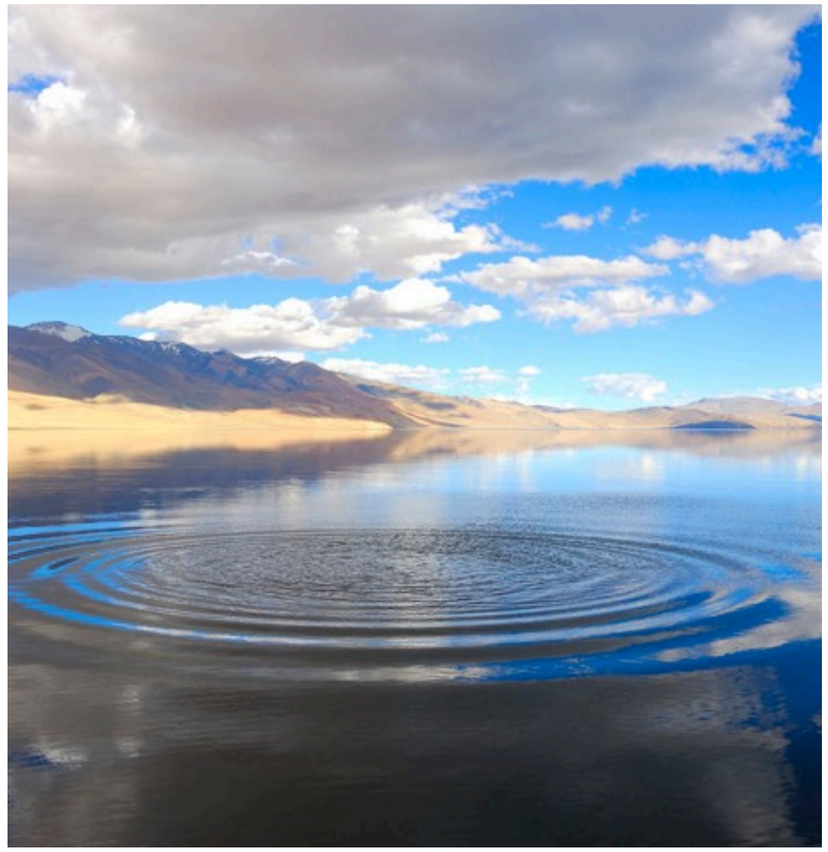
Lecture 21:

Visual & Numerical Linear Algebra

Computer Graphics
CMU 15-462/15-662, Fall 2015

Last time: Partial Differential Equations

- Physical systems as PDEs
- Discretize time *and* space
- Lots of beautiful phenomena
  - fluid, liquid, smoke, fire, ...
  - elasticity, hair, shells, cloth, ...
- Model PDEs were *linear* equations
- Express function at next time step as linear combination of values
- Led to large system of simultaneous *linear equations*
  - solved via iterative *Jacobi method* (slow)
  - TODAY: fast linear solvers
  - pervasive throughout graphics (physics, geometry, imaging...)



Review: Linear Systems of Equations

- Q: What is a linear equation?
- A: E.g., $ax + by = c$
- Q: What is a linear system of equations?
- A: Set of linear equations that must be satisfied simultaneously.
- Q: How you solve a linear system of equations? E.g.,

linear system

$$\begin{aligned} 3x + 4y &= 11 \\ 5x + 6y &= 17 \end{aligned}$$

isolate x	$x = (11 - 4y)/3$
substitute x	$5(11 - 4y)/3 + 6y = 17$
solve for y	$y = 2$
substitute y	$5x + 6(2) = 17$
solve for x	$x = 1$

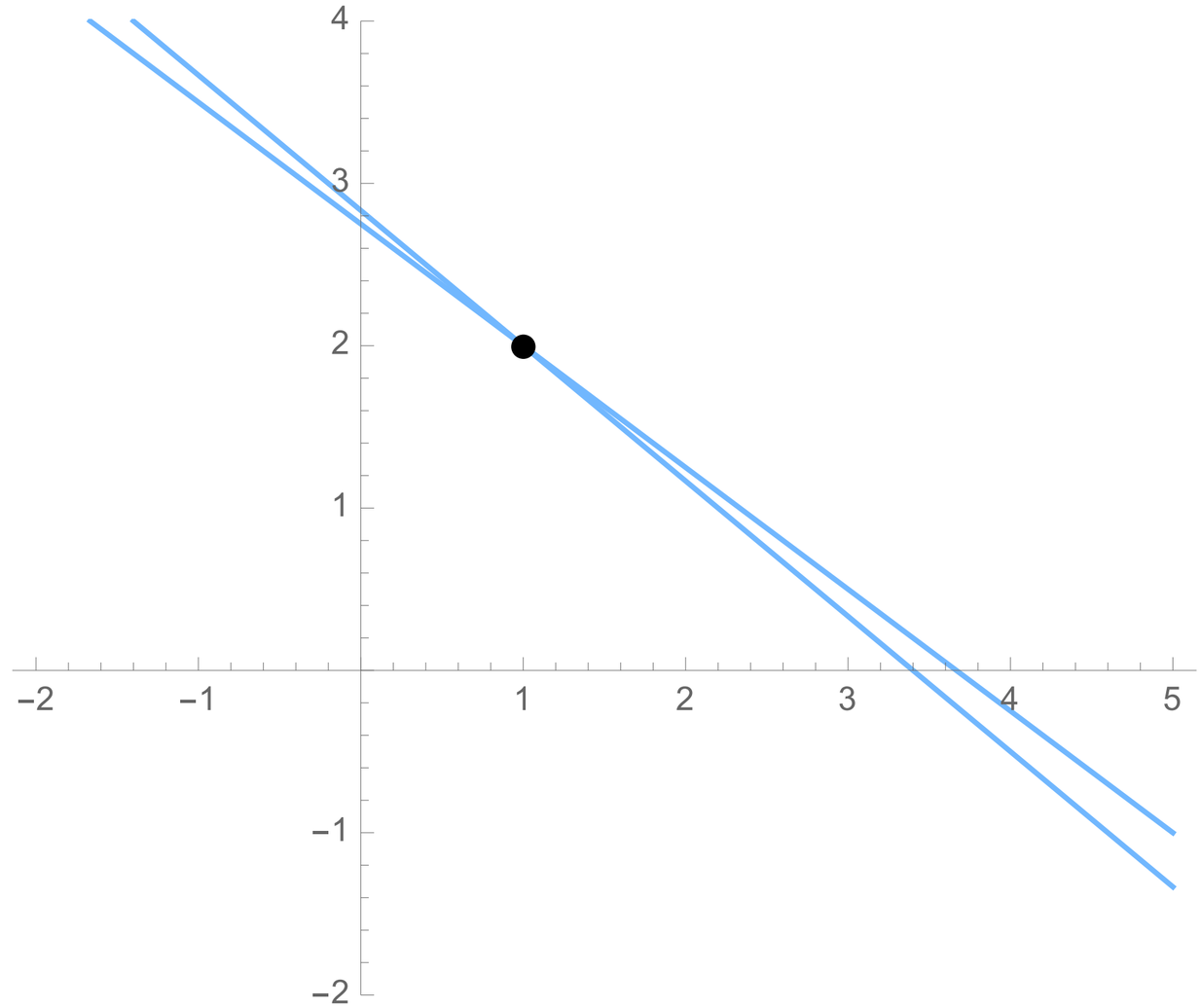
Today we will take this basic algorithm *much* further!

Review: Visualizing Linear Equations

- **Q: What does our linear system mean *visually?***
- **A: In this case, looking for intersection of lines:**

linear system

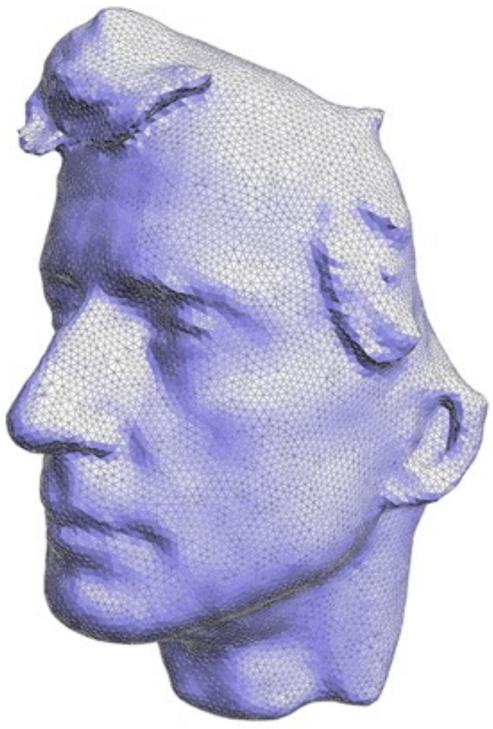
$$\begin{aligned} 3x + 4y &= 11 \\ 5x + 6y &= 17 \end{aligned}$$

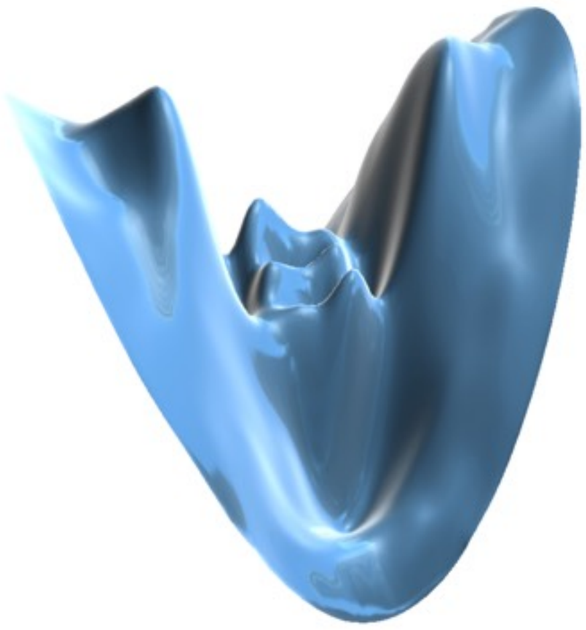


(Will also take this idea much further…)

Linear Systems in Computer Graphics

- Phenomena in CG increasingly modeled by *linear systems*
- A few examples that we've already seen:
  - GEOMETRY - Laplacian smoothing (assignment #2)
  - RENDERING - finite element radiosity (Lecture 15)
  - ANIMATION - PDE-based animation (Lecture 20)
  - IMAGE PROCESSING - tone mapping [later!]





View from 10,000 ft. up

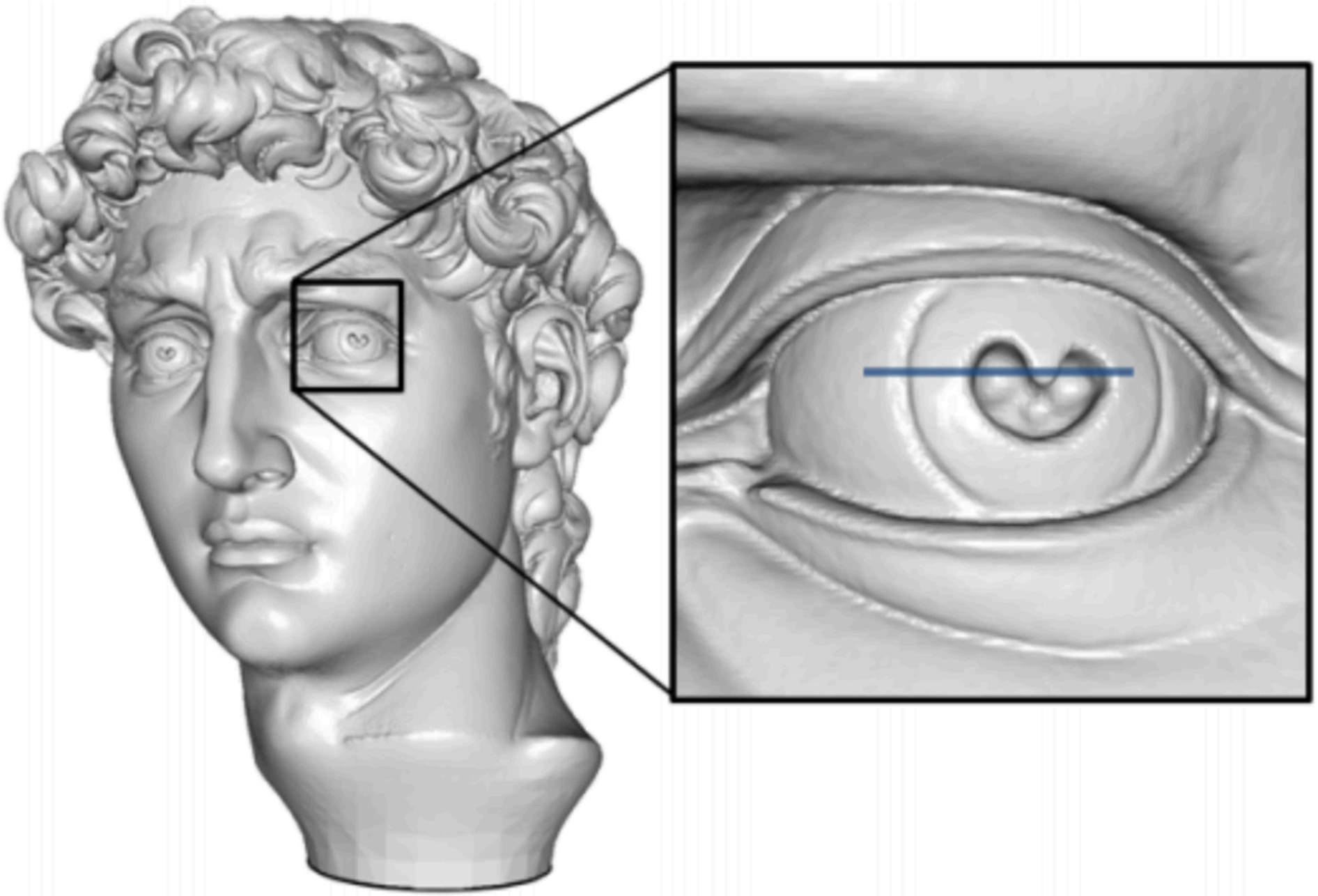
FFT : signal processing

Poisson equation : computer graphics*

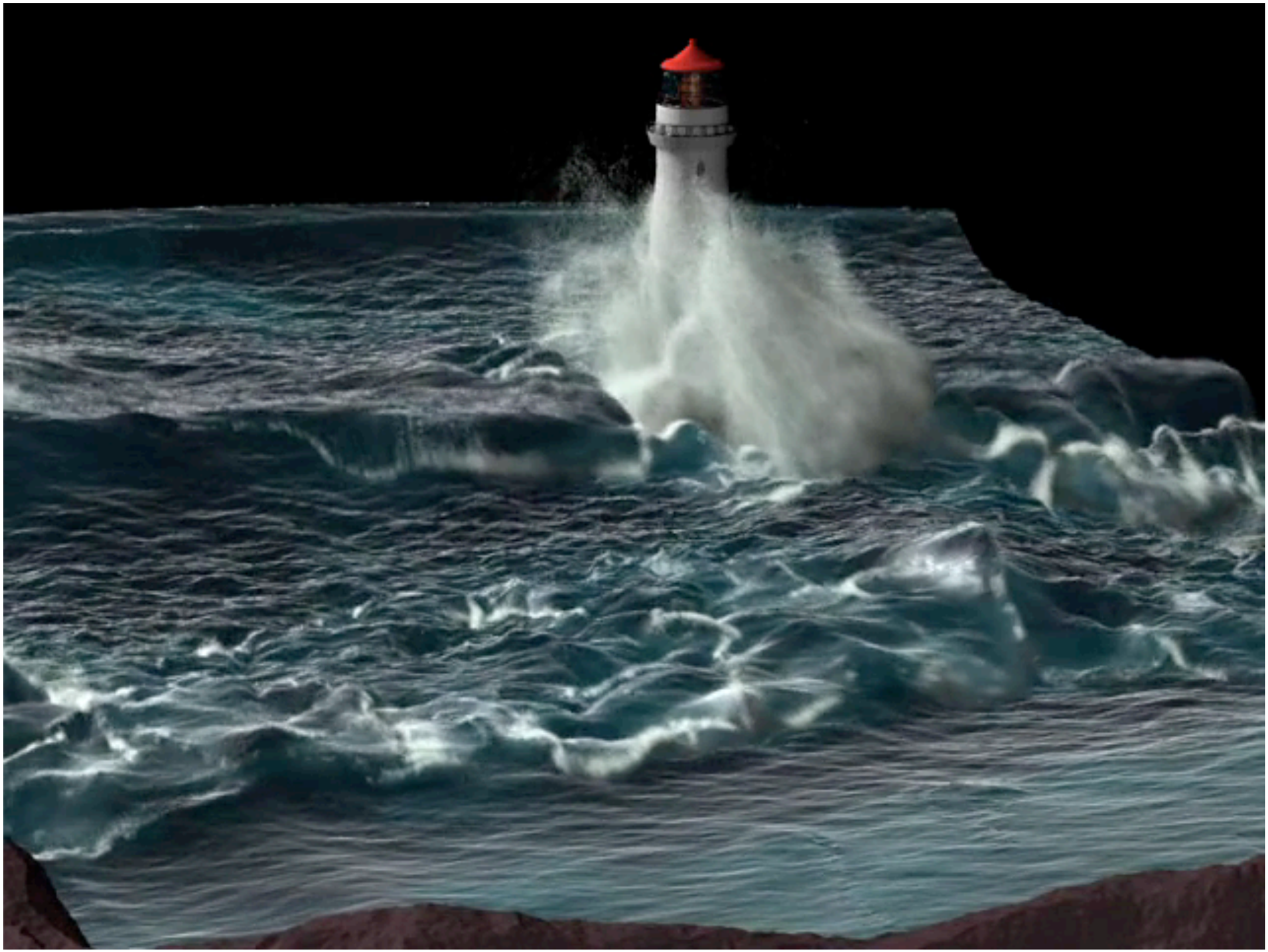
(linear!)

*Fairly recent development. ~15 years ago film studios, games unwilling to use algorithms that depend on linear solves. These days, much more open-minded...

Solving BIG linear systems - GEOMETRY

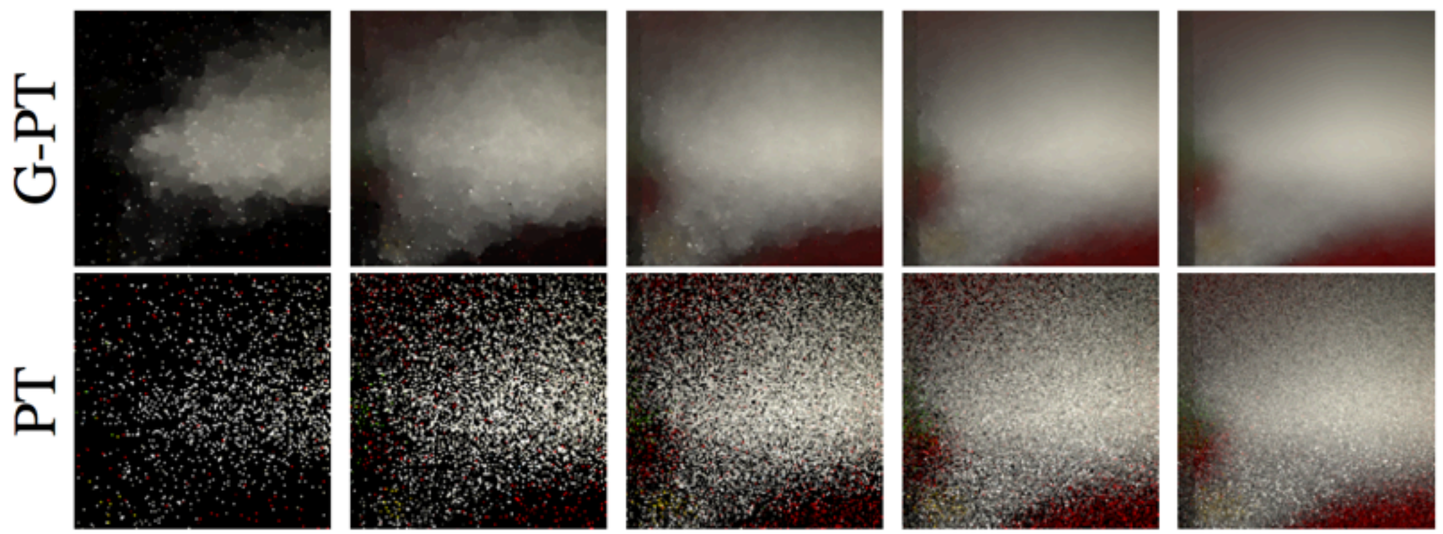


Solving BIG linear systems - ANIMATION



Solving BIG linear systems - IMAGE PROCESSING

Solving BIG linear systems - RENDERING



What kinds of problems do we encounter?

LINEAR SYSTEMS

$$Ax = b$$

ray intersection, splines, (un)projection, ...

linear (or linearized) PARTIAL DIFFERENTIAL EQUATIONS

$$\dot{u} = Lu$$

cloth, fluids, mesh smoothing, light transport, ...

LEAST SQUARES PROBLEMS

$$\min_x ||Ax - b||$$

constraint-based dynamics, elastic analogies, ...

EIGENVALUE PROBLEMS

$$Ax = \lambda x$$

modal analysis, parameterization, clustering, ...

All boil down to solving *linear systems.*

Most Important Algorithms of 20th Century

- **According to SIAM:**
  - **Monte Carlo method (1946)***
  - **simplex method (1947)**
  - **Krylov subspace methods (1950)***
  - **matrix decomposition (1951)***
  - **optimizing compilers (1957)**
  - **QR algorithm (1959)***
  - **quicksort (1962)**
  - **fast Fourier transform (1965)***
  - **integer relation detection (1977)**
  - **fast multipole algorithm (1987)***

– numerical linear algebra

– important in graphics

* – discussed in this course

Aside: Why *Linear?*

- **Because everything else is (WAY) too hard!**
- **Even just going from linear to cubic equations becomes NP-hard**
- **Simple reduction from 3-SAT**

(three-term clauses in conjunctive normal form)

$$(a \vee b \vee \neg c) \wedge (\neg b \vee c \vee \neg d)$$

System of polynomials with degree at most 3:

$$a(1-a) = 0$$
$$b(1-b) = 0$$
$$c(1-c) = 0$$
$$d(1-d) = 0$$

(solution: zero or one)

$$(1-a)(1-b)c = 0$$
$$b(1-c)d = 0$$

(negated variables satisfy original 3-SAT problem)

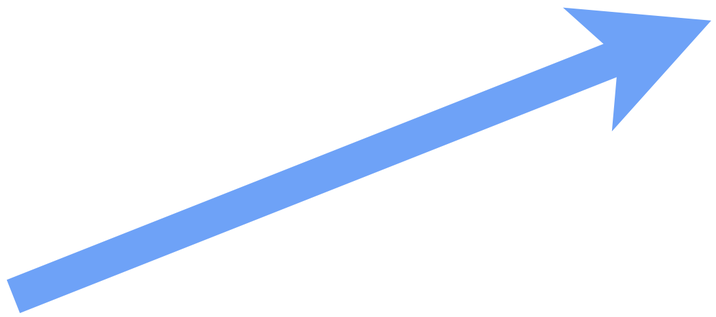
Also true for quadratic equations; slightly trickier to show…

- **Usually (try to) solve polynomial equations by linearizing, using good initial guess**
- **Also other methods like Gröbner bases but (no big surprise) cost can grow exponentially.**

So what is numerical linear algebra?

Review: Vectors

- Q: What is a vector? (intuitively! geometrically! *no formalism!*)
- A: A vector is a little arrow:



- Really just two pieces of data:
  1. direction
  2. magnitude
- You might also say "basepoint", but a "vector with a basepoint" is really just a tuple (point, vector).
- We will not use this idea, so you can forget about it now.

Seriously, just forget about it.

Review: Vector Spaces

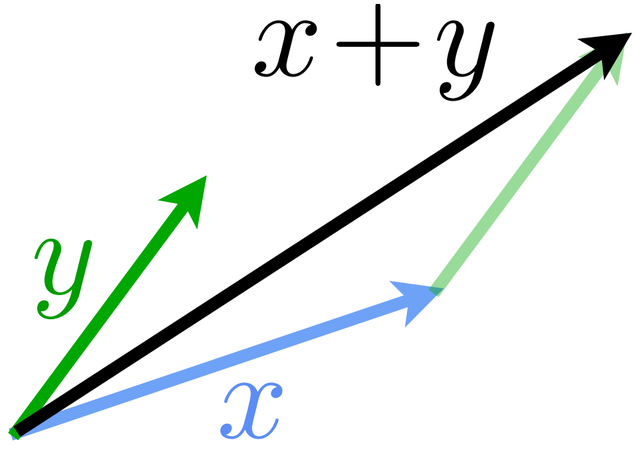
- Ok, so what is a vector then?
- A vector space is a set V with operations +, • such that for all vectors x,y, z in V, and all numbers a,b...
- You can add vectors: $x + y \in V$
  - order doesn't matter: $x + y = y + x$
  - grouping doesn't matter: $(x + y) + z = x + (y + z)$
  - there's a "do nothing" vector: $x + 0 = x$
  - and you can always go back: $x + (-x) = 0$
- You can scale vectors in the way you'd expect:

$$(ab)x = a(bx) \qquad a(x + y) = ax + ay$$

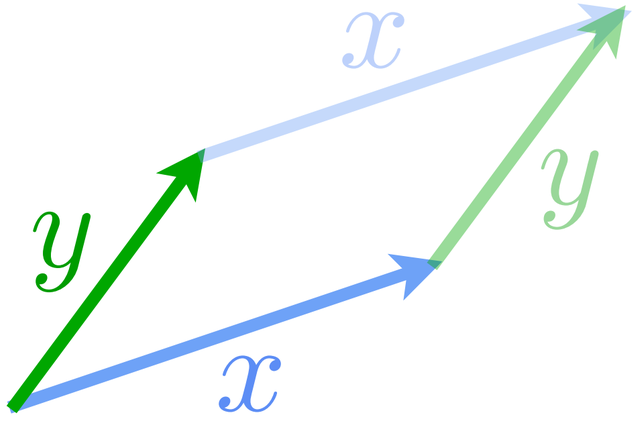
$$1x = x \qquad (a + b)x = ax + bx$$

Familiar Example: Vectors in R^n

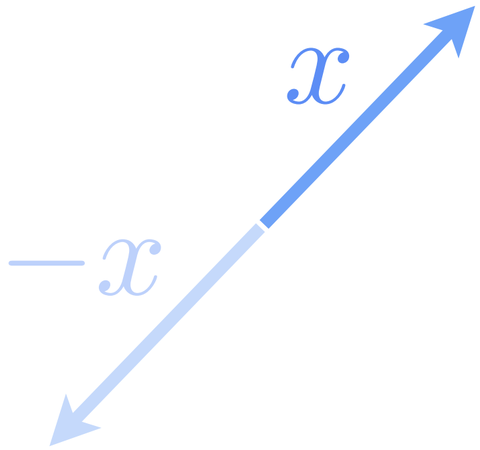
- **May be easier to understand visually:**



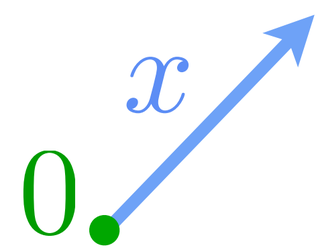
$$x + y \in V$$



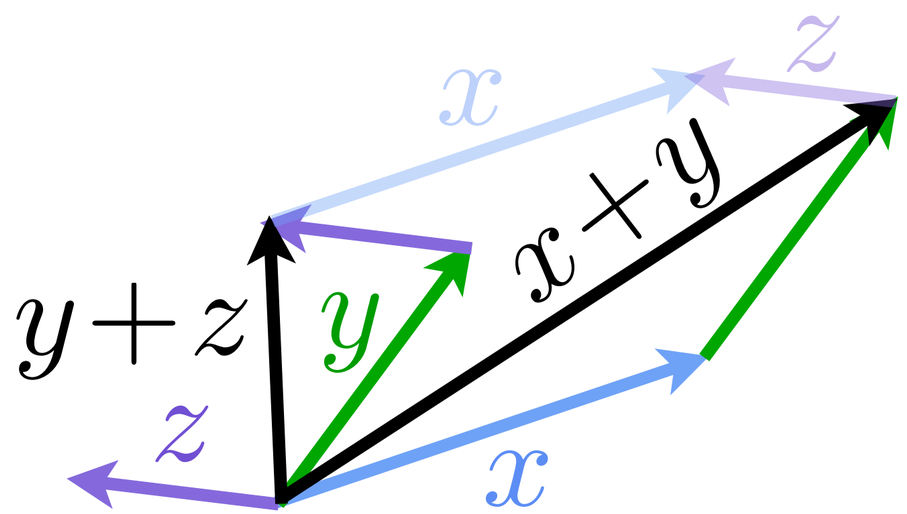
$$x + y = y + x$$



$$x + (-x) = 0$$



$$x + 0 = x$$



$$(x + y) + z = x + (y + z)$$

Test Case: Polynomials

- Reminder—what's a polynomial?

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

- Q: Do polynomials follow all the rules we had for vectors?
  - Is the sum of two polynomials a polynomial?
  - Does the order of polynomial addition matter?
  - Is there a zero polynomial?
  - ...
- A: YES!
- Q: Ok, but come on, polynomials aren't vectors. They don't look like little arrows!
- A: That wasn't a question. And polynomials are definitely vectors.

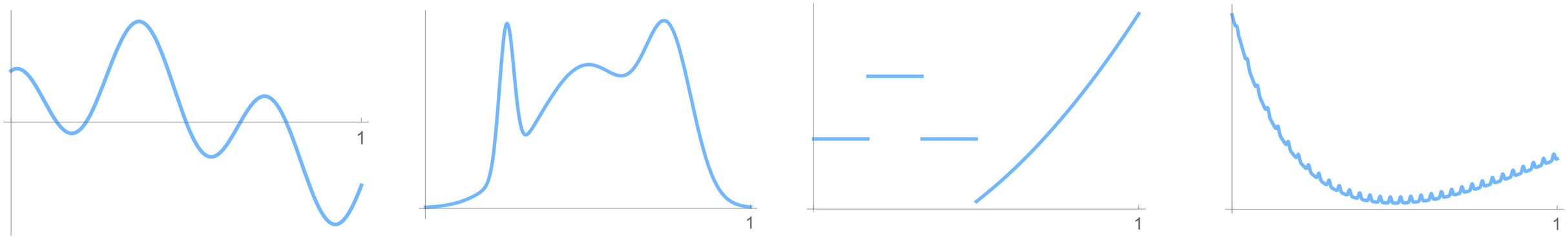
vector of coefficients

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

*Bonus: where in this class have we seen different bases for polynomials?

Test Case: Functions on [0,1]

- Ok, so polynomials were vectors—but that's pretty clear, because they're defined via a *vector* of coefficients.
- What about completely *arbitrary* functions f: [0,1] → R?
  - Is the sum of two functions a function?
  - What about all the other properties?
- Q: Are real functions on [0,1] vectors?
- A: YES!
- Q: Can we express them via a finite list of coefficients?
- A: NO! Nonetheless, these functions are still *vectors*.



A Vector is Not a Little Arrow

- Just like a pixel is not a little square, a vector is not a little arrow
- Our arrow drawings are a useful cartoon, but...
- Many (most?) important examples of "vectors" in graphics actually come from signals or *functions*. E.g.:
  - PDEs in physically-based animation $\dot{u} = F(t, u, \dot{u}, u', \ldots)$
  - incident illumination in rendering $\int_\Omega f_r(\omega) L_i(\omega) n \cdot \omega \; d\omega$
  - the surface itself in geometry processing $f : M \to \mathbb{R}^3$
  - the image itself in image processing $I : [0,1]^2 \to \mathbb{R}$
- These objects all start out life as infinite-dimensional *functions*
- We can work with them on a computer only by approximating/ discretizing them as (BIG) finite-dimensional vectors
- That's why we need to solve BIG numerical linear algebra problems.

Ok… so what's numerical linear algebra?

Numerical Linear Algebra

- **Linear algebra: *vector spaces* and *linear maps* between them**
- **Q: What's a linear map?**
- **A: Any map that preserves all vector space operations:**

$$f(ax + by) = af(x) + bf(y)$$

- **Numerical linear algebra: use computation to answer questions about vector spaces and linear maps.**
- **Prototypical example:**

$$\text{Solve } Ax = b \text{ for x.}$$

"given a linear map A between vector spaces and a point b, which vector x ends up getting mapped to b?"

Coordinate Representations Considered Harmful*

- First contact with this kind of problem usually via *matrices*
- Q: What is a matrix?
- A: It's a *big block of numbers!*
- And a vector isn't a little arrow, it's a *big list of numbers!*
- Q: What then is matrix-vector multiplication? (E.g., b = Ax)

matrix

$$A = \begin{bmatrix} 7 & 3 & 4 & 3 \\ 4 & 0 & 1 & 2 \\ 8 & 9 & 4 & 2 \\ 3 & 0 & 1 & 8 \end{bmatrix}$$

vector

$$x = \begin{bmatrix} 2 \\ 5 \\ 1 \\ 3 \end{bmatrix}$$

what the heck is this doing??

$$\text{for } i = 1, \ldots, r$$

$$\text{for } j = 1, \ldots, c$$

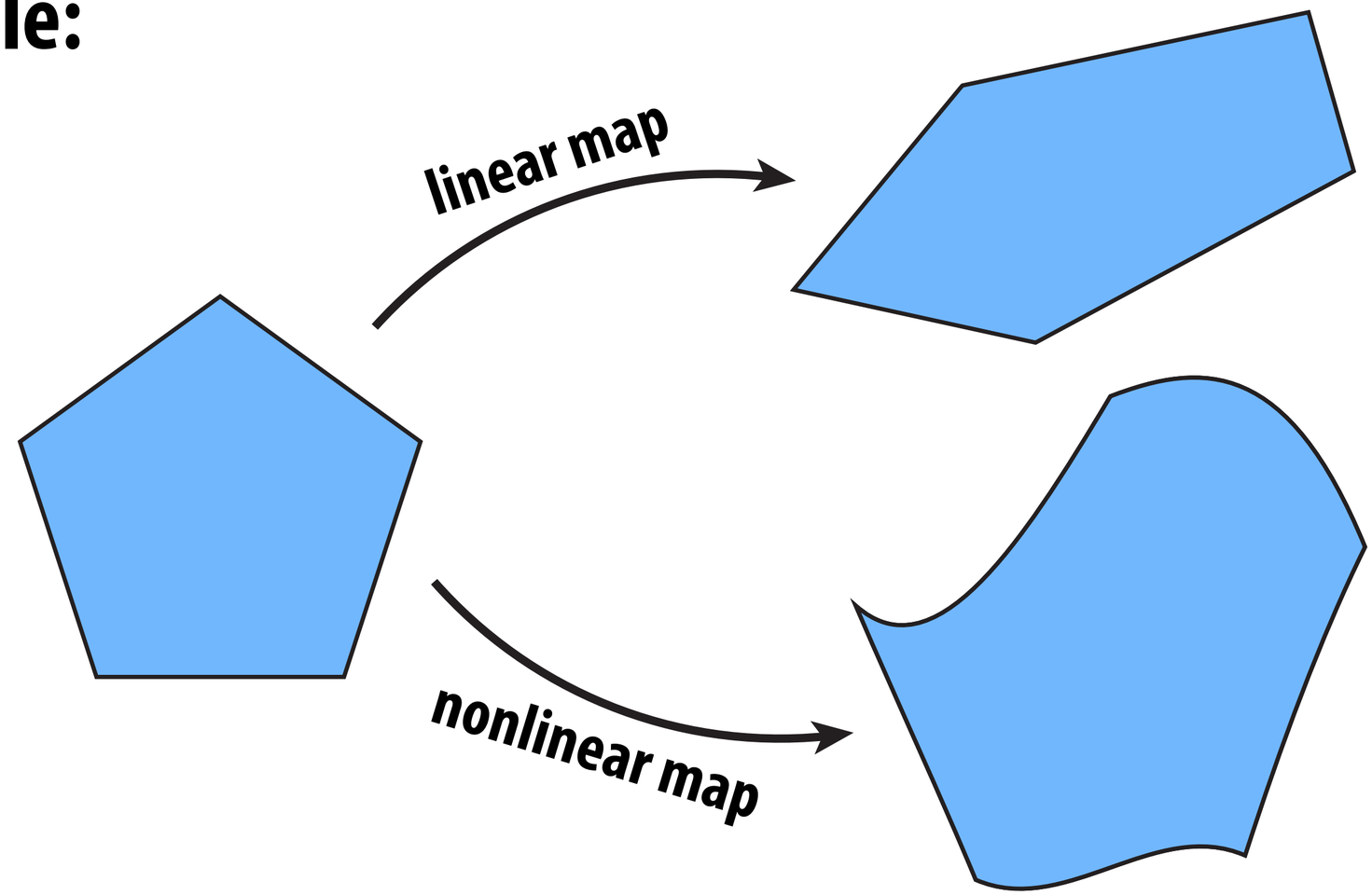
$$b_i \mathrel{+}= A_{ij} x_j$$

Linear algebra is not about pushing blocks of numbers around.

*See Dijkstra, "Go To Statement Considered Harmful" (CACM 1968)

Visual Linear Algebra

- If not coordinates, how do we think about linear maps?
- Especially in graphics, can think about them *visually*.
- Example:

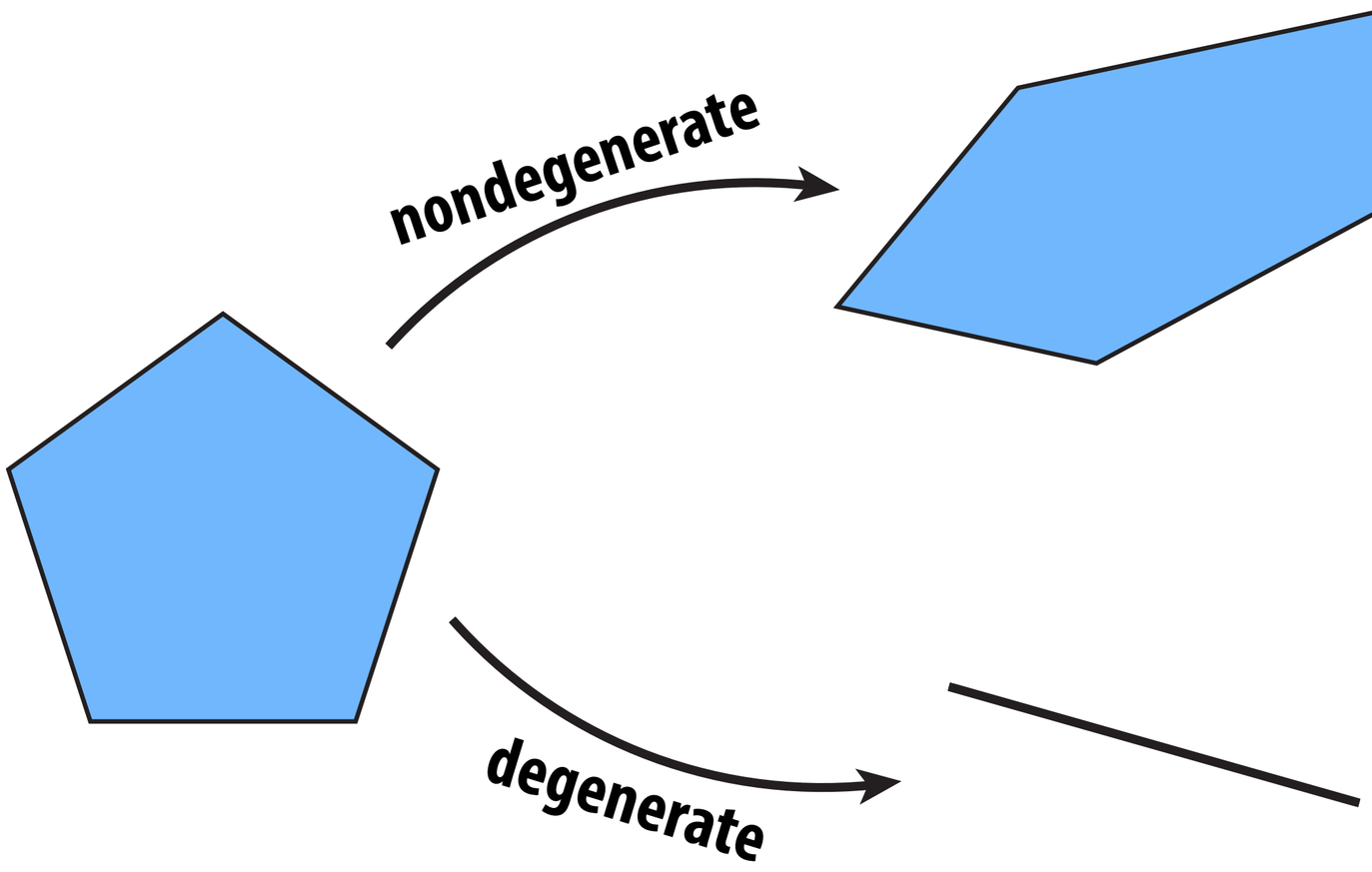


Key idea: *linear maps take lines to lines

***while keeping the origin fixed.**

Visual Linear Algebra - Linear Isomorphisms

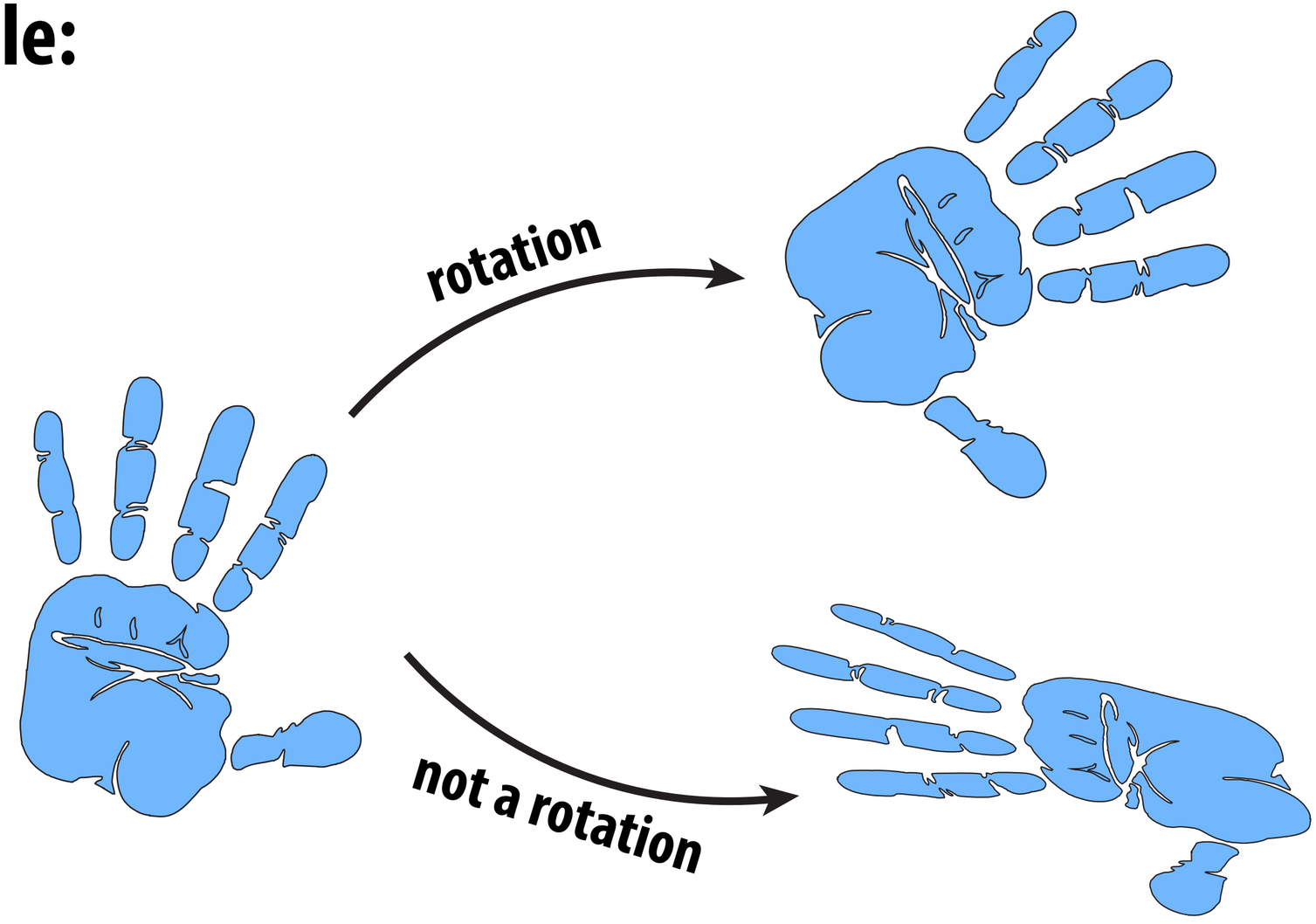
- Other names: nondegenerate, one-to-one, bijective…
- Example:



Key idea: *linear isomorphisms are linear maps that preserve dimension*

Visual Linear Algebra - Rotations

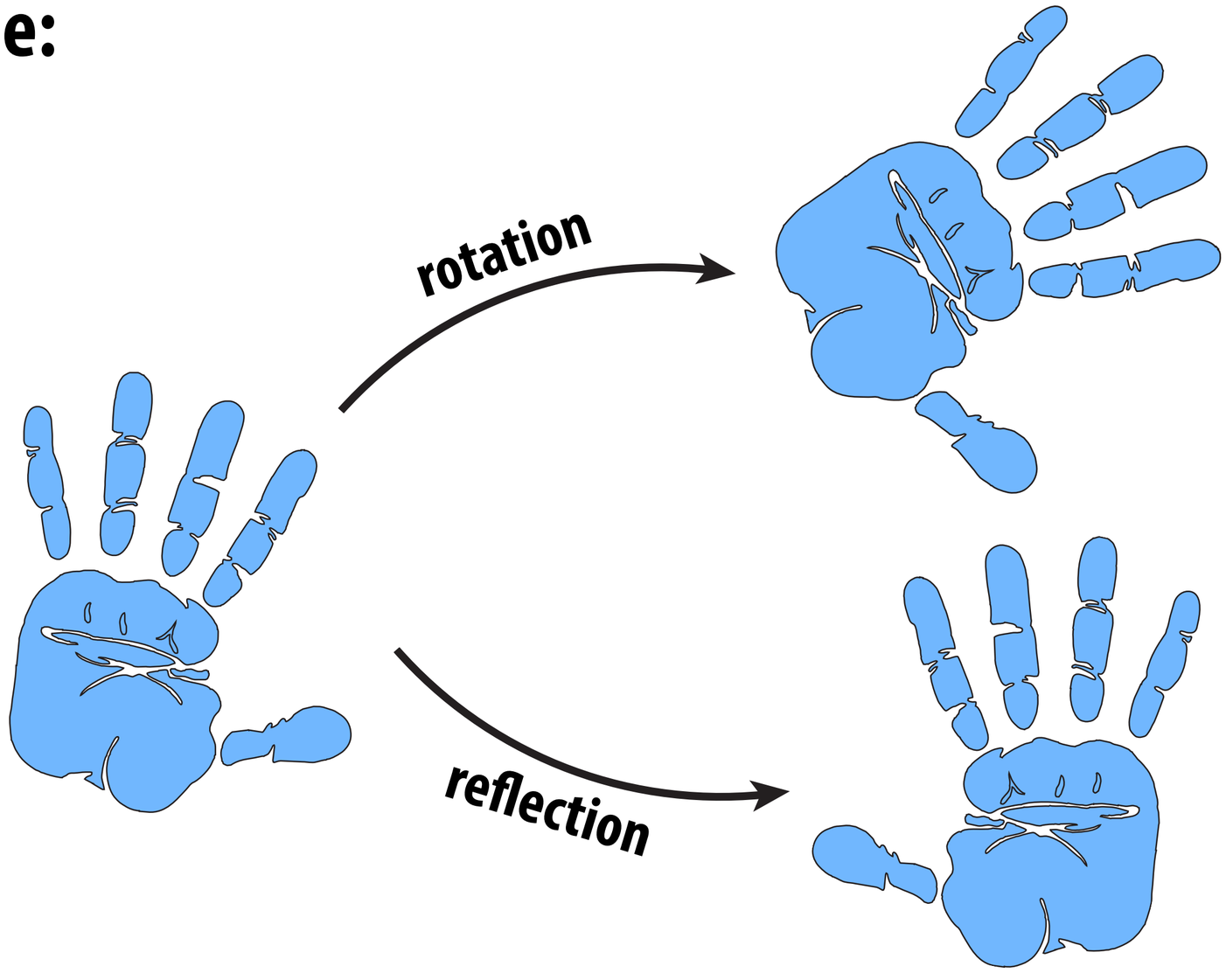
- Other names: special orthogonal, SO(n), ...
- Example:



Key idea: *rotations are linear maps that preserve scale, angles, and orientation*

Visual Linear Algebra - Reflections

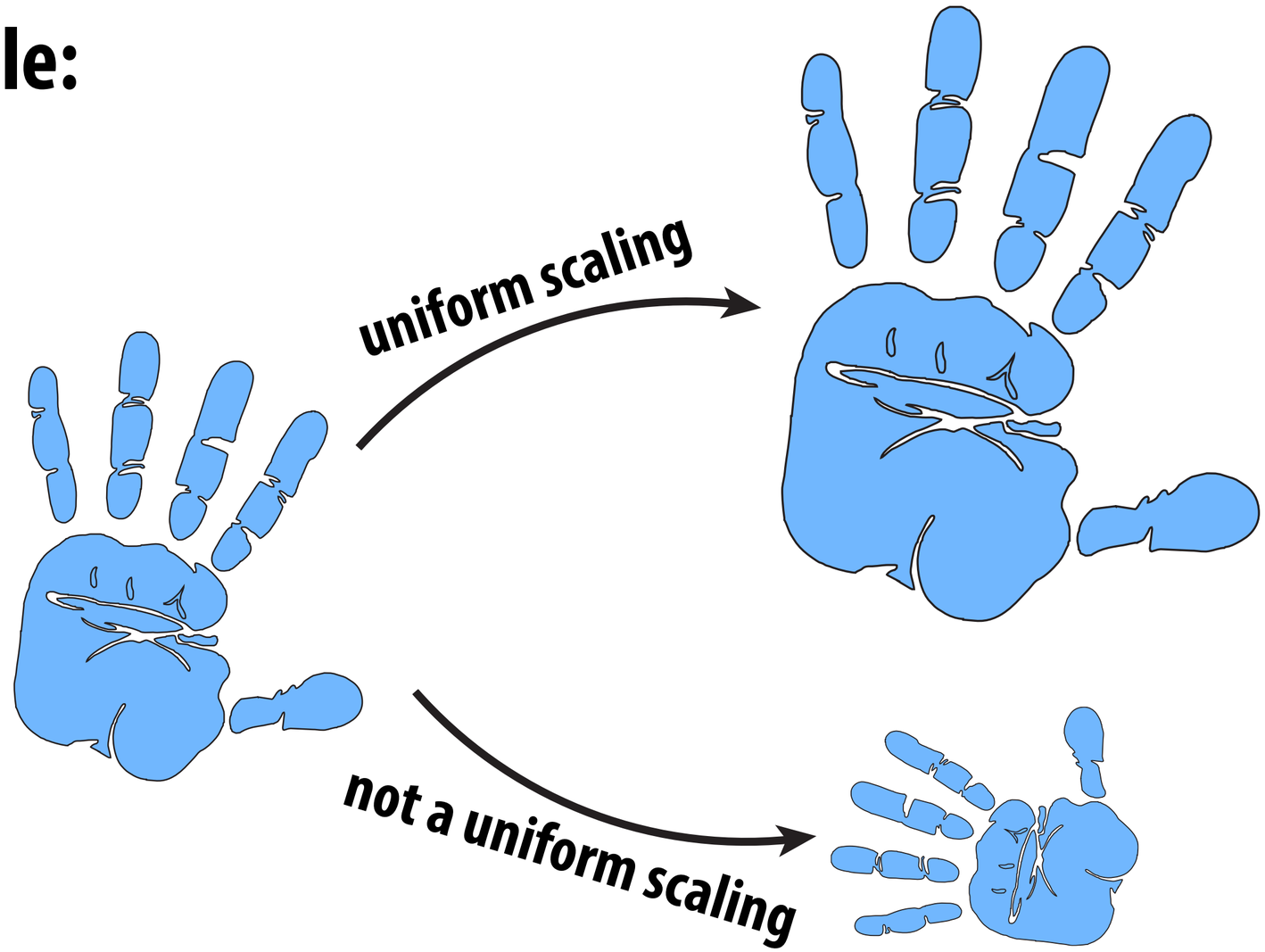
- Other names: *part of* "orthogonal group" O(n), *example of* similarity
- Example:



Key idea: *reflections are linear maps that preserve scale and angles, but flip orientation*

Visual Linear Algebra - Uniform Scaling

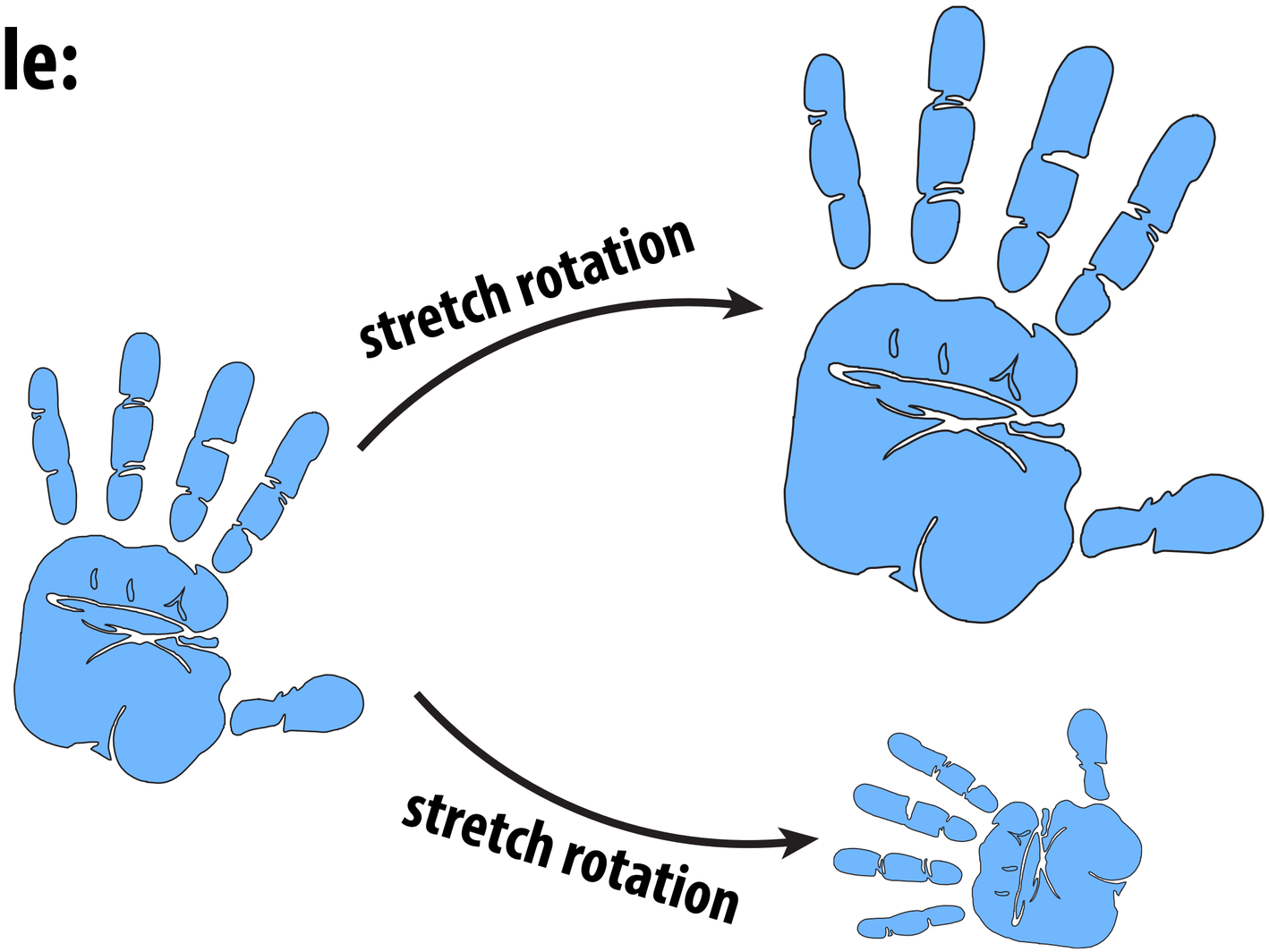
- Other names: uniform stretch, (linear) homothety, dilation
- Example:



Key idea: *uniform scaling preserves everything but scale!*

Visual Linear Algebra - “Stretch Rotation”

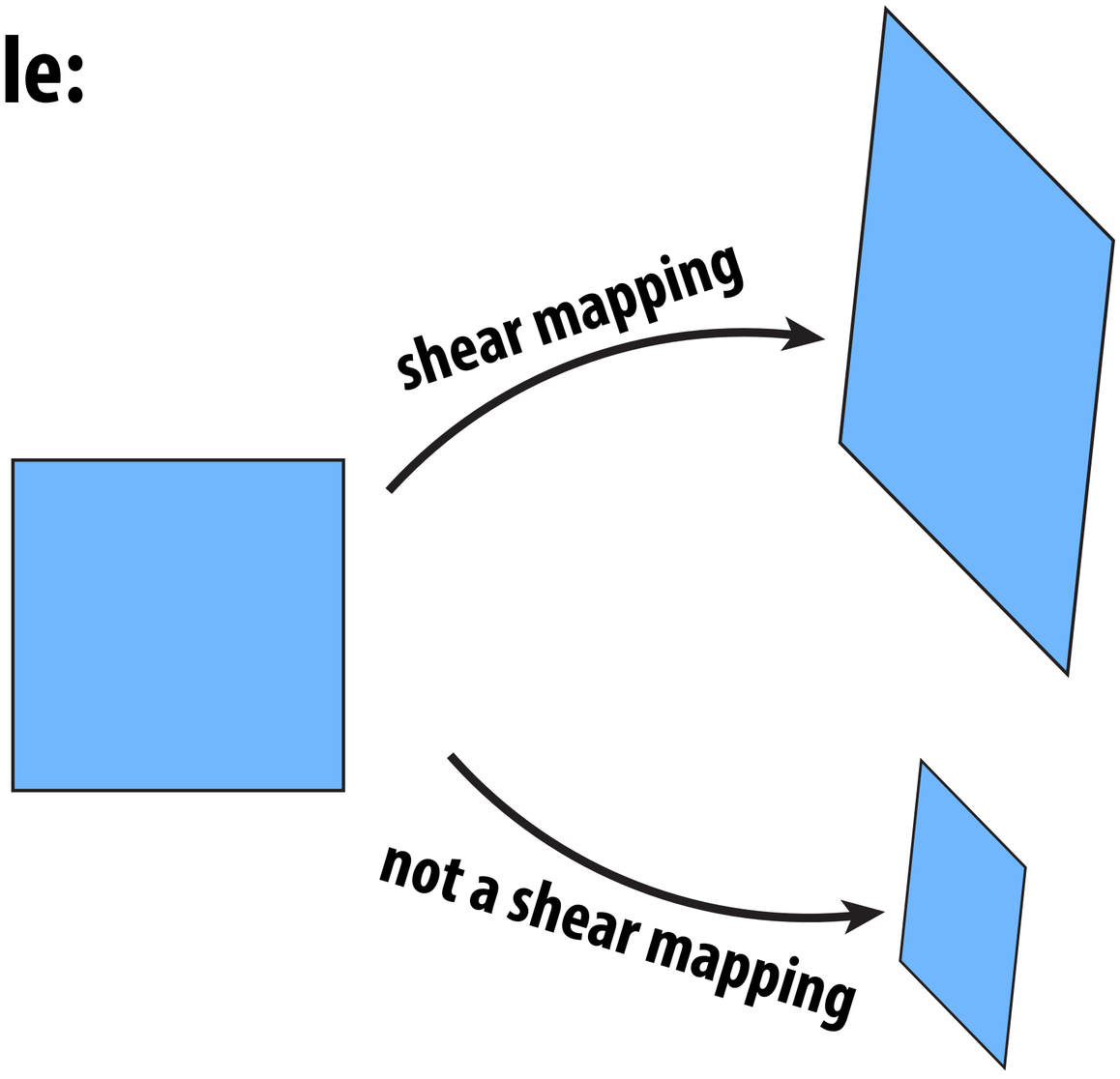
- Other names: ???
- Example:



Key idea: *stretch rotations preserve angles + orientation*

Visual Linear Algebra - Shear Mapping

- Other names: ???
- Example:



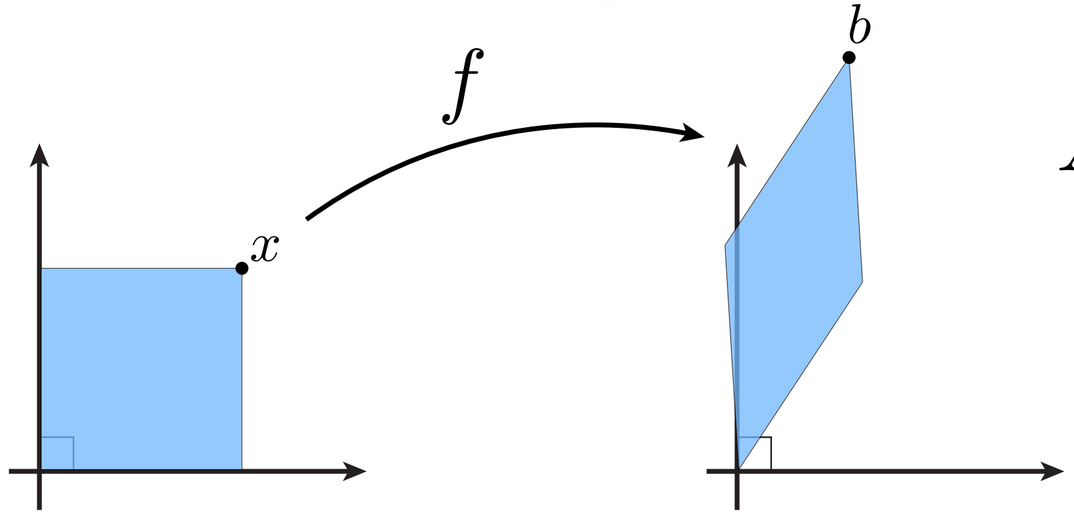
Key idea: *shears preserve areas/volumes*

Note that we did not need coordinates to (rigorously!) define various linear maps.

We just said *which quantities are preserved*.

Linear Algebra in Coordinates

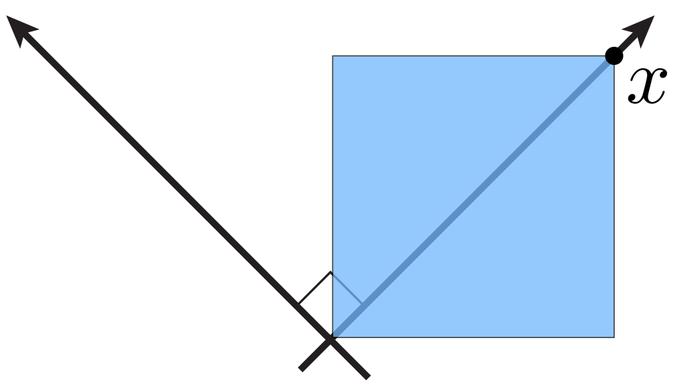
- For computation, we do ultimately need to build matrices.
- So now we can ask: how do geometric properties of our maps correspond to algebraic properties of matrices encoding them?
- To begin with, any linear map can be encoded by a matrix:



$$A = \begin{bmatrix} -1+2\sqrt{3} & -2+\sqrt{3} \\ 2+\sqrt{3} & 1+2\sqrt{3} \end{bmatrix} /4$$

(How did I get this? See what the map f does to basis vectors (1,0) and (0,1); those points become the columns of my matrix A.)

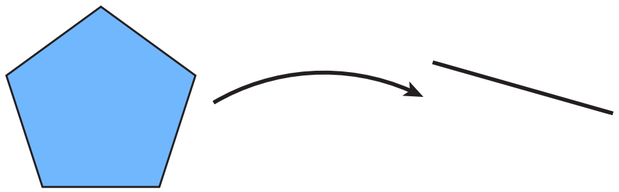
- WARNING: different basis yields completely different matrices!



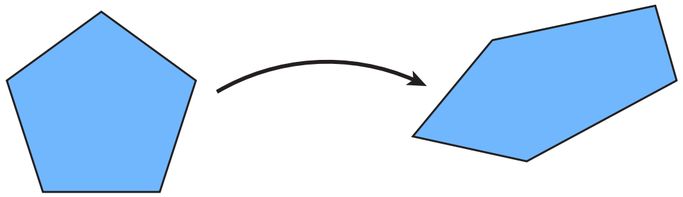
$$A = \begin{bmatrix} \sqrt{2}+3\sqrt{6} & \sqrt{2}\left(-1+3\sqrt{3}\right) \\ 2\sqrt{6-3\sqrt{3}} & \sqrt{2}\left(3+\sqrt{3}\right) \end{bmatrix} /8$$

Linear Algebra in Coordinates

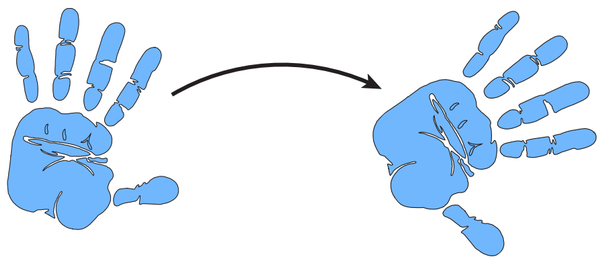
- **Geometric properties now become conditions on matrices:**



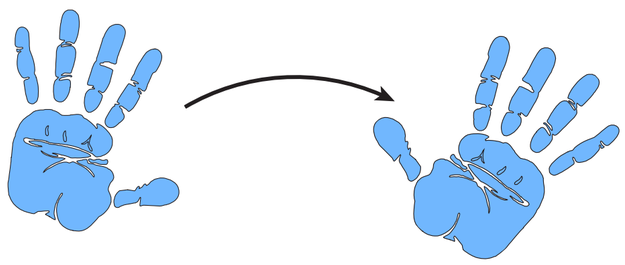
LINEAR MAP — **(no conditions on matrix)**



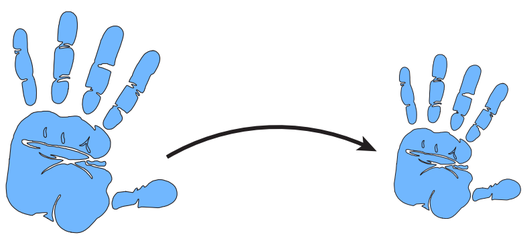
LINEAR ISOMORPHISM — $\det(A) \neq 0$



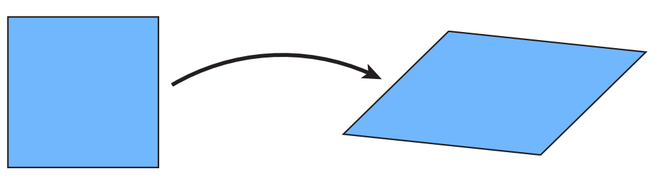
ROTATION — $A^T A = I,\ \det(A) > 0$



REFLECTION — $A^T A = I,\ \det(A) < 0$



UNIFORM SCALING — $A = cI,\ c \in \mathbb{R}$



SHEAR — $A = \begin{bmatrix} I & M \\ 0 & I \end{bmatrix}$

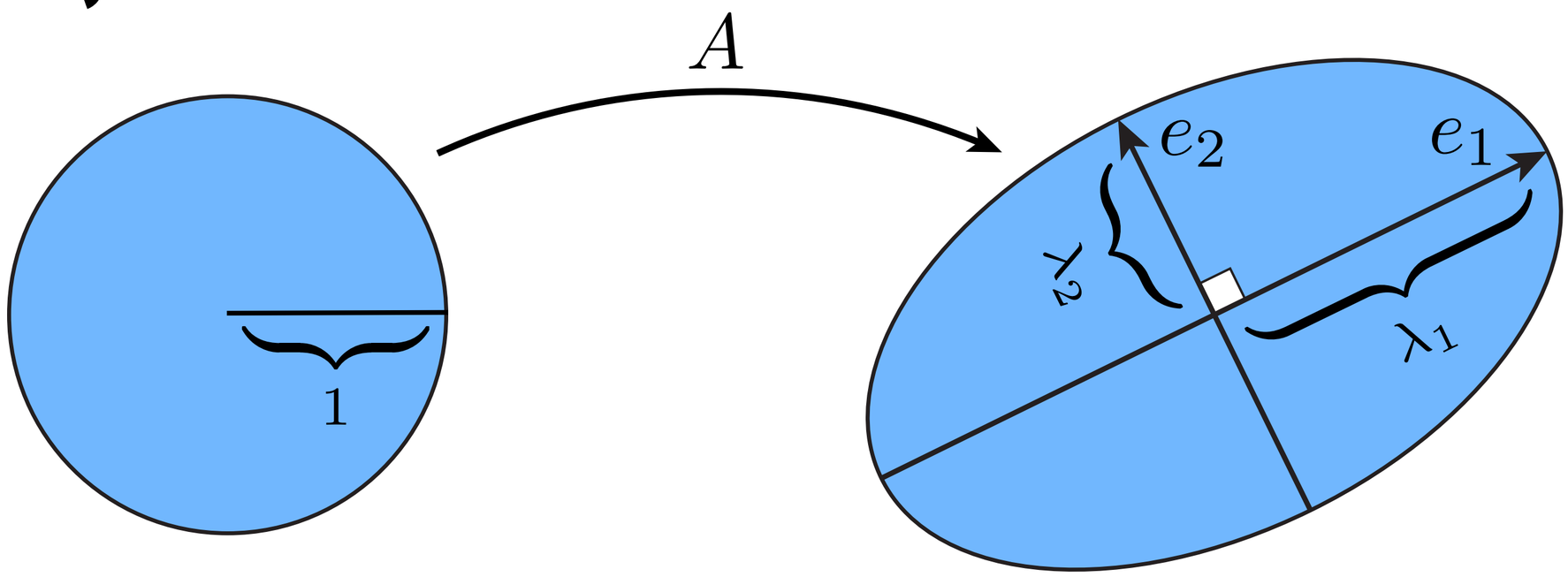
What about the other direction?

Given an algebraic property, what does it mean visually?

Here's one (extremely) important example....

Visual Linear Algebra - Symmetric Matrix

- **Algebraically, matrix A is *symmetric* if $A^T = A$**
- **Visually?**



Key idea: unequal stretch along orthogonal axes

Spectral theorem: symmetric matrix has real eigenvalues (stretch amounts), orthogonal eigenvectors (stretch directions)

$$Ae_i = \lambda_i e_i$$

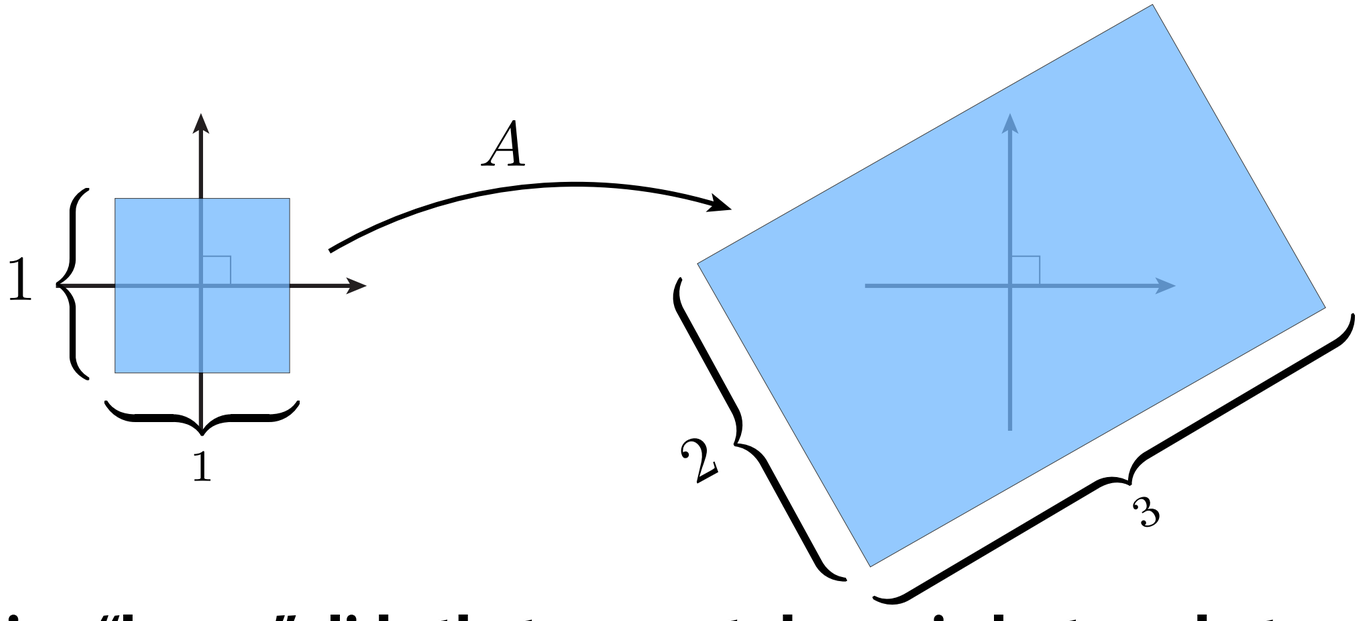
Visual Linear Algebra - Determinant (BONUS*)

- The *determinant* is another important idea in linear algebra
- Often introduced via another procedure (something like "recursively take alternating sums of subdeterminants"):

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} = aei + bfg + cdh - ceg - bdi - afh$$

what the heck is this doing??

- But like many things in linear algebra, has a simple geometric meaning (that does not depend on coordinates or matrices!)
- Determinant says how *volume* grows/shrinks under linear map



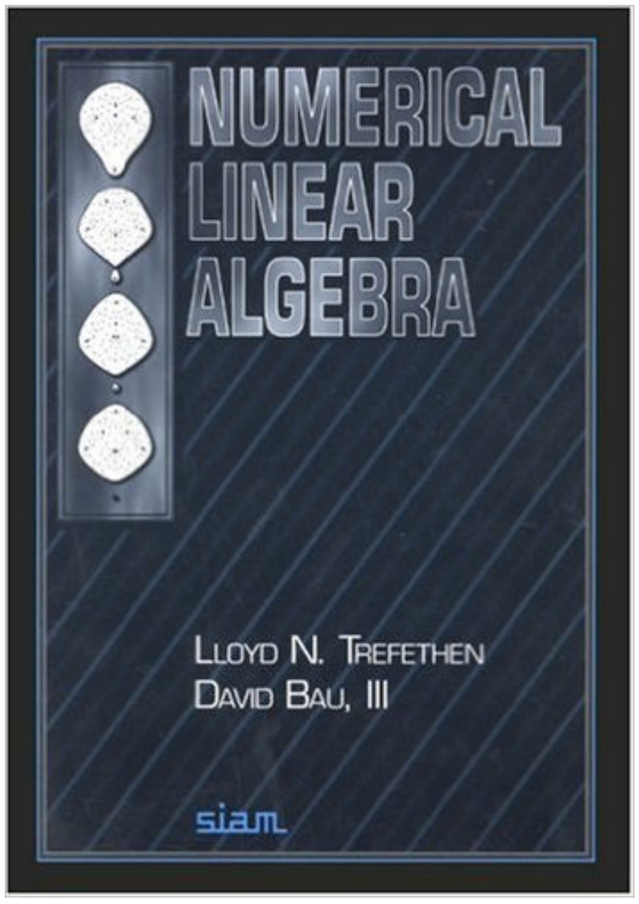
$$\det(A) = 6$$

*This is a "bonus" slide that was not shown in lecture, but was discussed.

In general, studying & decomposing linear transformations in terms of these atomic "pieces" will help us figure out how to solve numerical linear algebra problems.

Strategy for Learning Numerical Linear Algebra

- A LOT to know about numerical linear algebra
- Good strategy for learning/using it in practice:
  1. Get a good sense for different problems
     - underdetermined vs. overdetermined
     - definite vs. indefinite
     - sparse vs. dense
  2. Get a sense for which solvers work for which problems
  3. Gain a detailed knowledge of individual algorithms

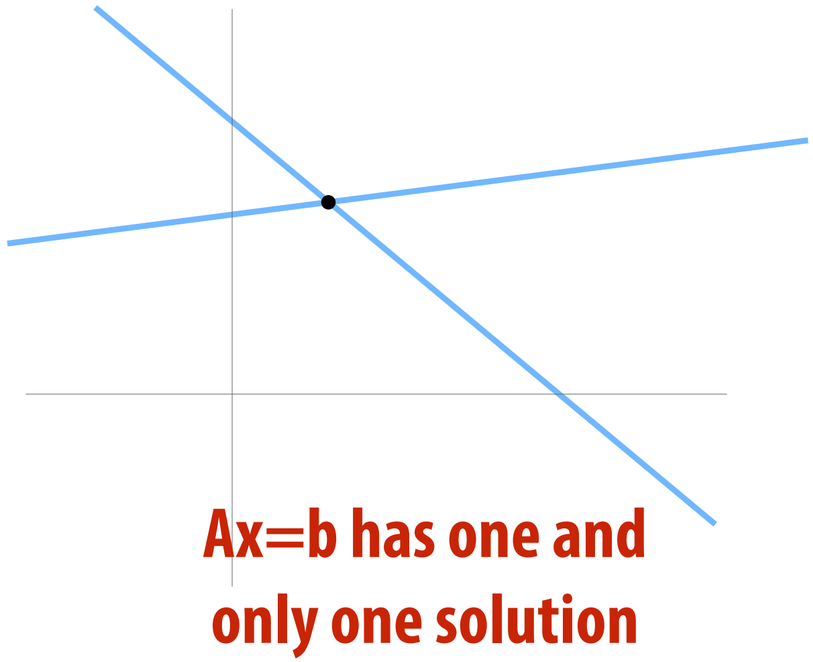


What are the most important features of a numerical linear algebra problem?

Unique / Underdetermined / Overdetermined

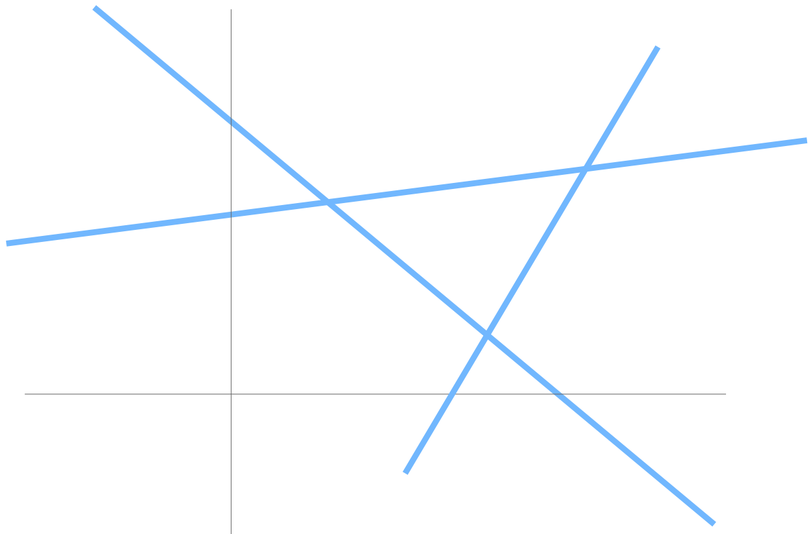
- Basic question about any problem: are there solutions?
- If so, how many?

UNIQUE



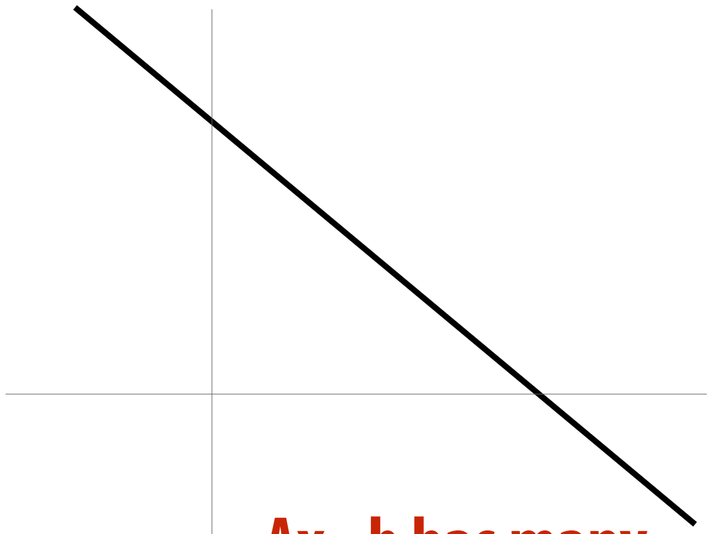
Ax=b has one and only one solution

OVERDETERMINED



Ax=b has no solutions

UNDERDETERMINED



Ax=b has many solutions

- In practice...
  - sometimes happy with *any* solution
  - sometimes want to know *all* solutions
  - sometimes ok with being *"as close as possible"* to solving all of our equations simultaneously (least-squares)

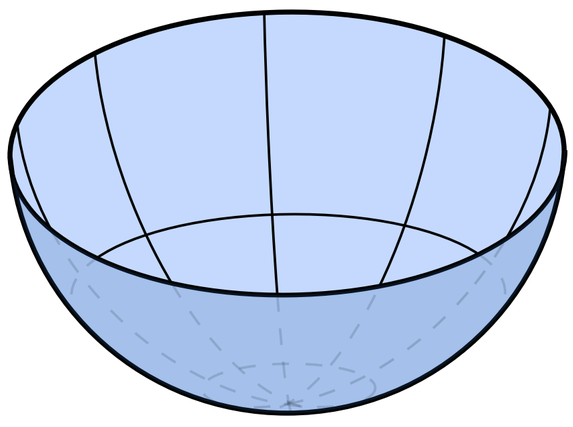
Definite / Semidefinite / Indefinite

- **Every linear system is the critical point of an *objective* or *energy***

$$f_0(x) := \frac{1}{2}x^T A x - b^T x$$

- **"Shape" of this energy tells us important information about how hard it is to find solutions:**

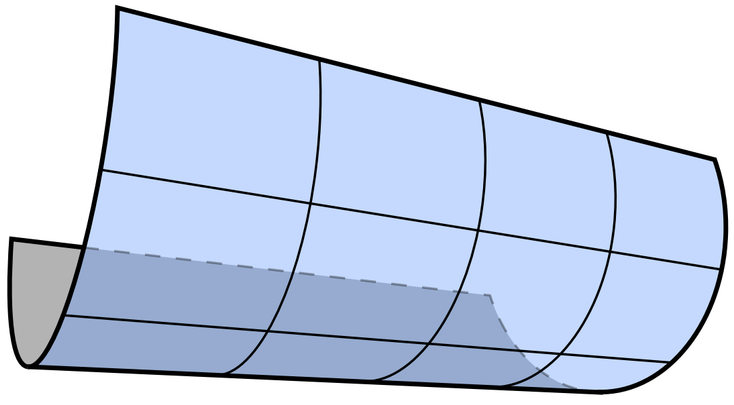
(EASY: just ski to the bottom!)



positive definite

$$x^T A x > 0$$

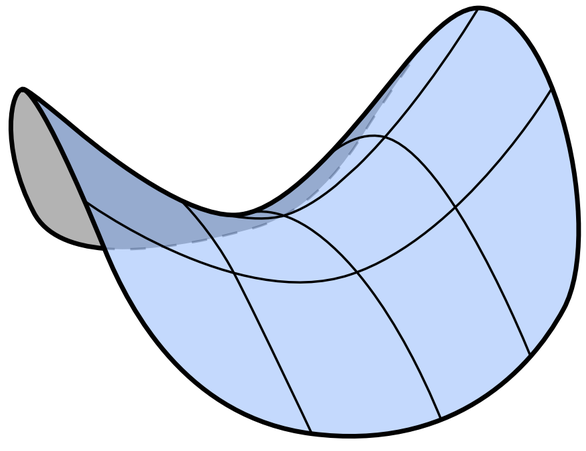
(HARDER: many "bottoms"!)



positive semidefinite

$$x^T A x \geq 0$$

(HARDEST: no bottom!)



indefinite

(positive *or* negative, depending on x)

for all x

Dense vs. Sparse

DENSE

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 \\ 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 \\ 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

"most" entries are nonzero
(~O(n²) nonzero values)

SPARSE

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

"most" entries are zero
(~O(n) nonzero values)

- **EXTREMELY important distinction in graphics (e.g., Poisson)**
- **Techniques for dense & sparse problems are very different**
- **Solving sparse problem with dense solver is *DUMB, DUMB, DUMB!***
- **O(n²) instead of O(n)—for no good reason at all!**

Representations of Linear Systems

- How do you encode a linear system on a computer?
- One idea: arrays of values

```
double A[nRows][nColumns];
```
(DENSE)

- Another idea: list of (row,column,value) tuples

```
int row[n], column[n];
double value[n];
```
(SPARSE)

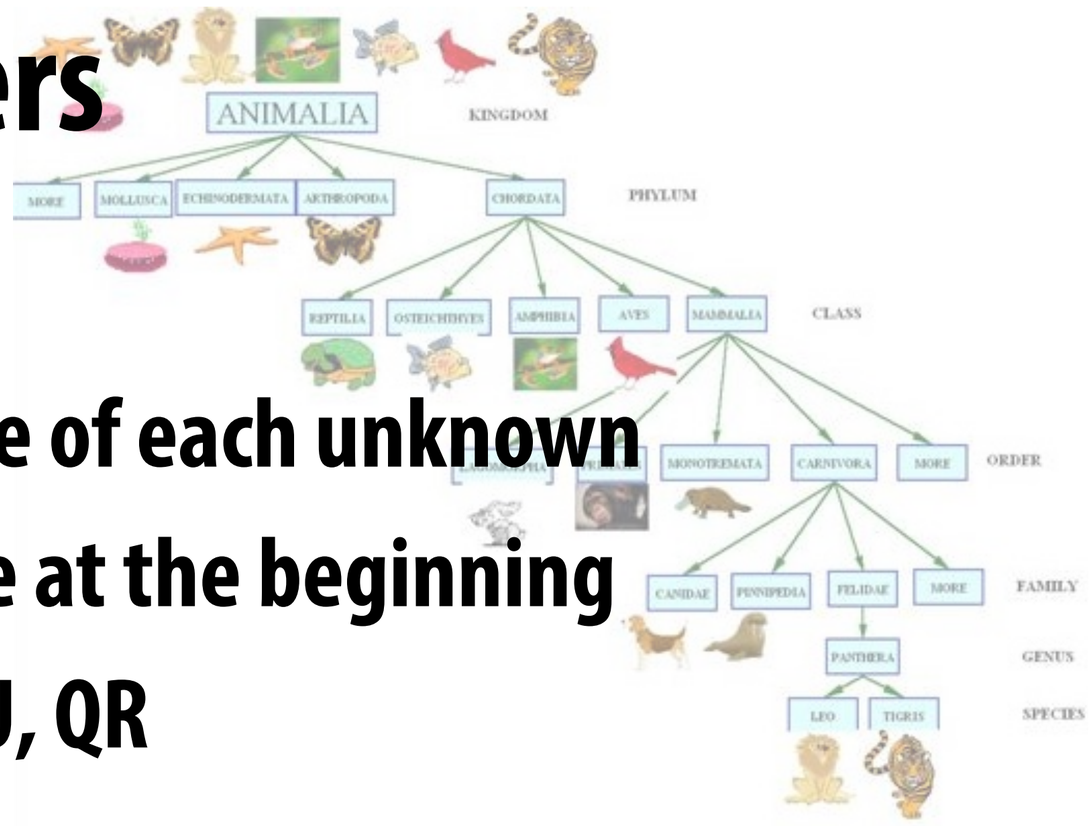
- Q: Which of these better suited to dense/sparse matrices?
- A third, less obvious idea: *callback function*

```
f( const double* x, double* Ax );
```

- Result Ax must be same as multiplying with A, but...
- ...don't have to explicitly build matrix! (Good for, e.g., Laplace)

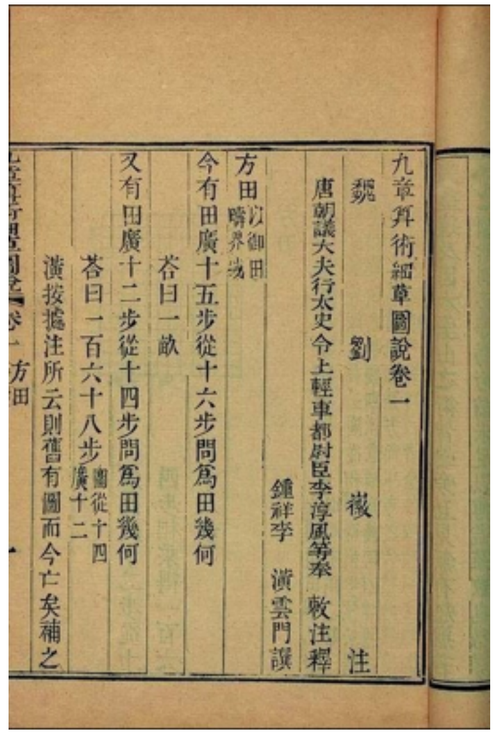
Which solvers work best for which problems?

Taxonomy of Linear Solvers



- **DIRECT**
  - Immediately solve for final value of each unknown
  - Much like our "by-hand" exercise at the beginning
  - Common examples: Cholesky, LU, QR
- **ITERATIVE**
  - Incrementally move unknowns toward solution
  - More like our "heat flow" solution to Laplace (last lecture)
  - Common examples: Jacobi/Gauss-Seidel, CG, multigrid
- **DIRECT + ITERATIVE**
  - Can also combine features of both approaches
  - E.g., Cholesky as *preconditioner* for conjugate gradient (CG)

Direct Solvers



- Immediately solve for final value of each unknown
- Prototypical algorithm: *Gaussian elimination*
- Much like our "by hand" calculation:

$$\left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 11 & 5 & 35 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 2 & 8 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 0 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

- Rough idea: try to isolate one variable at a time
- Once a variable is known, it can be treated like a *constant*
- *Backsubstition:* plug these "constants" into earlier equations
- History: invented in China—*not by Gauss!*—around 150 BCE

Direct Solvers - PROS & CONS

- **PROS:**
  - typically "most accurate in class" (machine precision)
  - superb amortized cost w/ prefactorization (next slide)
  - robust for over/underconstrained problems / *rank-revealing*
  - don't need special problem knowledge for good performance
- **CONS:**
  - need explicit representation of matrix (no callbacks)
  - storage can be a problem for large sparse problems (*"fill-in"*)
  - need intelligent *reordering* (but good heuristics exist)
  - historically worse asymptotic performance than "best-in-class" iterative methods (though that is changing...)

Iterative Solvers

- Gradually improve solution until convergence
- Prototypical algorithm: *Jacobi method*
- Much like the way we solved Laplace in PDE lecture:

	c	
d	a	b
	e	

$$\frac{4a - b - c - d - e}{h} = 0$$

$$\iff a = \tfrac{1}{4}(b + c + d + e)$$

- Rough idea: push each variable closer to final condition
- Do it over & over again until you get close enough
- History: well, it was invented by Gauss...

Iterative Solvers - PROS & CONS

- **PROS:**
  - can be extremely easy to implement/parallelize (e.g., Jacobi)
  - often low memory cost (O(n))
  - can use callbacks rather than explicitly building matrix
  - multigrid has ideal asymptotic complexity (O(n) iterations)

- **CONS:**
  - need special problem knowledge for good performance (preconditioners or multiresolution hierarchy)
  - often poor precision as a function of true wall clock time
  - poor amortized cost for repeated solves w/ different data
  - not as nice as direct for over/underdetermined problems; not rank-revealing

Conclusion: *pick the right tool for the job!*

Linear Algebra Software

- So, where can you get your hands on some tools?
- Dirty little secret: almost all software packages for numerical linear algebra are built on top of the same libraries:
  - DENSE: BLAS / LAPACK / ATLAS / …
  - SPARSE: SuiteSparse (used by *MATLAB*, *Mathematica*, …)
- But there are literally zillions of other packages out there, each tailored to a special purpose
- Also some nice wrappers (e.g., "Eigen" in C++, NumPy/SciPy in Python)… which often use the solvers above!
- Conclusion: absolutely no need to pay money for linear solvers!
- (Also makes your own code less useful/distributable).
- OPPORTUNITY: GOOD linear solvers for JavaScript / web?

Next up: Fourier Transform & Applications