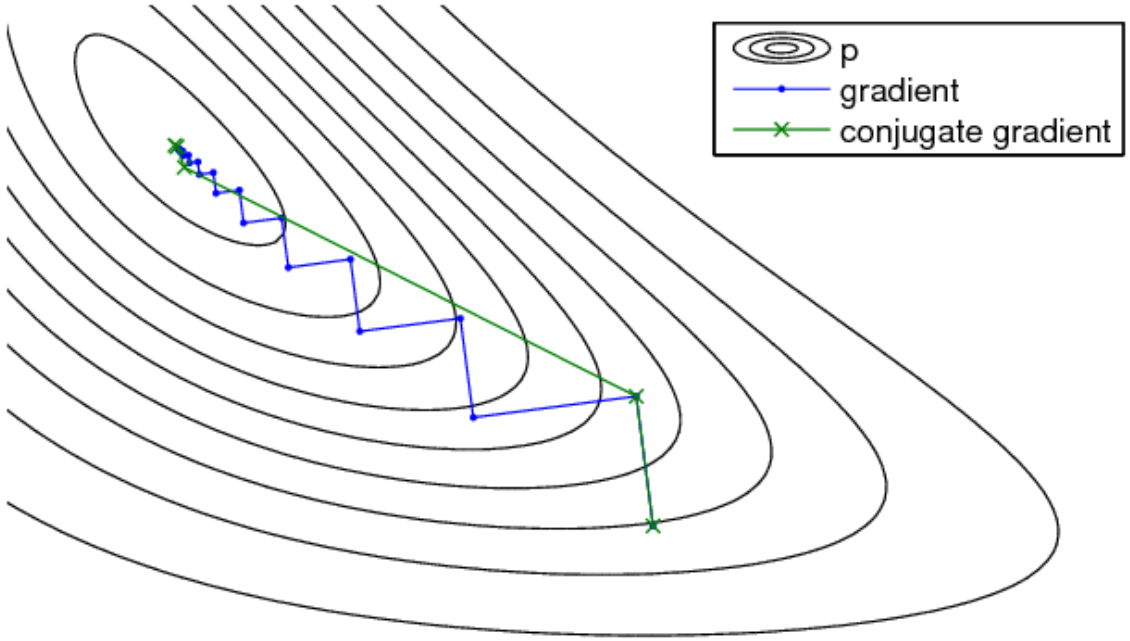
$$f(x) = \frac{1}{2}x^T Ax - b^T x\,,$$

and minimize

$$\min_{x \in \mathbb{R}^n} f(x)$$

Because A is positive definite, we have $f(x) = 0 \iff Ax = b$. It is sufficient to look at the gradient

$$\nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b = -r(x) = 0 \iff Ax = b$$



[Figure: Kuusela et al., 2009]

The convergence behavior of steepest descent in this context can be poor: we eventually get arbitrarily close to the minimum but we can always destroy something of the already achieved when applying the update $\rightsquigarrow$ can we find better search directions?

Conjugate gradient method

- All methods so far (relaxation, steepest descent) use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

Conjugate gradient method

- All methods so far (relaxation, steepest descent) use information about x_{k-1} to get x_k. All information about earlier iterations is ignored.

- The conjugate gradient (CG) method is a variation of steepest descent that *has a memory*.

- Let $p_1, \ldots, p_k$ be the directions up to step k, then CG uses the space

$$x_0 + \operatorname{span}\{p_1, \ldots, p_k\}, \qquad x_0 \text{ starting point}$$

  to find the next iterate x_k and thus

$$x_k = x_0 + \sum_{i=1}^{k} \alpha_i p_i$$

- (Recall that steepest descent uses only the search direction $p_k = r_{k-1} = -\nabla f(x_{k-1})$ to find the iterate x_k)

We want the following

a The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \mathrm{span}(p_1, \ldots, p_k)} f(x)$$

c The step x_k can be calculated easily from x_{k-1}

We then worked hard to derive the CG algorithm based on these three conditions

What was a critical step in CG?

We want the following

a The search directions $p_1, \ldots, p_k$ should be linearly independent ("we don't destroy what we have achieved")

b We have ("we do the best we can at each step")

$$f(x_k) = \min_{x \in x_0 + \mathrm{span}(p_1, \ldots, p_k)} f(x)$$

c The step x_k can be calculated easily from x_{k-1}

We then worked hard to derive the CG algorithm based on these three conditions

What was a critical step in CG? $\rightsquigarrow$ Gram-Schmidt orthogonalization to find the search direction

It can be shown that for $k \geq 1$ and $e_j \neq 0, j < k$ it holds

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \|e_0\|_A$$

for spd matrices A. $\rightsquigarrow$ Trefethen & Bau

For steepest descent, if A is spd, we obtained

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y \rangle_A = x^T A y$ and $\| \cdot \|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

We keep finding we are limited by the condition number of A. What can be done about it (in Numerical Methods II)?

It can be shown that for $k \geq 1$ and $e_j \neq 0, j < k$ it holds

$$\|e_k\|_A \leq 2 \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \|e_0\|_A$$

for spd matrices A. $\rightsquigarrow$ Trefethen & Bau

For steepest descent, if A is spd, we obtained

$$\|x^* - x_k\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^k \|x^* - x_0\|_A \,,$$

where $\langle x, y \rangle_A = x^T A y$ and $\| \cdot \|_A = \sqrt{\langle \cdot, \cdot \rangle_A}$.

We keep finding we are limited by the condition number of A. What can be done about it (in Numerical Methods II)? Multigrid, multi-level

Solving systems of nonlinear equations

Solving nonlinear equations ("root finding")

We want to solve the nonlinear equation

$$f(x) = 0, \quad x \in \mathbb{R}.$$

We could also have $n < \infty$ equations in n unknowns with $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\boldsymbol{x}) = \mathbf{0}$$

In general, we will need an iterative approach that constructs $x_1, x_2, x_3, \ldots$ such that

$$\lim_{k \to \infty} x_k = x^* ,$$

with $f(x^*) = 0$.

Reformulation as fixed point method so that x^* is fixed point

$$x^* = \Phi(x^*)$$

Let $F : \mathbb{R}^n \to \mathbb{R}^n$, $n \geq 1$ and solve

$$F(\boldsymbol{x}) = 0.$$

Truncated Taylor expansion of F about starting point $\boldsymbol{x}^0$:

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}^0) + F'(\boldsymbol{x}^0)(\boldsymbol{x} - \boldsymbol{x}^0).$$

Hence:

$$\boldsymbol{x}^1 = \boldsymbol{x}^0 - F'(\boldsymbol{x}^0)^{-1} F(\boldsymbol{x}^0)$$

Newton iteration: Start with $\boldsymbol{x}^0 \in \mathbb{R}^n$, and for $k = 0, 1, \ldots$ compute

$$F'(\boldsymbol{x}^k)\Delta\boldsymbol{x}^k = -F(\boldsymbol{x}^k), \quad \boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \Delta\boldsymbol{x}^k$$

Requires that $F'(\boldsymbol{x}^k) \in \mathbb{R}^{n \times n}$ is invertible.

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

  No, can converge to local optima

- You want to solve $F(x) = 0$ but you have access only to a scrambled F through $(G_k \circ F) = G_k(F(x))$, where k is the Newton iteration, and thus you solve a different problem $G_k(F(x)) = 0$ in each iteration. Derive another condition on G_k than G_k being the identity so that Newton converges to x^* with $F(x^*) = 0$ in a neighborhood of x^*. Provide a proof.

$$G_k \circ F = A_k F \qquad , \text{ regular } A_k$$

$$y_{k+1} = y_k - \left[(G_k \circ F)' \right]^{-1} (y_k) \, (G_k \circ F)(y_k)$$

$$= y_k - \underbrace{\left[(A_k F)' \right]^{-1} (y_k)}_{A_k F'} \, A_k F(y_k)$$

$$= y_k - F'(y_k)^{-1} \underbrace{A_k^{-1} A_k}_{} F(y_k)$$

$$= y_k - F'(y_k)^{-1} F(y_k)$$

Problem 4: Answer the following questions with explanations of 1-2 sentences and/or proofs

- Describe two key requirements for Newton to converge quadratically.

  Need $F'(x)$ invertible and starting point x_0 close to solution x^*

- You have implemented Newton in Matlab and tried it on two starting points x_0 and y_0 for your problem. Newton converges for both but to a different solution. Does this mean there something wrong with your implementation?

  No, can converge to local optima

- You want to solve $F(x) = 0$ but you have access only to a scrambled F through $(G_k \circ F) = G_k(F(x))$, where k is the Newton iteration, and thus you solve a different problem $G_k(F(x)) = 0$ in each iteration. Derive another condition on G_k than G_k being the identity so that Newton converges to x^* with $F(x^*) = 0$ in a neighborhood of x^*. Provide a proof.

  Newton is affine invariant. So as long as G_k is linear and the corresponding matrix is regular, it does not influence the Newton iterations. Proof is the same as for the affine invariance that we did in class.

Interpolation

Polynomial interpolation

Consider $n+1$ pairs $(x_i, y_i), i = 0, \ldots, n$ of a function f with

$$y_i = f(x_i)$$

Let now $\mathbb{P}_n$ be the set of all polynomials up to degree n over $\mathbb{R}$ so that we have for all $P \in \mathbb{P}_n$

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \,, \qquad a_n, \ldots, a_0 \in \mathbb{R}$$

We would like to find a $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

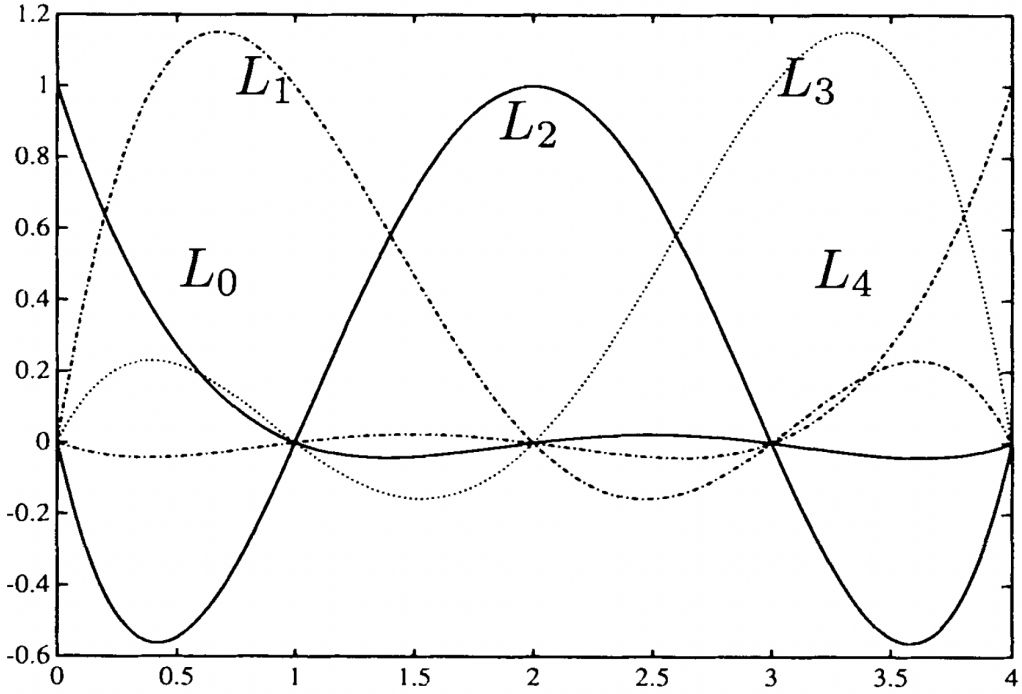
Theorem: Given $n+1$ points (x_i, y_i) with pairwise distinct $x_0, \ldots, x_n$, there exists a unique polynomial $P \in \mathbb{P}_n$ such that

$$P(x_i) = y_i \,, \qquad i = 0, \ldots, n$$

Lagrange basis

The Lagrange polynomials $L_0, \ldots, L_n \in \mathbb{P}_n$ are uniquely defined for distinct $x_0, \ldots, x_n$

$$L_i(x_j) = \delta_{ij}\,, \qquad L_i \in \mathbb{P}_n\,.$$



Lagrange polynomials up to order $n = 4$ for equidistant $x_0, \ldots, x_4$. [Figure: Deuflhard]

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free

The corresponding explicit formula is

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0, \ldots, n$$

What are the coefficients $a_n, \ldots, a_0$ so that $P(x_i) = y_i$ for $i = 0, \ldots, n$?

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

because

$$P(x_j) = \sum_{i=0}^{n} y_i L_i(x_j) = \sum_{i=0}^{n} y_i \delta_{ij} = y_j$$

If we have the basis $L_0, \ldots, L_n$, we obtain the polynomial P for free but the cost of evaluating the polynomial is too high for practical computations

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

Polynomial interpolation in Newton basis

The Newton basis $\omega_0, \ldots, \omega_n$ is given by

$$\omega_i(x) := \prod_{j=0}^{i-1}(x - x_j) \in \mathbb{P}_i.$$

Would like to find coefficients $c_0, c_1, \ldots, c_n$ of interpolating polynomial in Newton basis

$$P_f(x|x_0, \ldots, x_n) = c_0\omega_0(x) + c_1\omega_1(x) + \cdots + c_n\omega_n(x)$$

The leading coefficient a_n of the interpolation polynomial

$$P_f(x|x_0, \ldots, x_n) = a_n x^n + \cdots + a_0$$

is called the *n-th divided difference*, $[x_0, \ldots, x_n]f := a_n$.

The divided differences are the coefficients $c_0, \ldots, c_n$: The interpolation polynomial $P_f(\cdot|x_0, \ldots, x_n)$ for $x_0 \leq x_1 \leq \cdots \leq x_n$ is given by

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i] f \, \omega_i(x).$$

The following recurrence relation holds for $x_i \neq x_j$:

$$[x_0, \ldots, x_n] f = \frac{([x_0, \ldots, \hat{x}_i, \ldots, x_n] f - [x_0, \ldots, \hat{x}_j, \ldots, x_n] f)}{x_j - x_i}$$

which is helpful to compute the divided differences

Problem 5: Compute the polynomial p with $p(0) = 2, p(1) = 3, p(3) = 0$ in the Newton basis

x_i		
0	$[x_0]p = 2$	P_1
1	$[x_1]p = 3$	
3	$[x_0]p = 0$	

$$[x_0, x_1]p = \frac{[x_0]p - [x_1]p}{x_0 - x_1} = \frac{2-3}{0-1} = 1 \quad P_2$$

$$P_3(x) = 2 + x - \frac{5}{6}x(x-1)$$

Newton-Cotes formulas for quadrature

Given fixed nodes $t_0, \ldots, t_n$, use polynomial approximation

$$\hat{f} = P_f(t|t_0, \ldots, t_n) = \sum_{i=0}^{n} f(t_i) L_{in}(t)$$

with Lagrange polynomials $L_{0n}, \ldots, L_{nn}$

Thus:

$$\hat{I}(f) = (b-a) \sum_{i=0}^{n} \lambda_{in} f(t_i),$$

where $\lambda_{in} = \frac{1}{b-a} \int_a^b L_{in}(t)\, dt \rightsquigarrow$ weights are unique
Quadrature formulas defined in this way are exact for polynomials $P \in \mathbb{P}_n$ of degree less than or equal to n

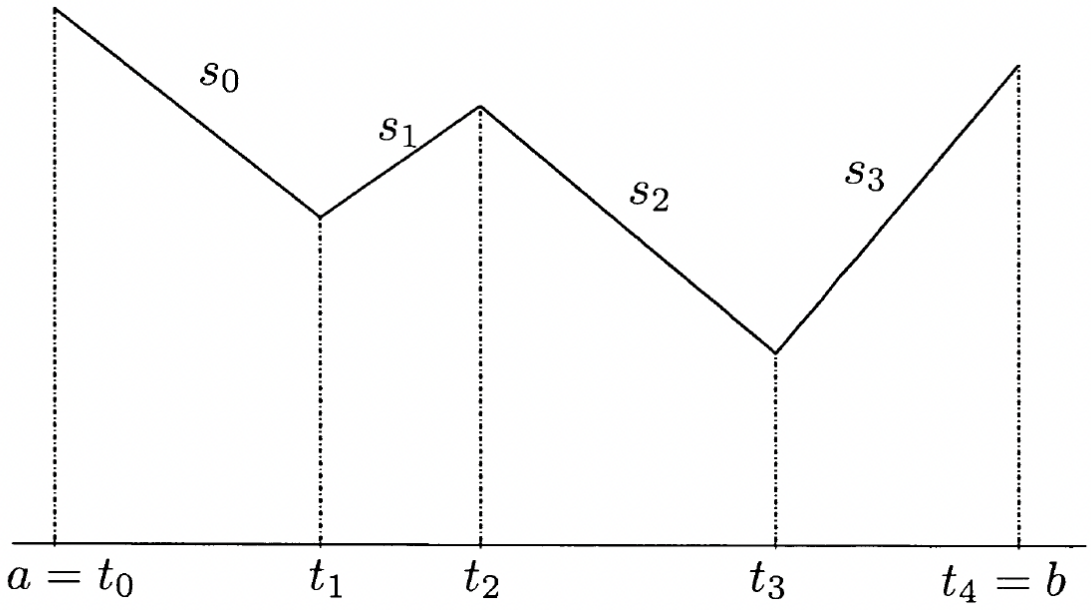
$$\hat{I}(P) = I(P_n(P)) = I(P)\,, \qquad \text{for all } P \in \mathbb{P}_n$$

Trapezoidal sums

To avoid poorly conditioned problems, let us split the integration interval $[a, b]$ into n sub-intervals $[t_{i-1}, t_i]$, $i = 1, \ldots, n$. Then consider the rule

$$\hat{I}(f) = \sum_{i=1}^{n} \hat{I}_{t_{i-1}}^{t_i}(f),$$

where $\hat{I}_{t_{i-1}}^{t_i}$ is a quadrature formula on the interval $[t_{i-1}, t_i]$.



We have seen already the trapezoidal sum with $h = (b-a)/n$

$$T(h) = \sum_{i=1}^{n} T_i = h\left(\frac{1}{2}(f(a)+f(b)) + \sum_{i=1}^{n-1} f(a+ih)\right)$$

that has error

$$T(h) - \int_a^b f = \frac{(b-a)h^2}{12} f''(\tau)\,, \qquad \tau \in [a,b]$$

$\rightsquigarrow$ we can increase n (and thus decrease h) to reduce the error without increasing the degree of the underlying polynomial

Conclusions

- This was a very selective review of the topics that we covered!
- Be prepared to answer True/False questions to show your basic understand of topics that we discussed.
- Be prepared to answer questions that require 1-2 sentence explanations or short proofs.
- You should be able to do quick calculations such as, e.g., approximating an integral with the trapezoidal rule and computing the LU decomposition of small matrices and computing eigenvalues of a 2×2 matrix etc.
- Recapping basic linear algebra topics will be beneficial (e.g., computing determinant of a 2×2 matrix)