

Lecture 3

Linear Systems II

Conditioning & introduction to least squares problems

**CS328 - Numerical Methods for
Visual Computing and Machine Learning**

Prof. Wenzel Jakob

Where are we now?

Veggies first, then the cake



MidJourney: two plates next to each other. one contains heaps of carrots, cucumbers, and tomatoes. The other is full of sumptuous cake and candy.
Hyper-realistic, f/1.4

1. Floating Point arithmetic & error



2. Linear systems, LU

$$[A] = [L] [U]$$

4. Least squares, QR

$$[A] = [Q] [R]$$

5. Writing efficient numerical code

```
vmovaps ymm5, cs:ymmword_100004840 ; Move aligned memory
vmovaps ymm3, cs:ymmword_100004860 ; Move aligned memory
vandnps ymm7, ymm5, ymm1 ; Bitwise logical AND NOT (256
vandnps ymm6, ymm5, ymm0 ; Bitwise logical AND NOT (256
vminps ymm4, ymm7, ymm6 ; Packed FP minimum (8-wide)
vmaxps ymm2, ymm6, ymm7 ; Packed FP maximum (8-wide)
vcvtpgt_oqps ymm6, ymm6, ymm7 ; Packed FP comparison -- d
```

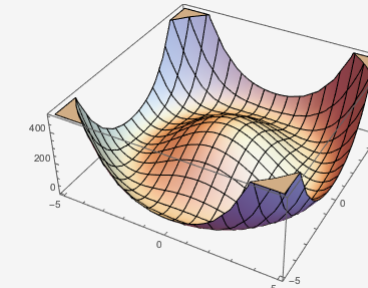
3. Conditioning & least squares basics

$$[A^T A] [x] = [A^T b]$$

6. Eigenanalysis, SVD

$$[A] = [U] [\Sigma] [V^T]$$

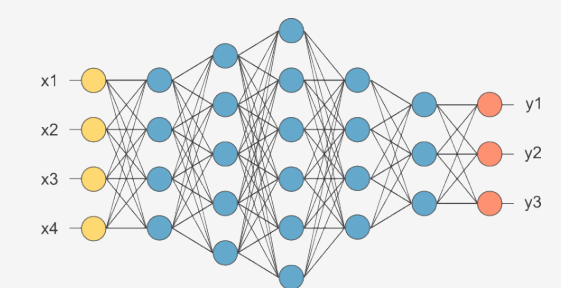
7. Nonlinear problems



8. Aut. differentiation

PYTORCH
TensorFlow

9. Neural networks



Solving a linear system using Gaussian Elimination

$$\left[\mathbf{A} \mid \mathbf{b} \right] = \left[\begin{array}{cccc|c} \textcircled{\times} & \times & \times & \times & \times \\ 0 & \textcircled{\times} & \times & \times & \times \\ 0 & 0 & \textcircled{\times} & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array} \right]$$

pivot element

Observation: None of the decisions taken by the algorithm depend on the right hand side.

Recap: LU factorization

$$\begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} L \end{bmatrix} \begin{bmatrix} U \end{bmatrix}$$

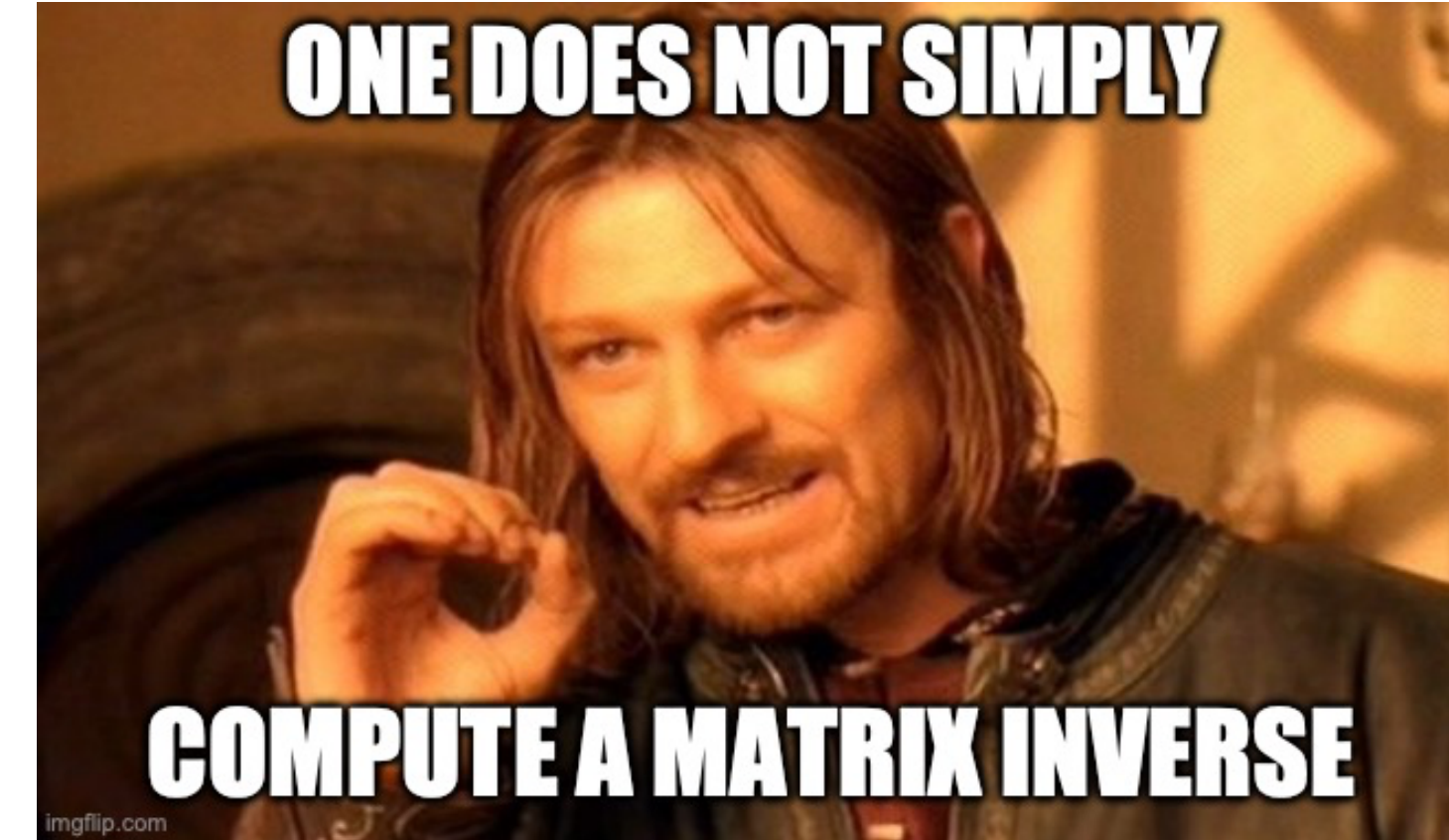
$$U = \underbrace{E_6 E_5 E_4 E_3 E_2 E_1}_{} A$$

$$A = LU$$

$$\underbrace{\left(E_6 E_5 E_4 E_3 E_2 E_1 \right)^{-1}}_{= E_1^{-1} E_2^{-1} E_3^{-1} E_4^{-1} E_5^{-1} E_6^{-1}}$$

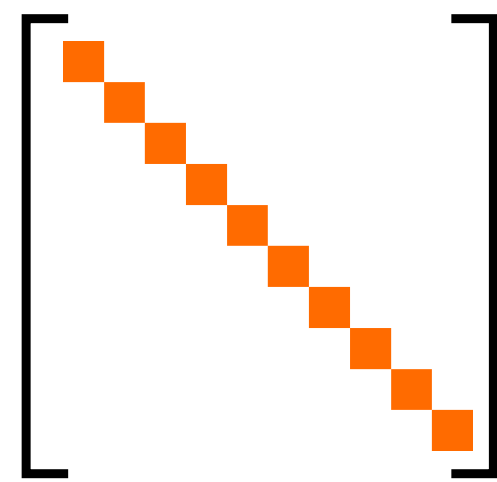
Recap: don't invert (big) matrices

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{b}$$



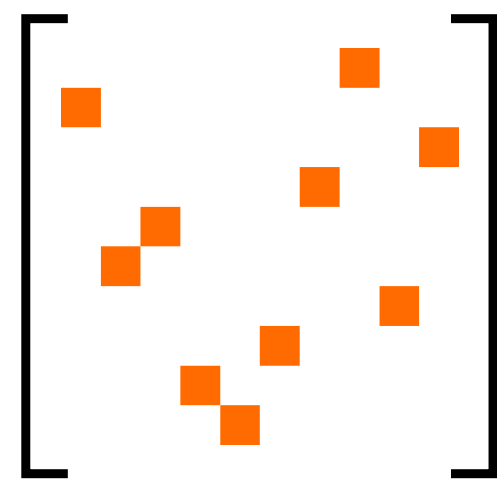
Usually this **does not mean** "compute an inverse of A". Read this as "call a linear system solver to solve $\mathbf{Ax} = \mathbf{b}$ "

What kind of linear system solver?



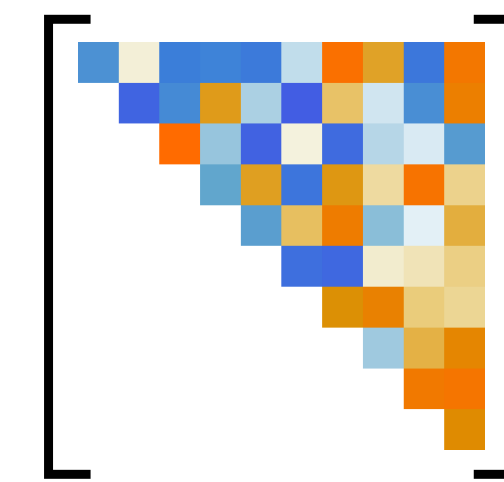
Diagonal

```
for i in range(n):  
    x[i] = b[i] / A[i, i]
```



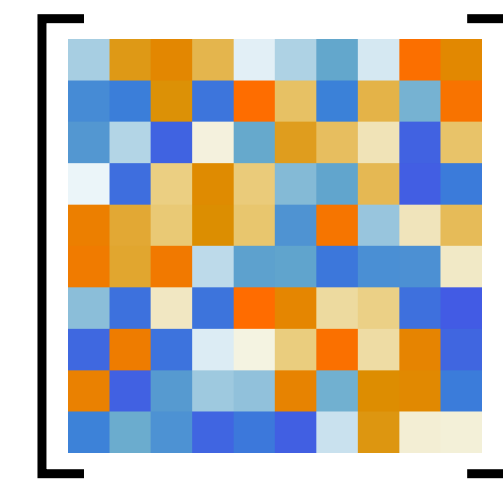
Permutation

```
for i in range(n):  
    x[i] = b[perm[i]]
```



Triangular

Forward/backward substitution



Dense

General solver, e.g. LU factorization

A worked out example

LU factorization of a 2x2 matrix

(Elimination matrix that performs this operation)

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \begin{array}{l} \leftarrow \\ \leftarrow \end{array} * -2$$

$$\mathbf{E}_1 = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

$$\mathbf{E}_1 \mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

$$\mathbf{L} = \mathbf{E}_1^{-1} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$\longrightarrow \mathbf{A} = \mathbf{L}\mathbf{U} \quad \checkmark$$

Cost of matrix operations

1. Matrix-vector multiplication

$\mathbf{b} \in \mathbb{R}^{10000}$
 $\mathbf{A} \in \mathbb{R}^{10000 \times 10000}$
100 million matrix entries!

`b = A @ x` _____ 11 *ms* (measured on an M1 Mac)

`b = np.zeros(n)`
`for i in range(n):` _____ 66 *ms* ~6x slower
 `b[i] = A[i, :] @ x`

`b = np.zeros(n)`
`for j in range(n):` _____ 309 *ms* ~28x slower
 `b += A[:, j] * x[j]`

`b = np.zeros(n)`
`for j in range(n):` _____ 30'282 *ms* ~2'752x slower
 `for i in range(n):`
 `b[i] += A[i, j] * x[j]`

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

Cost of matrix operations

$$A, B \in \mathbb{R}^{1000 \times 1000}$$

2. Matrix-matrix multiplication

1 million matrix entries!

`C = A @ B` _____ 19 ms (measured on an M1 Mac)

```
C = np.zeros((n, n))
for j in range(n):
    C[:, j] = A @ B[:, j]
```

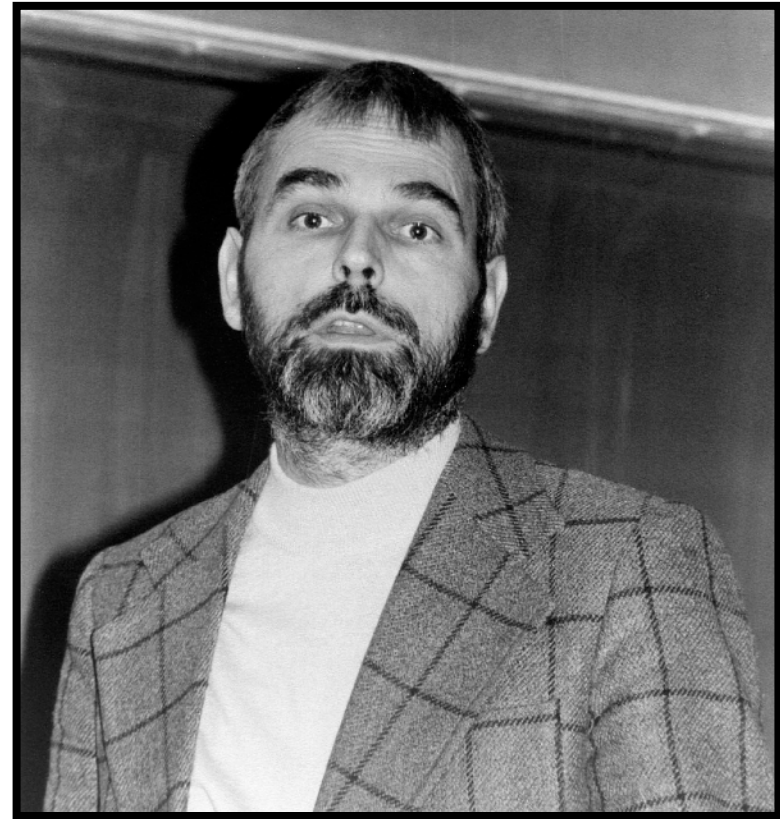
$\mathcal{O}(n^3)$ 236 ms ~6x slower

```
C = np.zeros((n, n))
for i in range(n):
    for j in range(n):
        for k in range(n):
            C[i, j] += A[i, k] * B[k, j]
```

_____ 341'495 ms ~17'973x slower

Cost of matrix operations

$O(n^3)$ time complexity.. is that really true?



Volker Strassen, 1969

"Matrix multiplication has $O(n^{2.8074})$ time complexity."

Still an active area of research



Virginia Williams et al., 2023

"Matrix multiplication has $O(n^{2.371552})$ time complexity."



MidJourney: *a galactic computer*

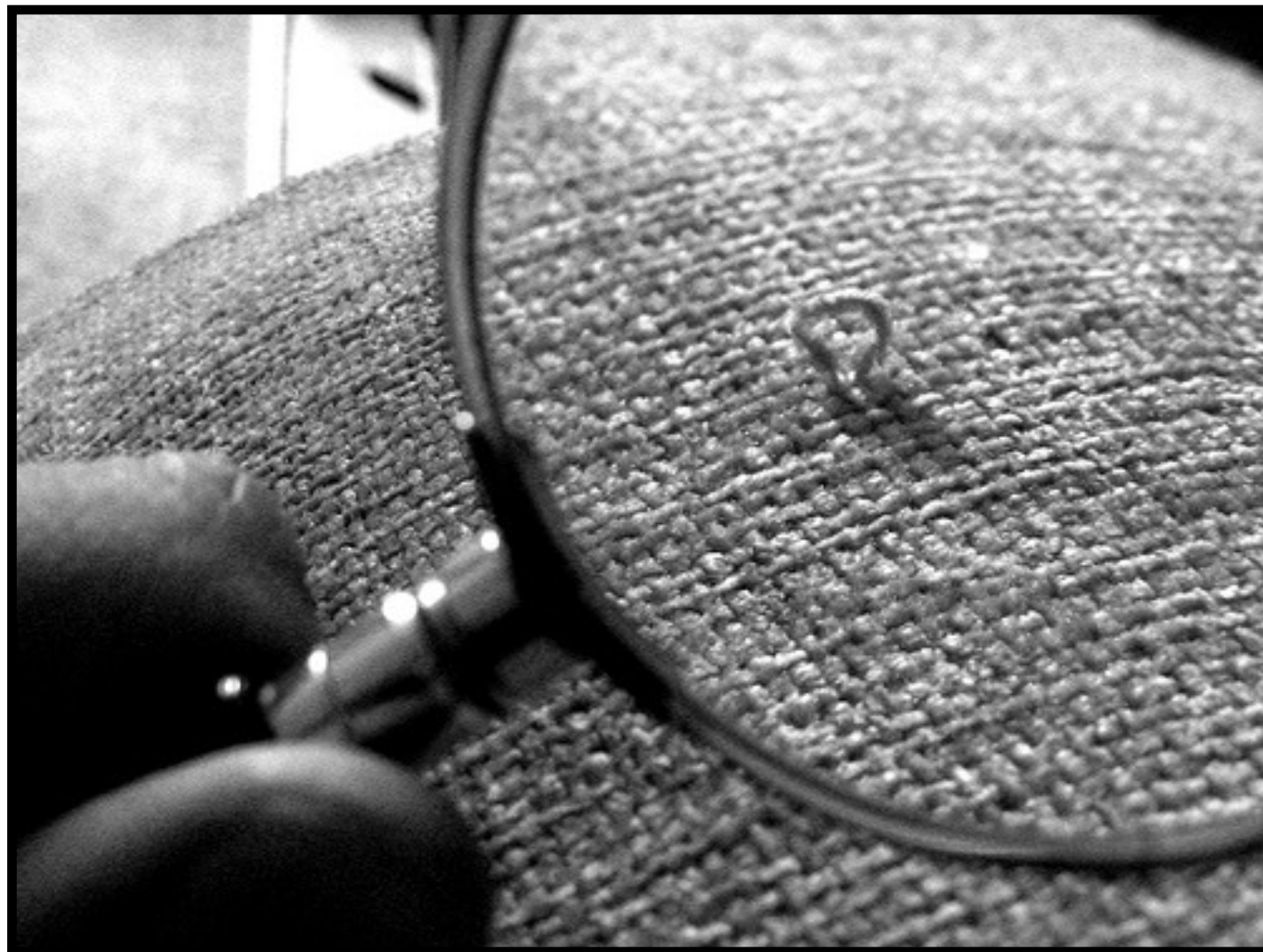
Quite possible that the true cost is $O(n^2)$. *Nobody knows!*

Many works are *galactic* algorithms. In practice, the naïve algorithm tends to be fastest.

Cost of matrix operations

Forward/backward substitution

$$\left[\mathbf{A} \mid \mathbf{b} \right] = \left[\begin{array}{cccc|c} \times & 0 & 0 & 0 & \times \\ 0 & \times & 0 & 0 & \times \\ 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & \times \end{array} \right]$$



Cost of matrix operations

Triangular system solving aka. "substitution".

LU permutation produces $PLU = A$

Usage: solve $Px_1 = b$ (permute), solve $Lx_2 = x_1$ (forward subst.), solve $Ux_3 = x_2$ (backward subst.)

$$b \in \mathbb{R}^{10000}$$

$$A \in \mathbb{R}^{10000 \times 10000}$$

100 million matrix entries!

```
lup = la.lu_factor(A)
x = la.lu_solve(lup, b)
```

Factorization

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

275ms (measured on an M1 Mac)
30ms

```
P, L, U = la.lu(A)
```

```
x1 = P.T @ b
```

```
x2 = la.solve_triangular(L, x1, lower=True)
```

```
x3 = la.solve_triangular(U, x2, lower=False)
```

Substitution

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

102ms 3.4x slower

```
# As above, but using the following function instead
def solve_triangular(M, y, lower):
```

```
    x = np.zeros(n)
```

```
    todo = range(n) if lower else reversed(range(n))
```

```
    for i in todo:
```

```
        x[i] = (y[i] - M[i, :] @ x) / M[i, i]
```

```
    return x
```

150ms 5x slower

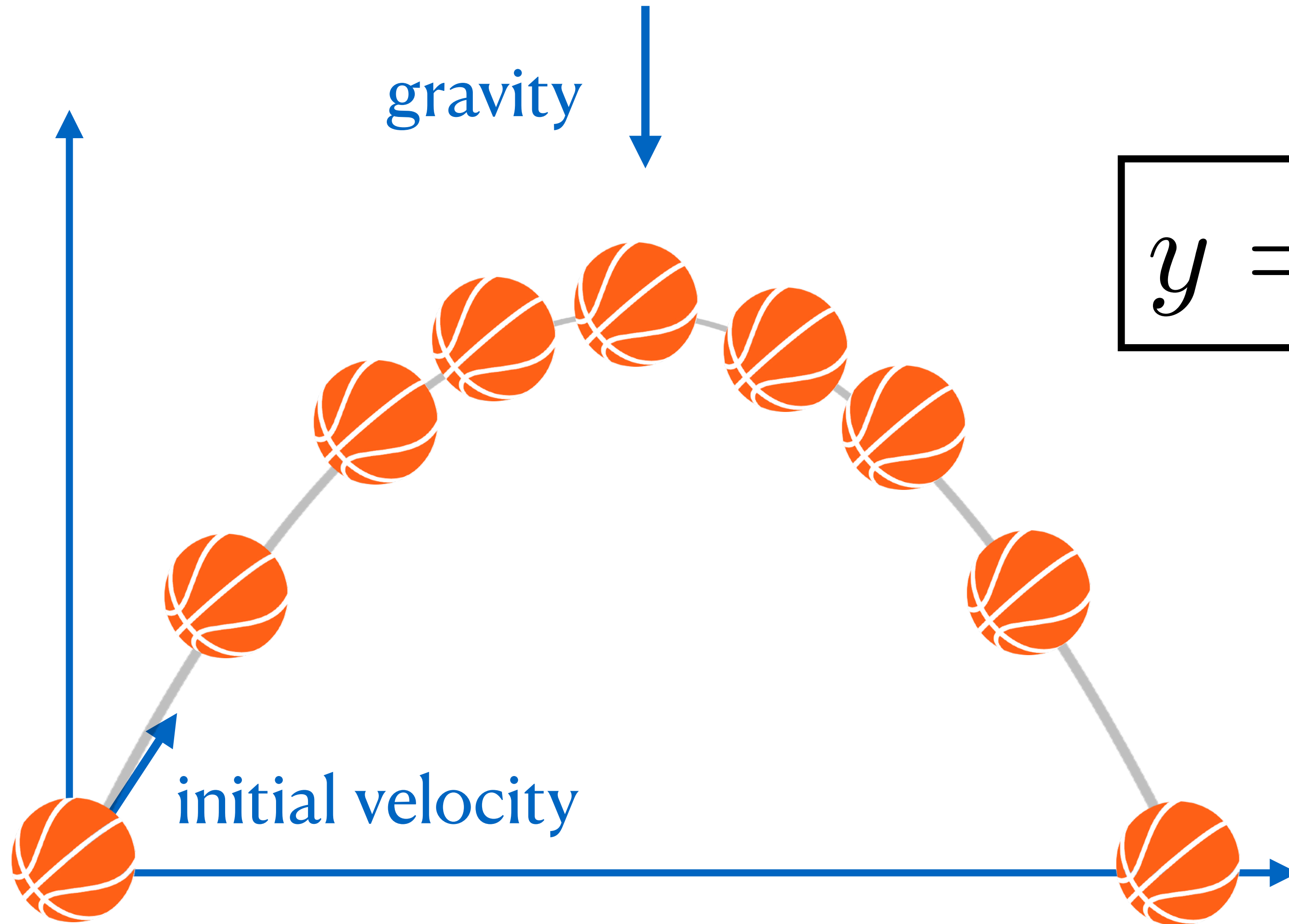
Examples of linear systems

Examples of linear systems



How can we model this?

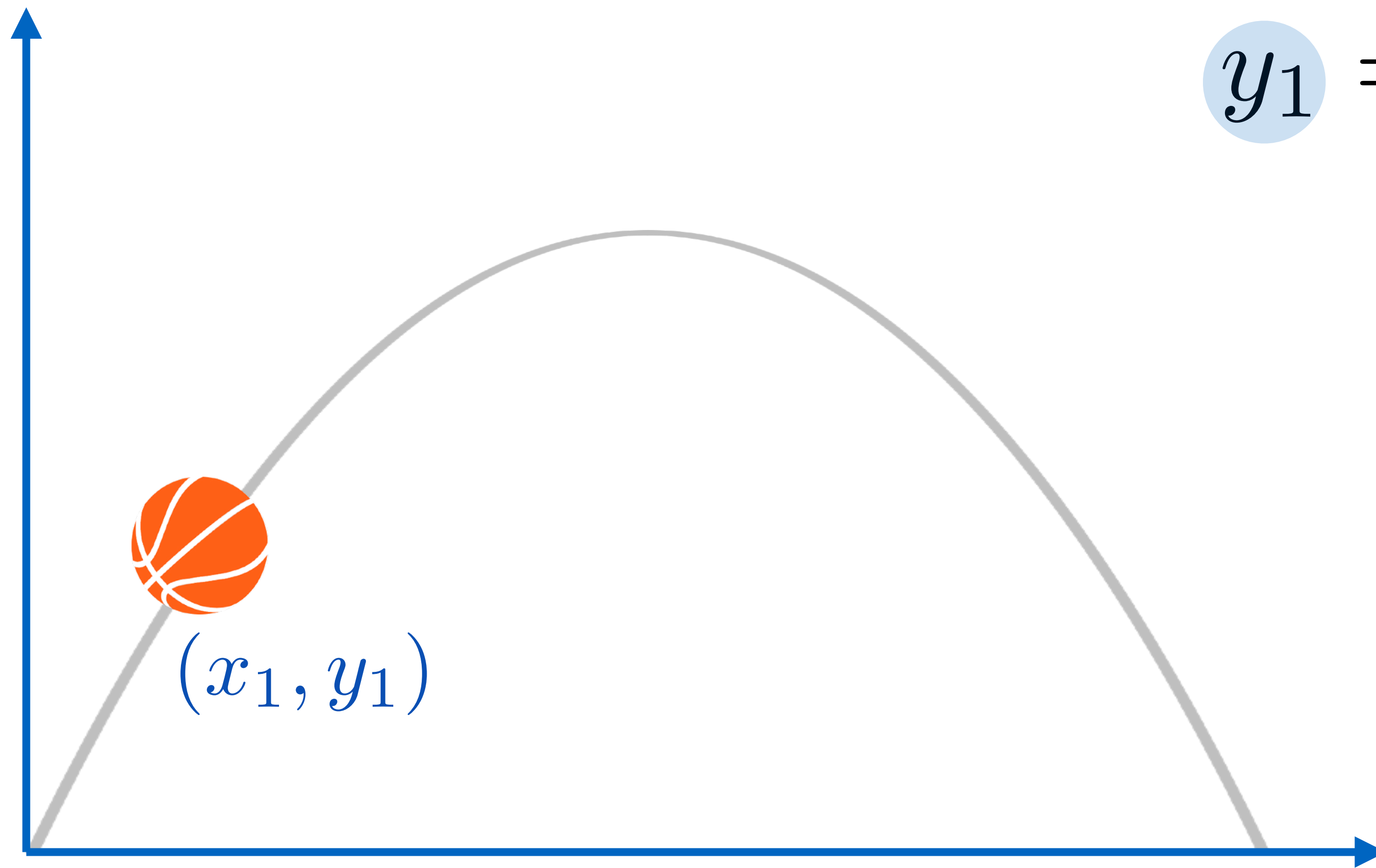
This kind of motion is well-understood! (**Ideal ballistic motion**)



$$y = ax^2 + bx + c$$

Ideal Ballistic Motion

Step 1: Think about observables and unknowns



$$y_1 = ax_1^2 + bx_1 + c$$

unknown

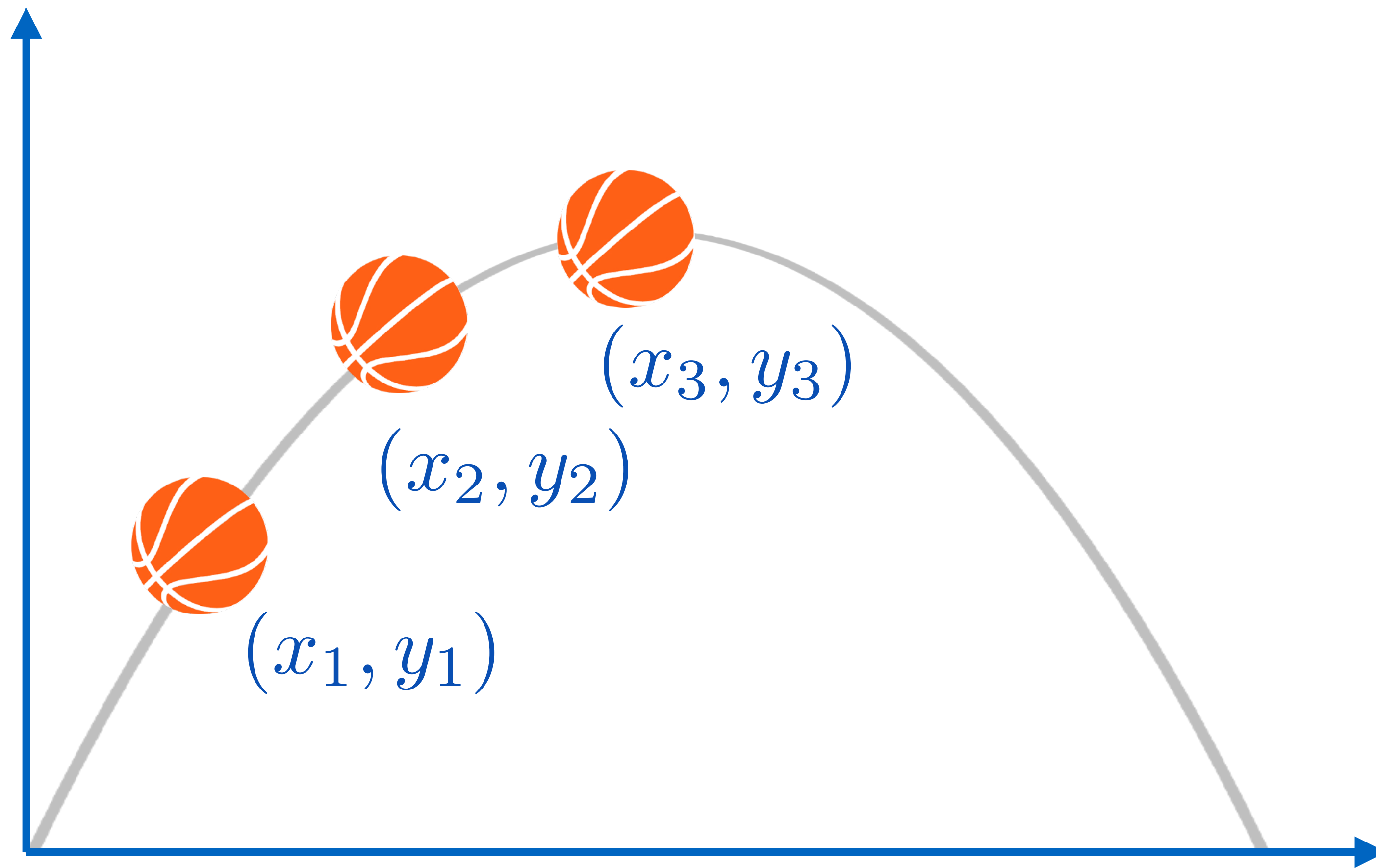
Regression

- We have a model of the **expected behaviour**.
- The model contains unknown parameter values.
- We have data / measurements that *may* be supported by the model.
- **Goal:** find parameters that best support the data.

- Can we express this as a solution to a linear system?
 - In that case, this is called **linear regression**.

Ideal Ballistic Motion

How many data points (i.e., *measurements*) do we need to solve this system?



The equation is **linear** in the unknowns.

$$y_1 = ax_1^2 + bx_1 + c$$

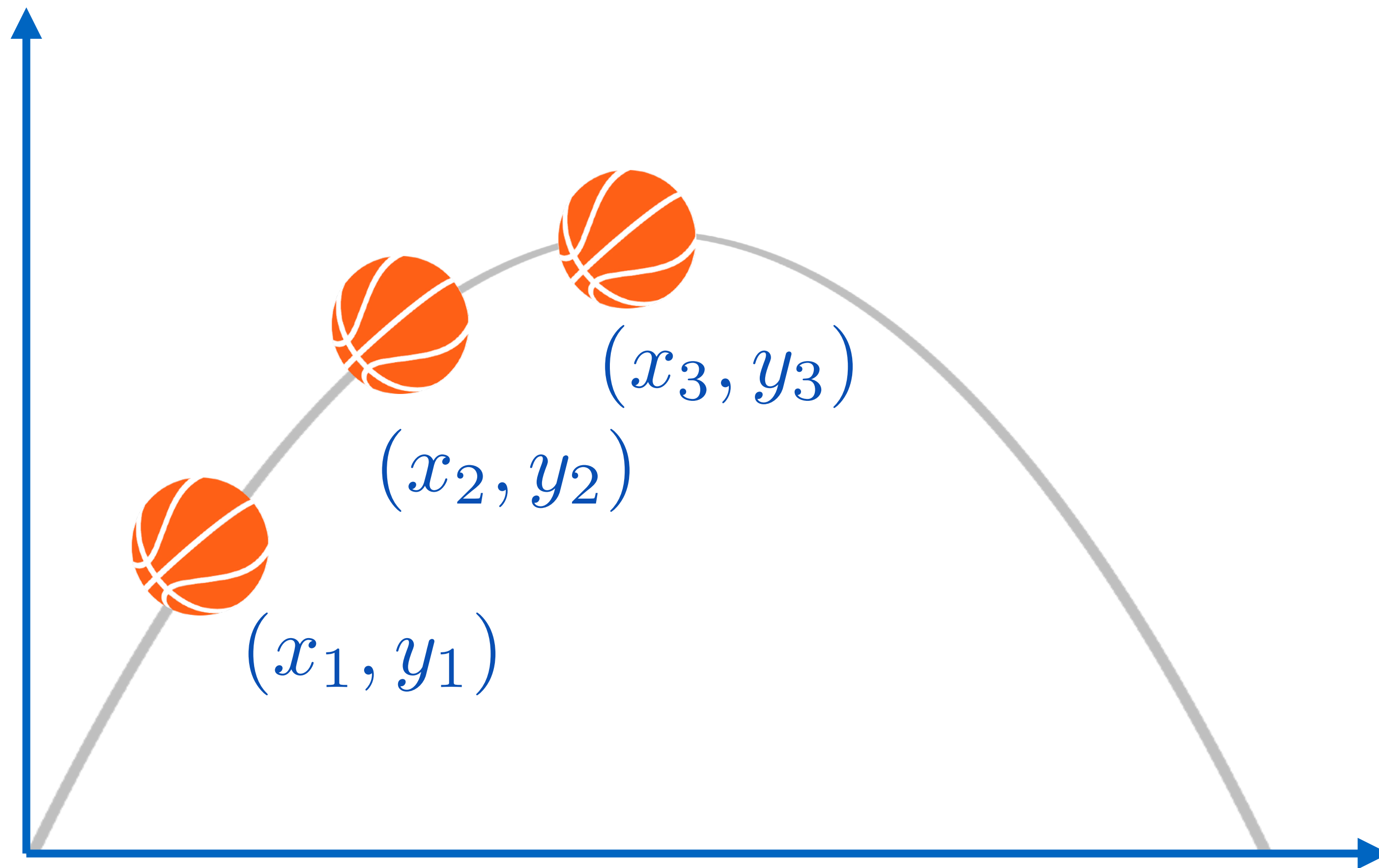
Arrows from the text above point to the terms ax_1^2 , bx_1 , and c in the equation.

$$y_2 = ax_2^2 + bx_2 + c$$

$$y_3 = ax_3^2 + bx_3 + c$$

Ideal Ballistic Motion

How many data points (i.e., *measurements*) do we need to solve this system?



The equation is **linear** in the unknowns.

$$y_1 = ax_1^2 + bx_1 + c$$

$$y_2 = ax_2^2 + bx_2 + c$$

$$y_3 = ax_3^2 + bx_3 + c$$

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

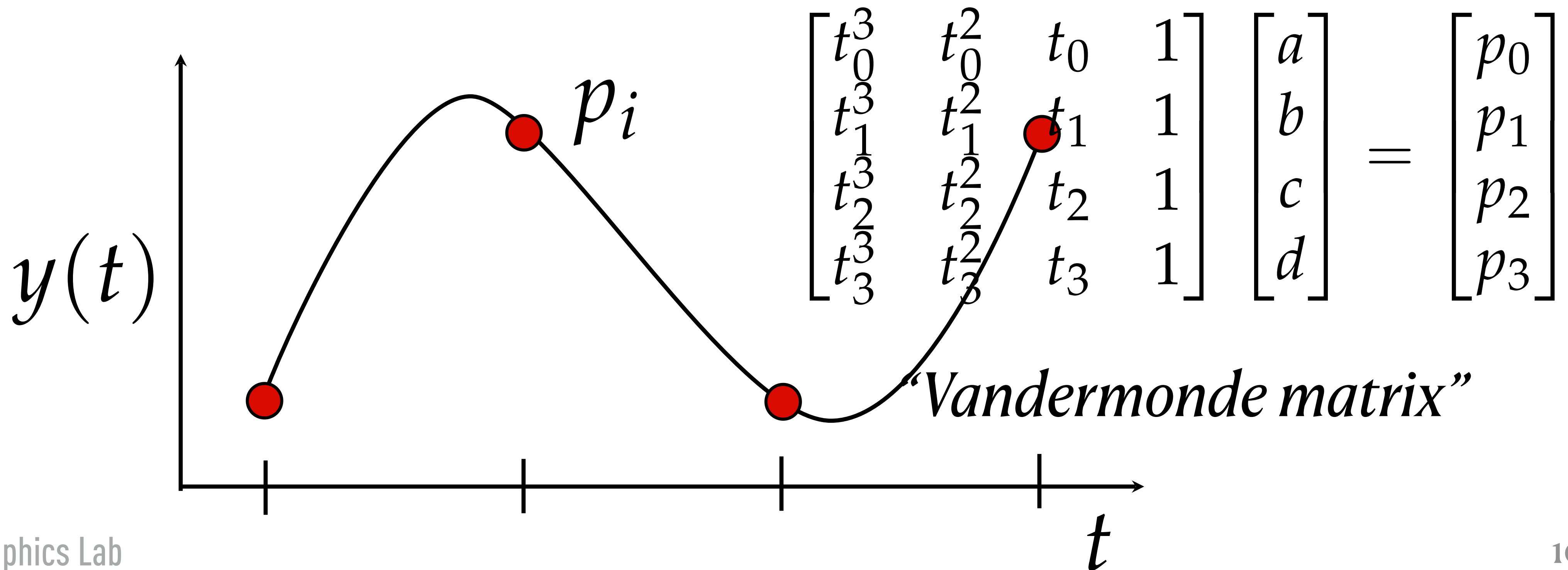
(identical, but using matrix-vector notation)

Polynomial regression

The basketball example is a representative of this larger class

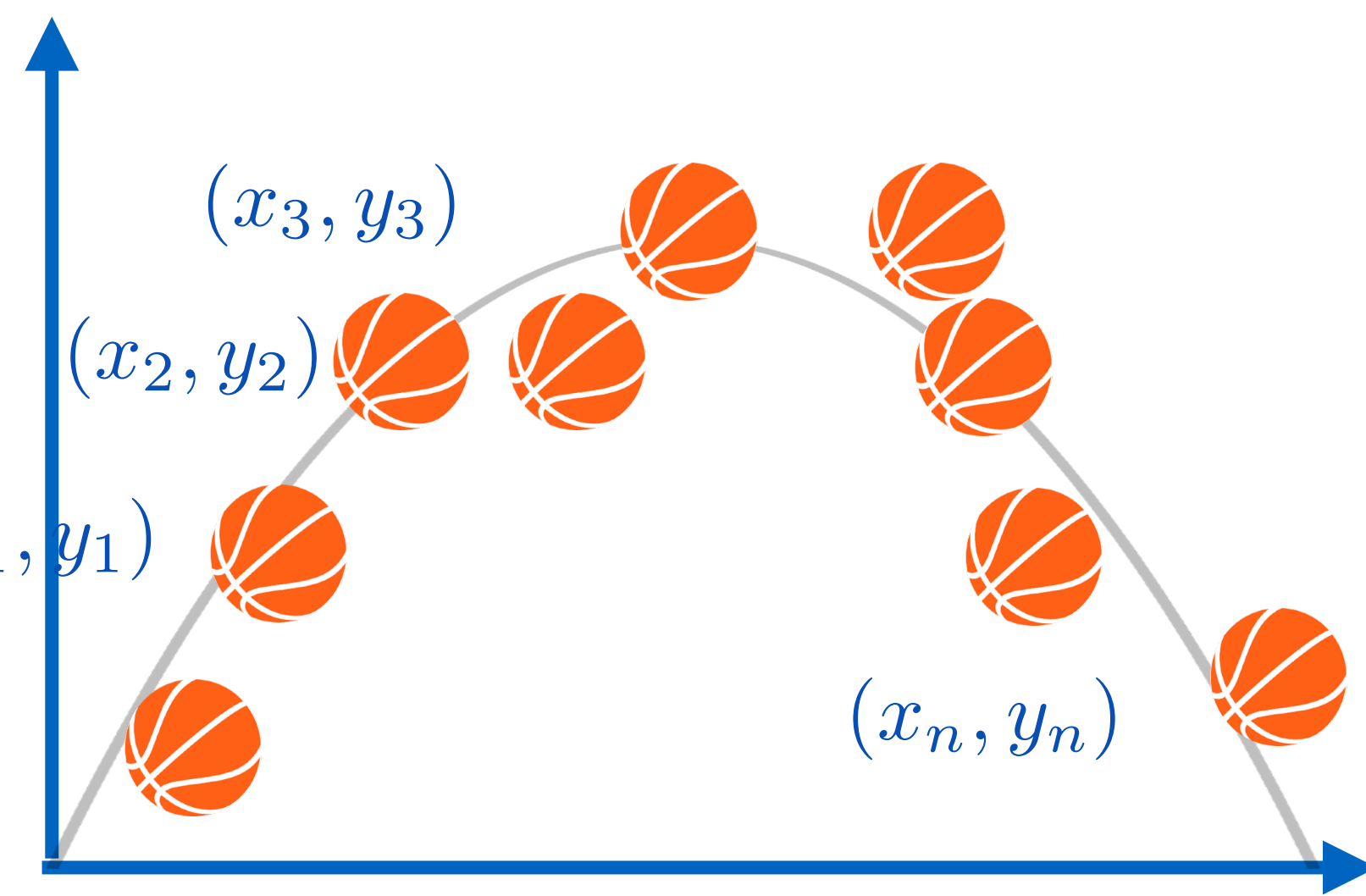
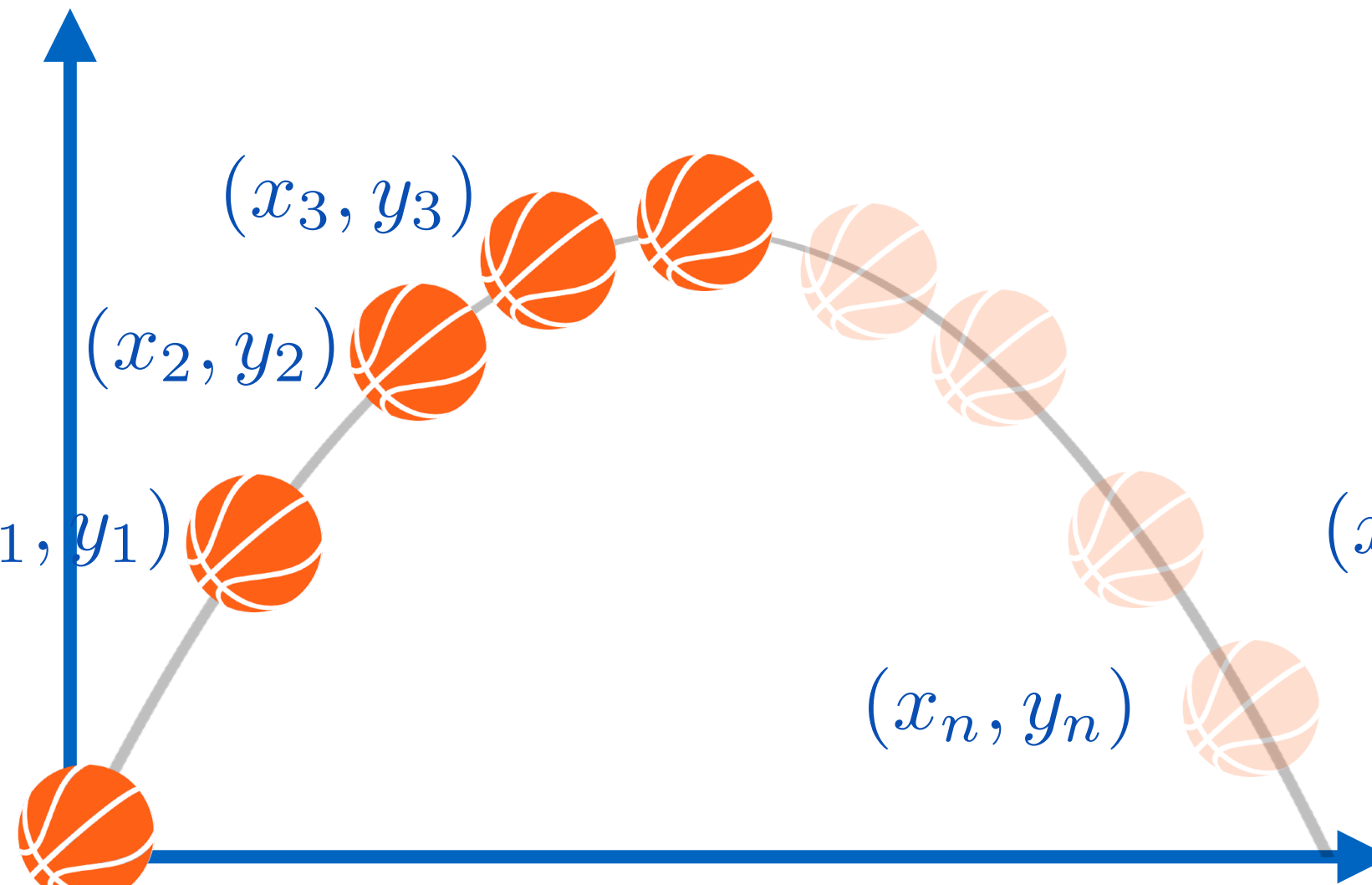
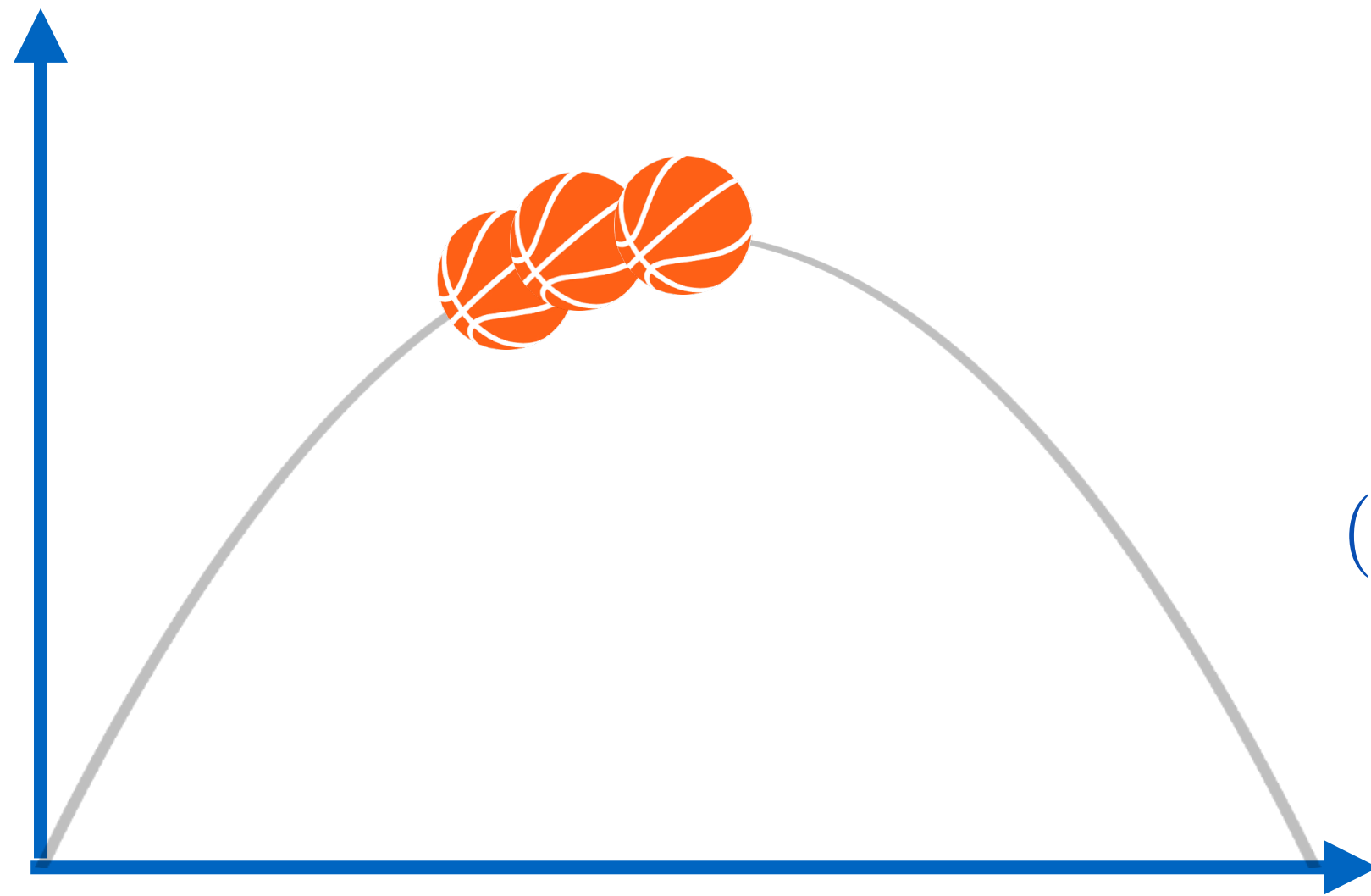
Find a polynomial that *perfectly interpolates* a given set of points so that

$$y(t_i) = p_i$$



Food for thought

What could possibly go wrong?



What if the observations are *weak*?

What if there are > 3 observations?

Use subset? Which ones? Use all somehow?

What if there is noise?

What if the model violates the assumptions?

Demo time

KABOOM!

A vibrant comic book style illustration. The central focus is the word "KABOOM!" written in a bold, black-outlined, bubbly font. The letters are filled with a red and yellow halftone dot pattern. This text is contained within a white, cloud-like speech bubble with a thick black outline. The background consists of radiating yellow and orange beams, creating a sense of explosion or energy. The overall color palette is dominated by red, yellow, and orange, with a halftone dot pattern throughout.

Another classical example of a "bad" linear system

Deconvolution: removing blur from a signal (e.g., an image)



Blurry



Original

Another classical example of a "bad" linear system

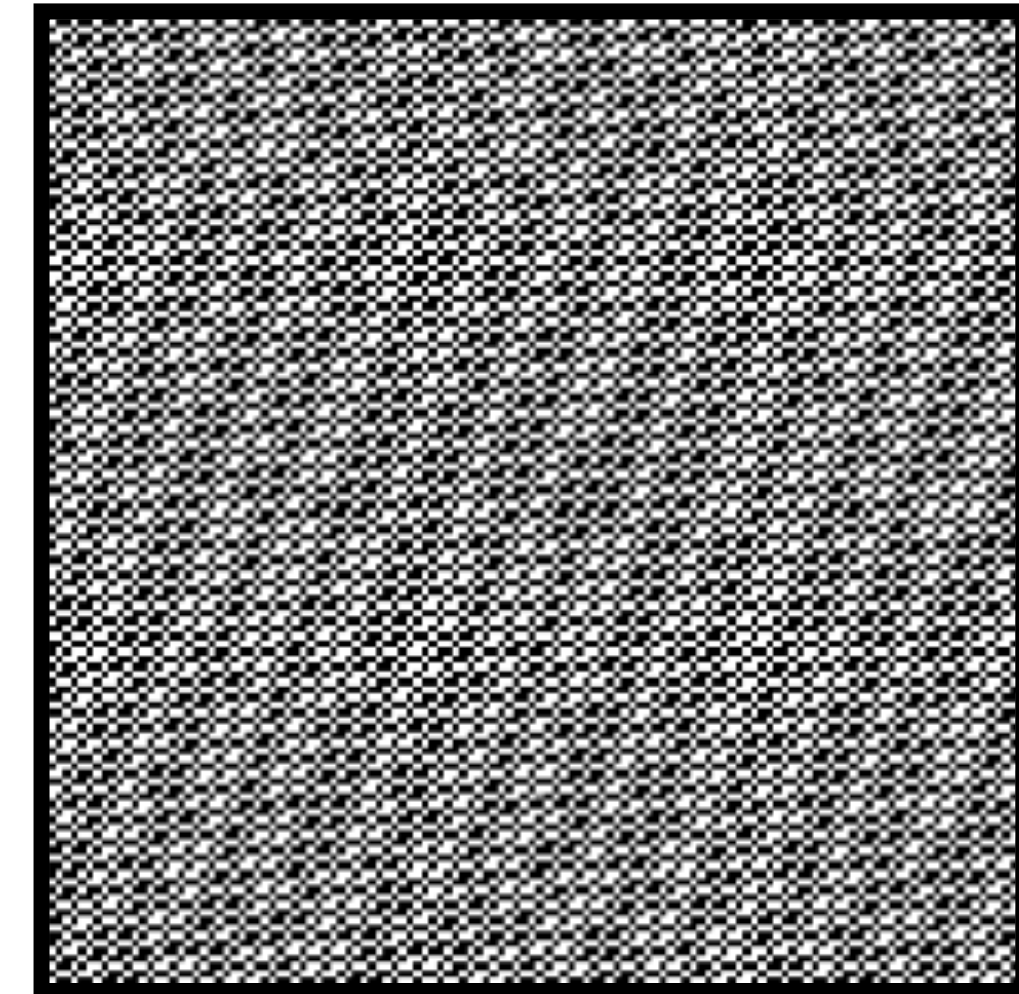
Original



Blurry



Recovered

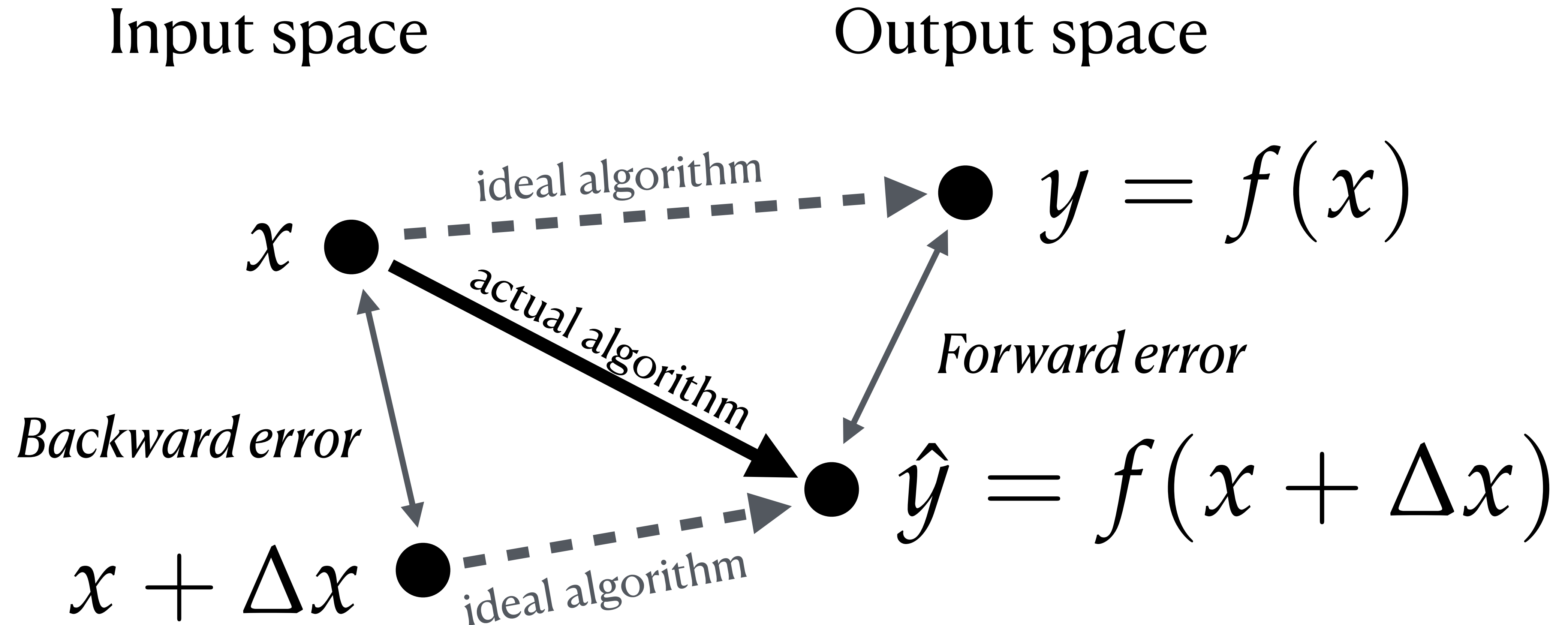


Regularized

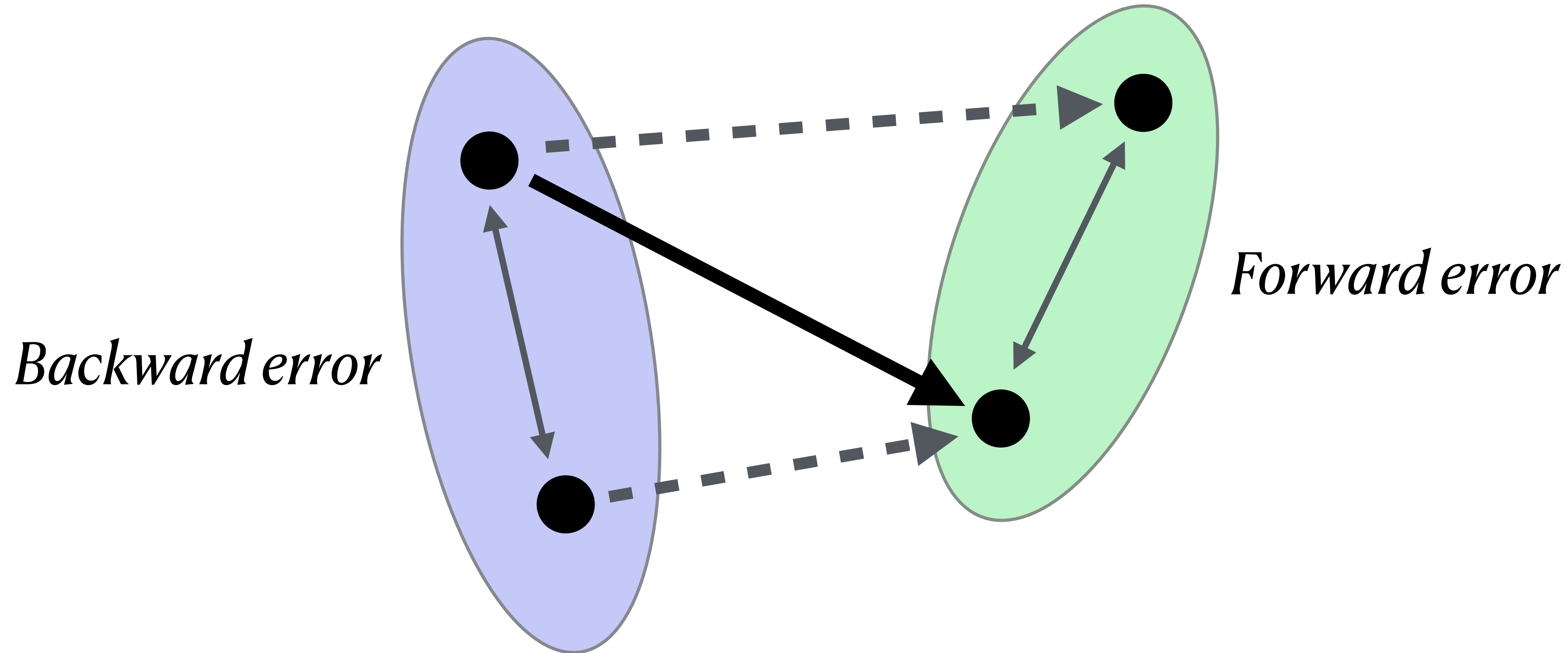


Forward and Backward Error

Recap



Conditioning of linear systems



Condition number: ratio $\frac{\text{forward error}}{\text{backward error}}$

Forward and Backward error

.. applied to linear system solving

Theoretical infinite precision solver

$$\mathbf{x} = f(\mathbf{b}) = \mathbf{A}^{-1}\mathbf{b}$$

Its inverse is simple!

$$f^{-1}(\mathbf{x}) = \mathbf{A}\mathbf{x}$$

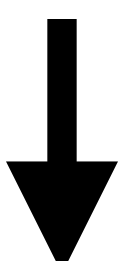
Actual finite precision solver

$$\hat{\mathbf{x}} = \hat{f}(\mathbf{b})$$

.. natural to expect that

$$f^{-1}(\hat{\mathbf{x}}) = \mathbf{A}\hat{\mathbf{x}} \approx \mathbf{b}$$

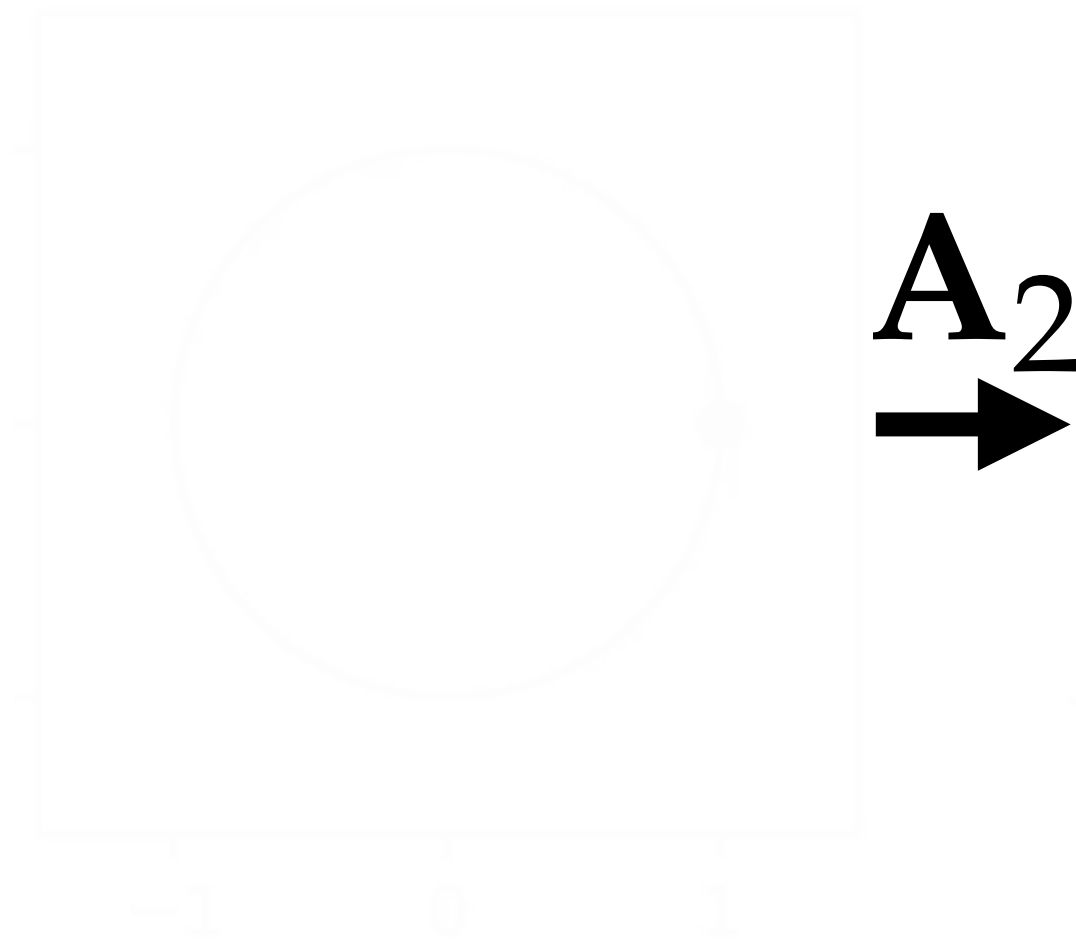
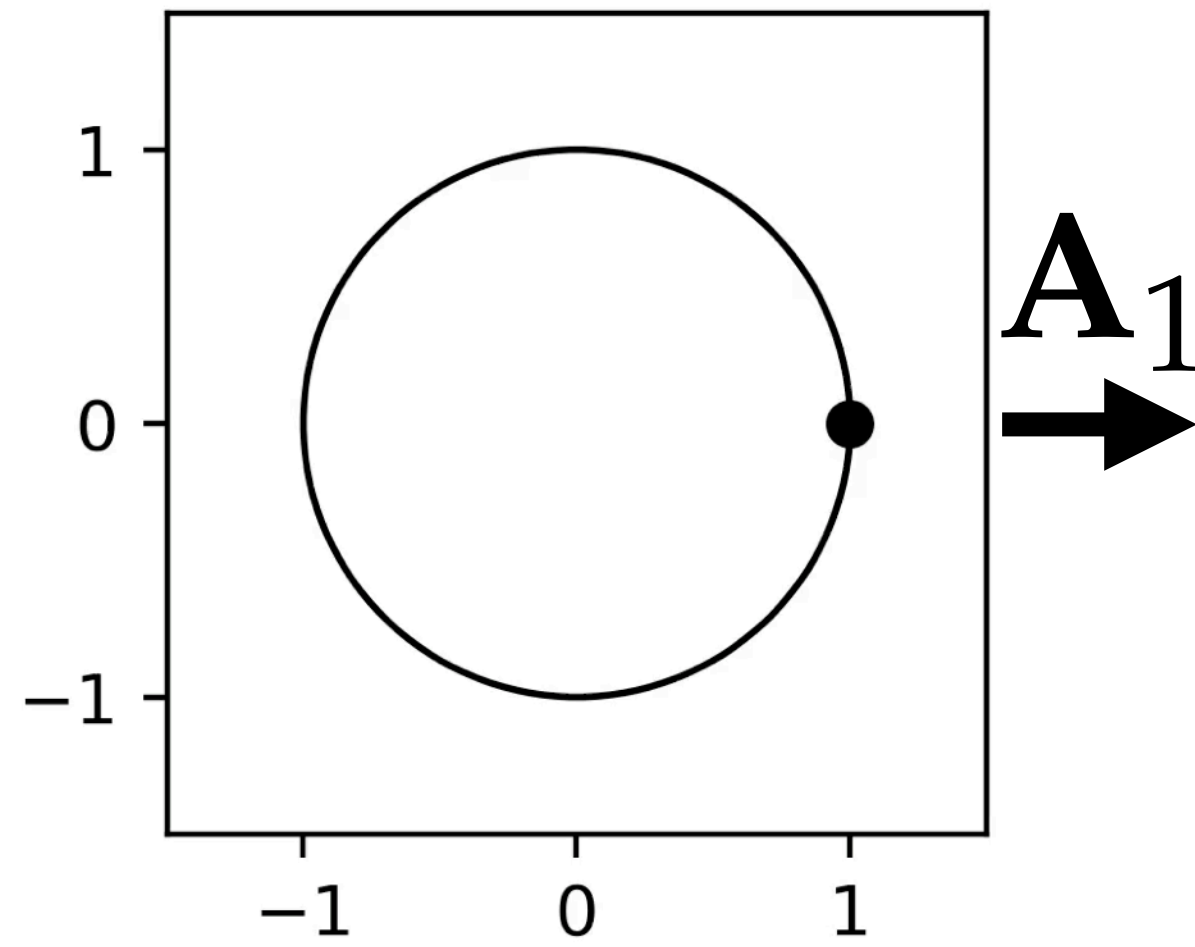
.. but how accurate is $\hat{\mathbf{x}}$?


$$r(\hat{\mathbf{x}}) = \mathbf{A}\hat{\mathbf{x}} - \mathbf{b}$$

The *residual* gives an approximation of the backward error of a linear system solver.

Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

Four randomly generated matrices — how do they transform points on a circle?



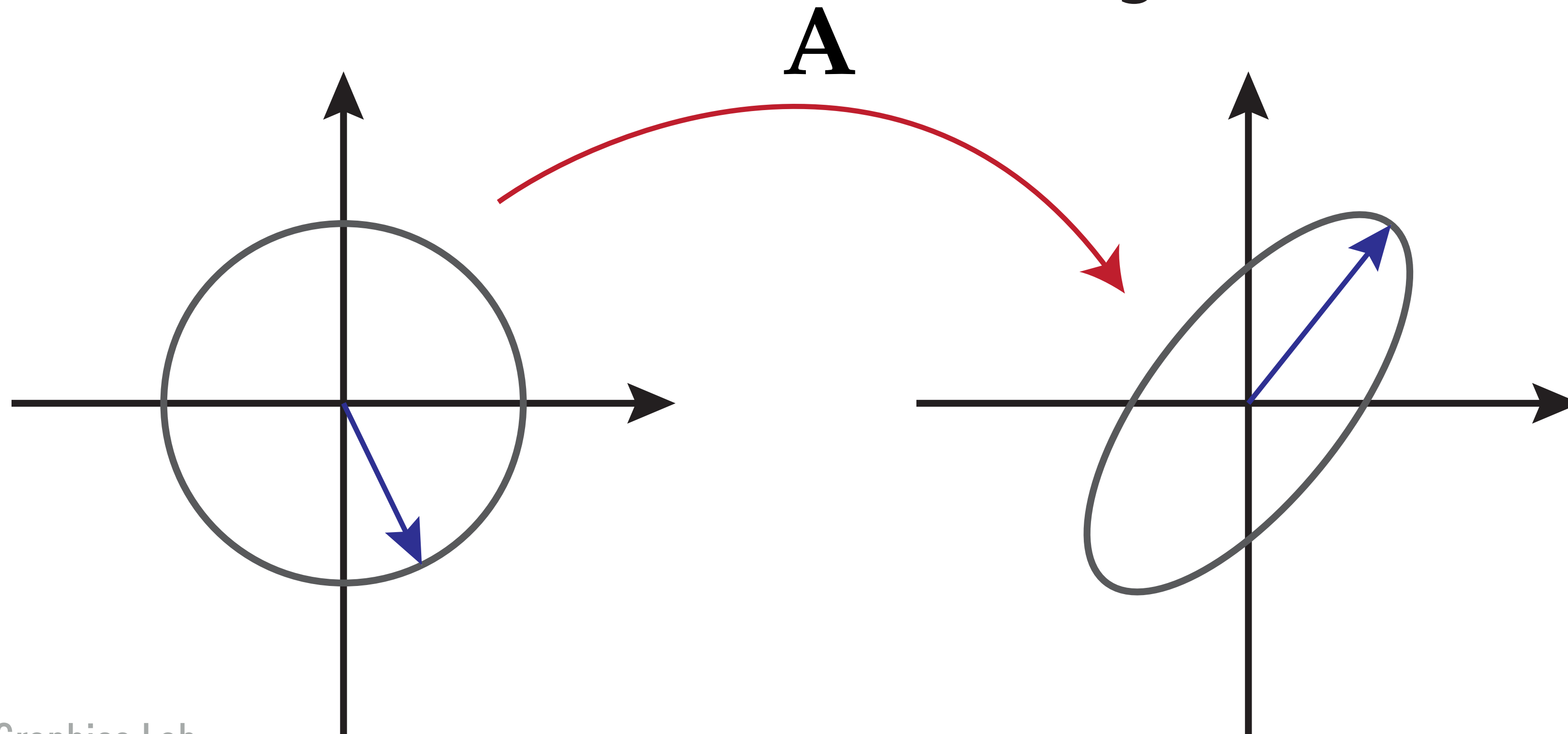
The matrix norm (aka. operator norm)

Determines the maximum radius of this ellipse

$$\|A\| = \max\{\|Ax\|, \text{ where } \|x\| = 1\}$$

Source domain

Target domain



Some inequalities

$$\|Ax\| \leq \|A\| \|x\|$$

Vector norm Matrix norm Vector norm

"The length of a transformed vector is bounded by its original length and the most that it can be stretched by A"

$$\|AB\| \leq \|A\| \|B\|$$

Matrix norm Matrix norm Matrix norm

"Successive application of A and B cannot stretch a vector by more than the product of their individual bounds."

Solving a perturbed linear system

Original system

$$\mathbf{Ax} = \mathbf{b}$$

Perturbed system

$$\mathbf{A}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} + \Delta\mathbf{b}$$

What we'd like to know:

how big is $\|\Delta\mathbf{x}\| / \|\mathbf{x}\|$? (rel. error of solution)

Ingredient 1:

$$\|\mathbf{b}\| = \|\mathbf{Ax}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\| \Leftrightarrow \|\mathbf{x}\| \geq \frac{\|\mathbf{b}\|}{\|\mathbf{A}\|}$$

Ingredient 2:

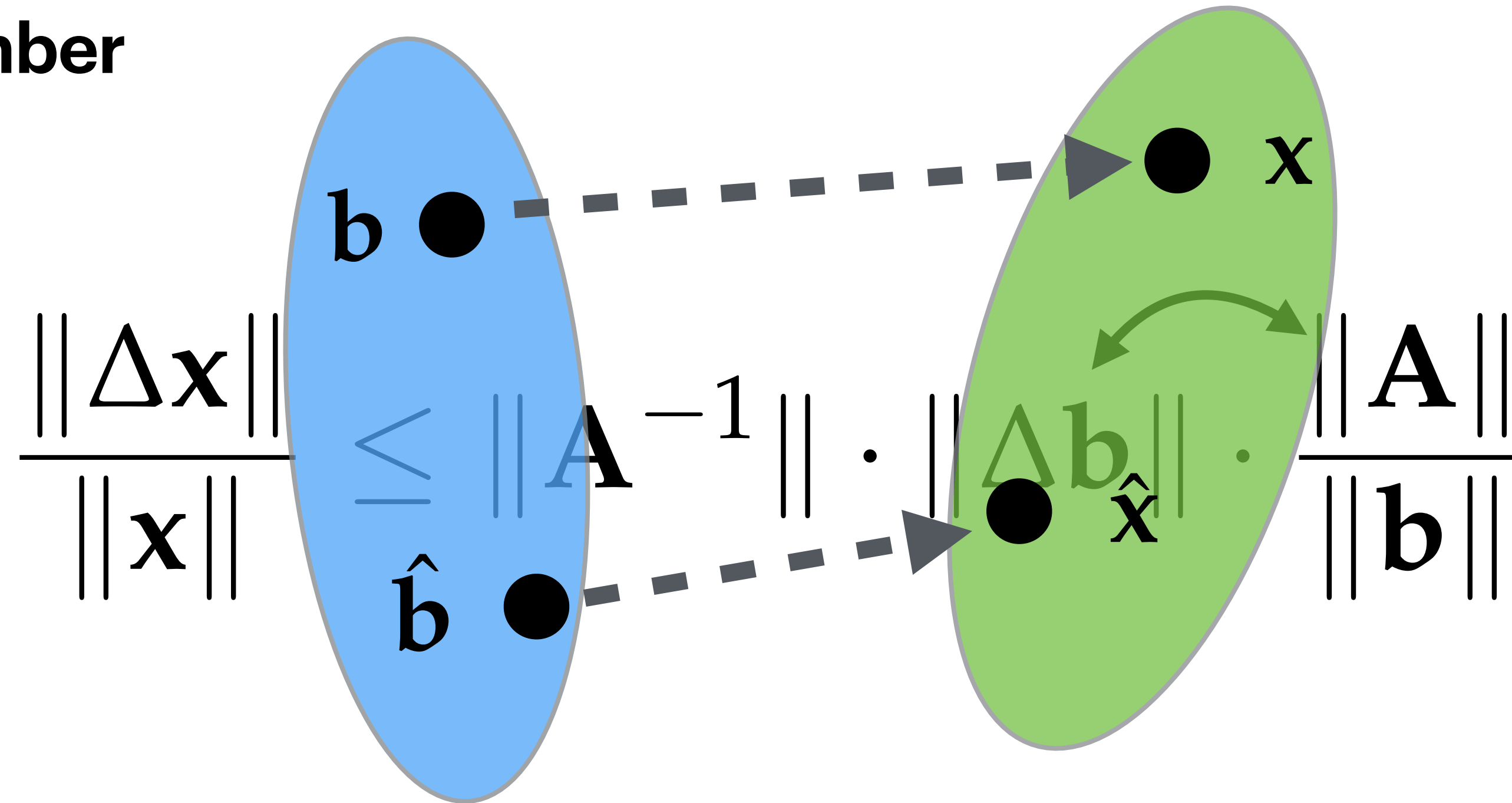
$$\|\Delta\mathbf{x}\| = \|\mathbf{A}^{-1}\Delta\mathbf{b}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{b}\|$$

Putting it together:

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{b}\| \cdot \frac{\|\mathbf{A}\|}{\|\mathbf{b}\|}$$

Solving a perturbed linear system

The condition number

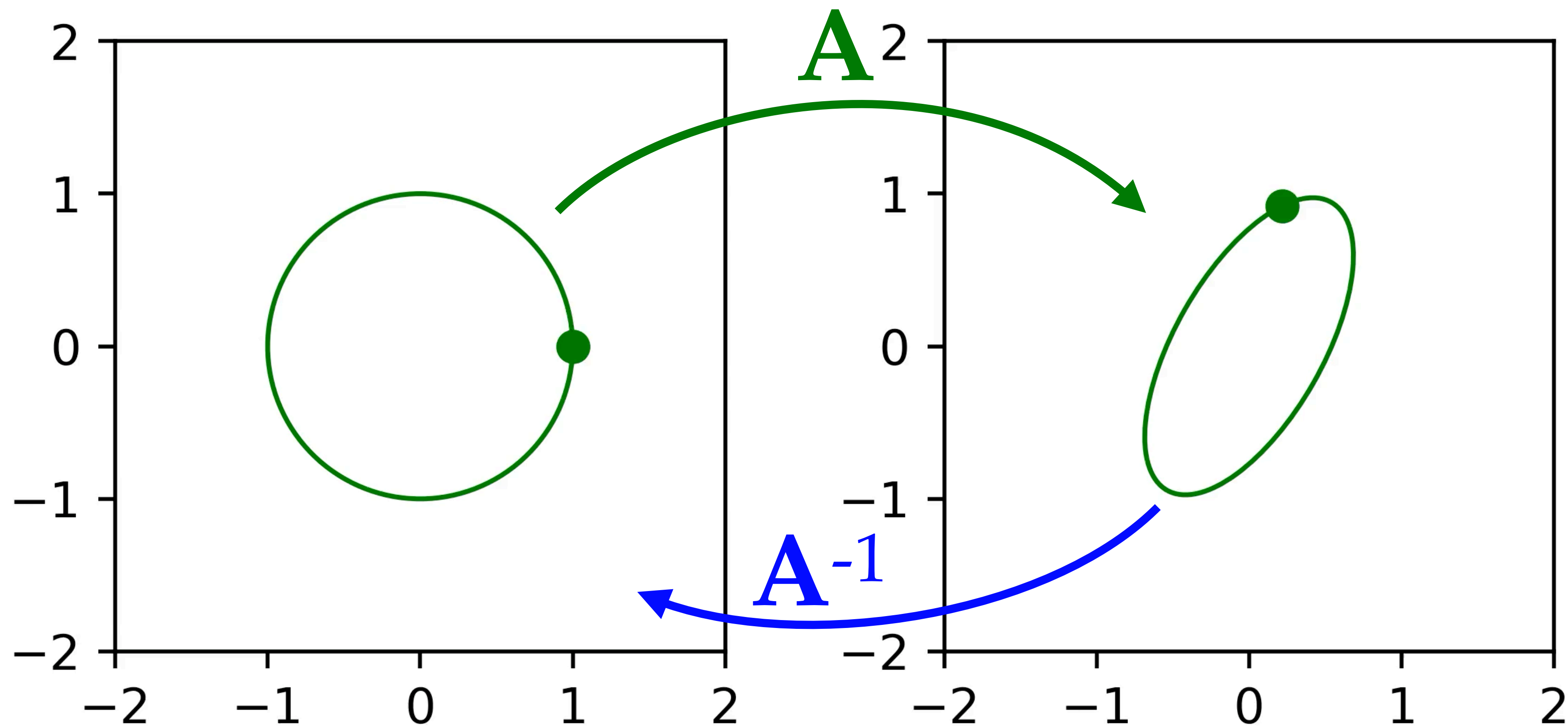


$$\frac{\|\Delta x\|}{\|x\|} \leq \underbrace{\|A^{-1}\| \cdot \|A\|}_{= \text{cond}(A) = \kappa(A)} \cdot \frac{\|\Delta b\|}{\|b\|}$$

The condition number

What does this number measure?

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$



The condition number

Some useful properties

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$

- $\text{cond}(\mathbf{A}) = \infty$ iff \mathbf{A} is singular
- $\text{cond}(\mathbf{A}) = 1$ if \mathbf{A} is the identity matrix
- $\text{cond}(\gamma\mathbf{A}) = \text{cond}(\mathbf{A}) \quad \gamma \in \mathbb{R}$
- $\text{cond}(\mathbf{Q}\mathbf{A}) = \text{cond}(\mathbf{A})$ where \mathbf{Q} is a rotation matrix

.. but what about the determinant?

The determinant is *not* a good way to determine whether a linear system is easy to solve.

- We know that $\det(\mathbf{A}) = 0$ *iff* \mathbf{A} is singular.
- Let's try to set $\mathbf{A} = \frac{1}{10} \mathbf{I} \in \mathbb{R}^{1000 \times 1000}$
- Then $\det(\mathbf{A}) = 10^{-1000}$. Yet, this linear system is *trivial* to solve.
- On the other hand, $\text{cond}(\mathbf{A}) = 1$ (!)

Implications

Stated without proof.

If you are using a good linear system solver, and if \mathbf{b} can be assumed to be accurate, then

$$\frac{\|\Delta \mathbf{x}\|}{\|\mathbf{x}\|} \lesssim \text{cond}(\mathbf{A}) \text{ulp}(1)$$

(rel. error of solution)

Where $\text{ulp}(1)$

$\approx 2^{-23}$ (*single precision*)

$\approx 2^{-52}$ (*double precision*)

A Chicken-and-Egg problem

- Inverting matrices is expensive.
- We want to know the condition number to find out if this is *even possible?*
- Our definition of $\text{cond}(\mathbf{A})$ requires inverting a matrix and computing a matrix norm (*how?!)*

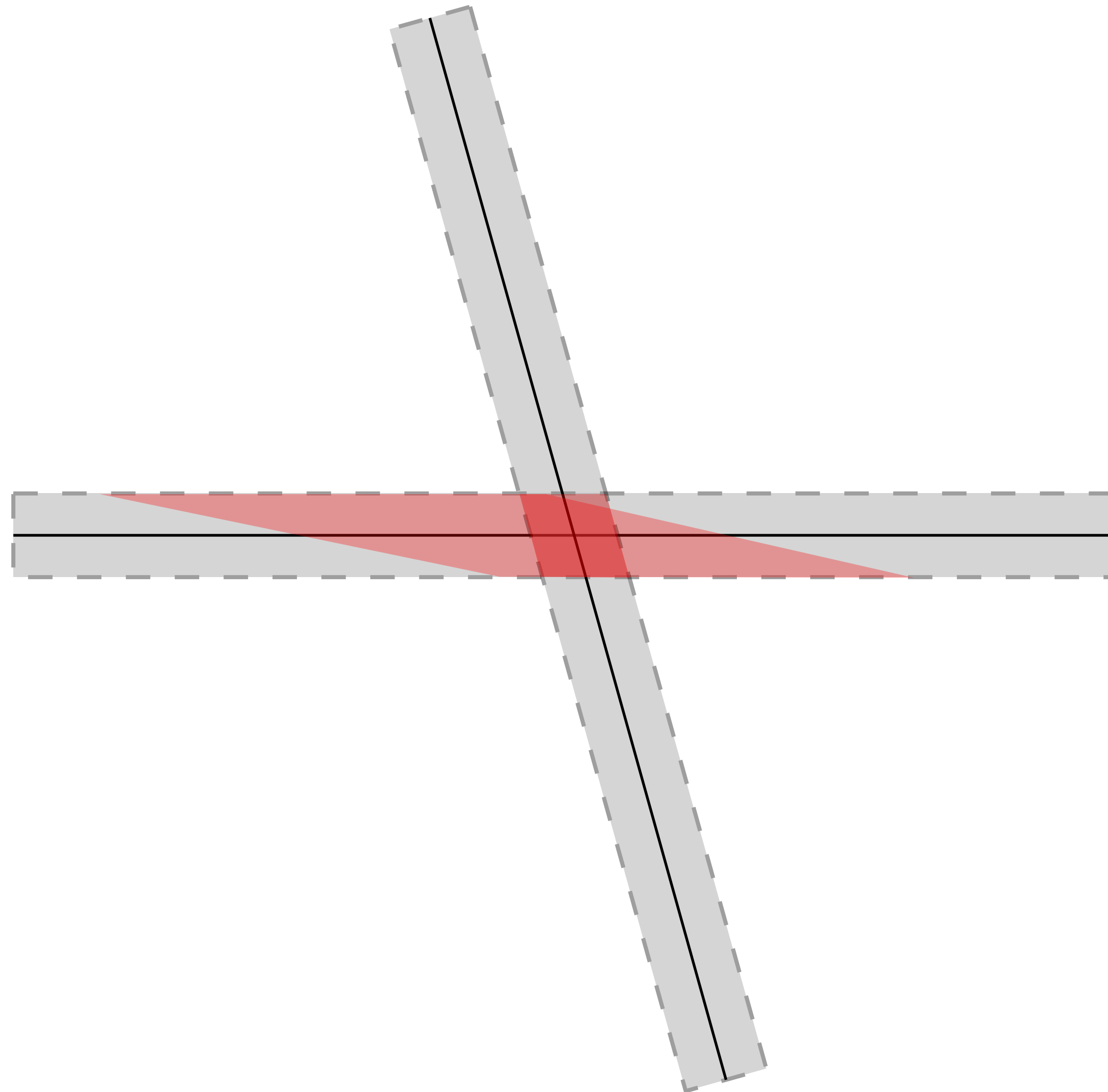


Midjourney: *chicken and egg problem*

Stay tuned!

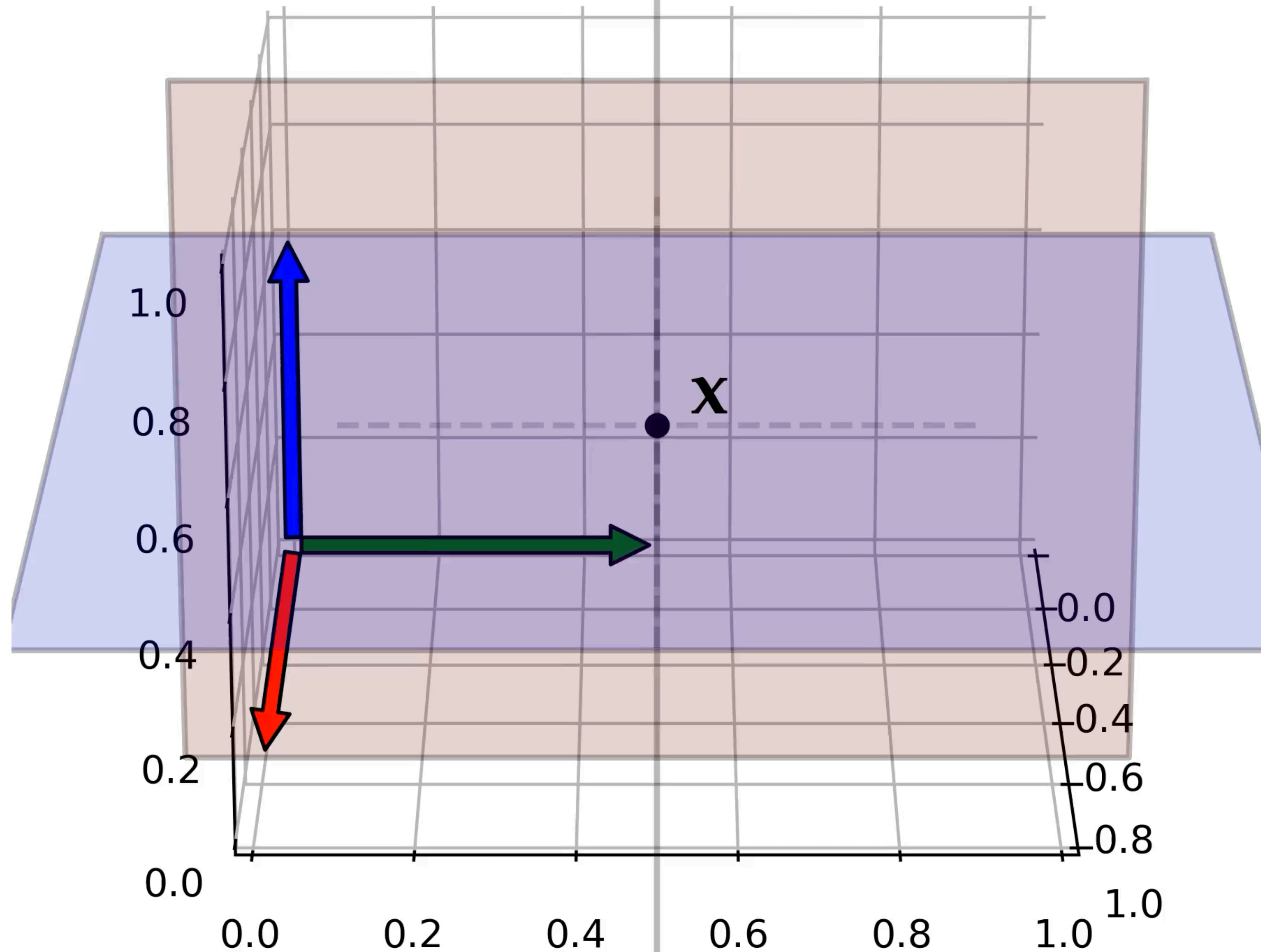
Geometry of badly conditioned systems

Noise & errors may be greatly magnified. Row-based interpretation.



Geometric interpretation of linear systems

Row-based view



$$\mathbf{A} = \begin{pmatrix} \text{---} & \mathbf{a}^{(1)} & \text{---} \\ \text{---} & \mathbf{a}^{(2)} & \text{---} \\ \text{---} & \mathbf{a}^{(3)} & \text{---} \end{pmatrix}$$

Interpretation: Compute the intersection of planes with normals $\mathbf{a}^{(i)}$ and offset b_i

$$\mathbf{Ax} = \mathbf{b}$$

$$\Leftrightarrow \begin{aligned} \mathbf{a}^{(1)} \cdot \mathbf{x} &= b_1 \\ \mathbf{a}^{(2)} \cdot \mathbf{x} &= b_2 \\ \mathbf{a}^{(3)} \cdot \mathbf{x} &= b_3 \end{aligned}$$

Conditioning of the Vandermonde Matrix

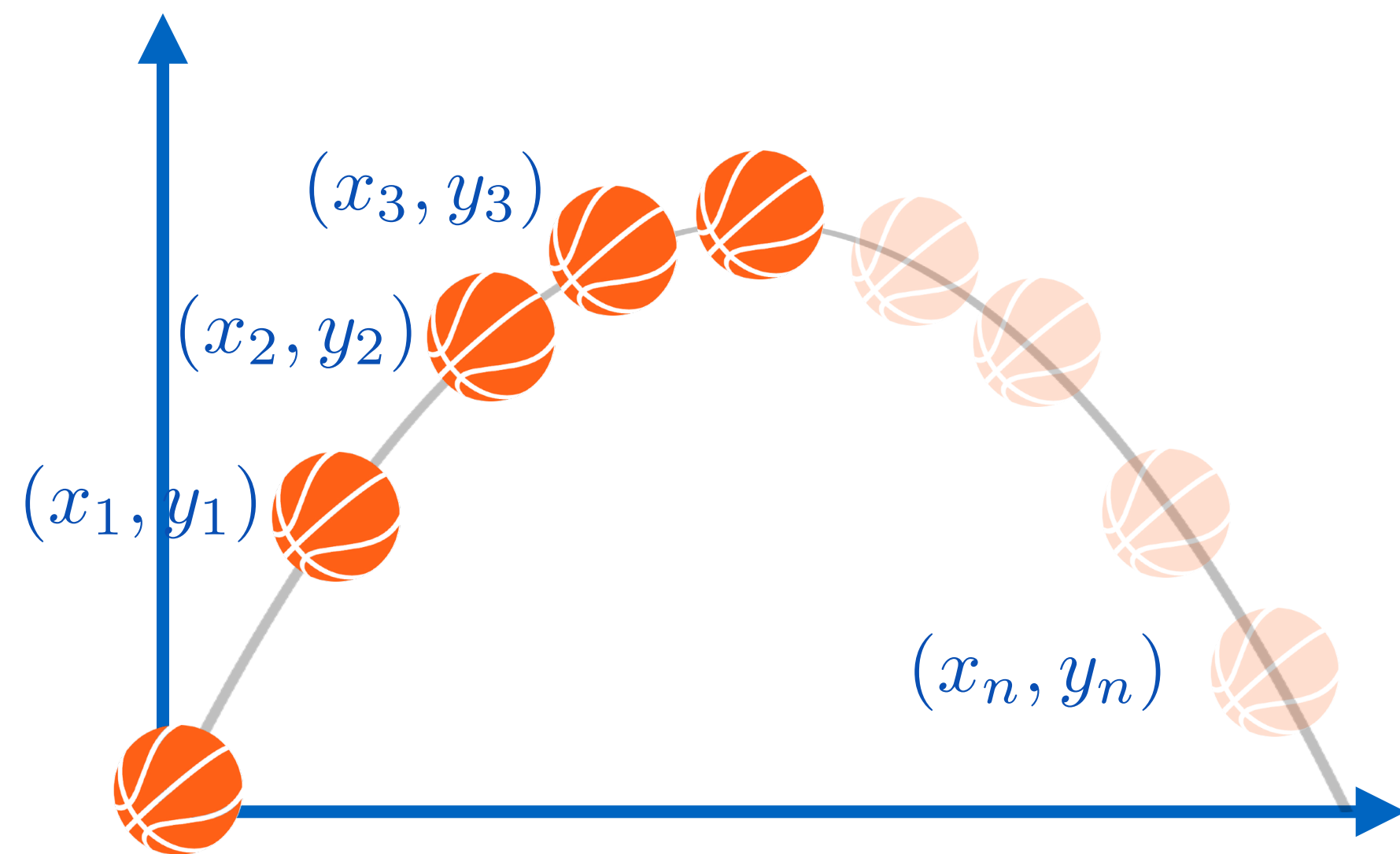
Maps from polynomial coefficients to polynomial evaluations.

$$\mathbf{A} = \begin{pmatrix} \alpha_1^n & \dots & \alpha_1^2 & \alpha_1 & 1 \\ \alpha_2^n & \dots & \alpha_2^2 & \alpha_2 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_3^n & \dots & \alpha_3^2 & \alpha_3 & 1 \\ \alpha_m^n & \dots & \alpha_m^2 & \alpha_m & 1 \end{pmatrix}$$

Condition number for $n = 100$ (for uniformly spaced positions on $[0, 1]$):

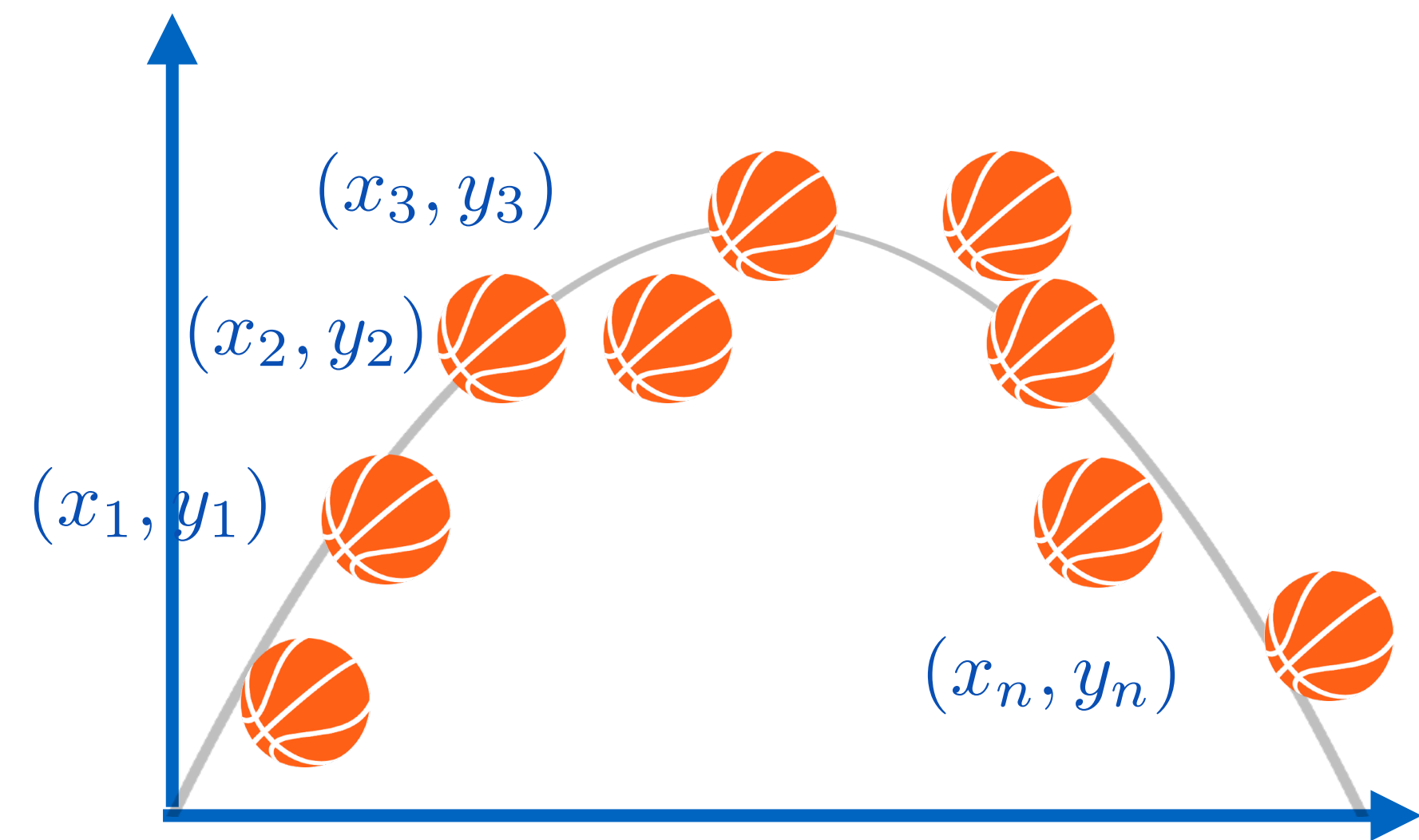
$$\text{cond}(\mathbf{A}) = 7.31 \cdot 10^{19} (!!!!!)$$

Linear Least Squares



What if there are > 3 observations?

Use subset? Which ones? Use all somehow?



What if there is noise?

What if the model violates the assumptions?

From linear systems to least squares

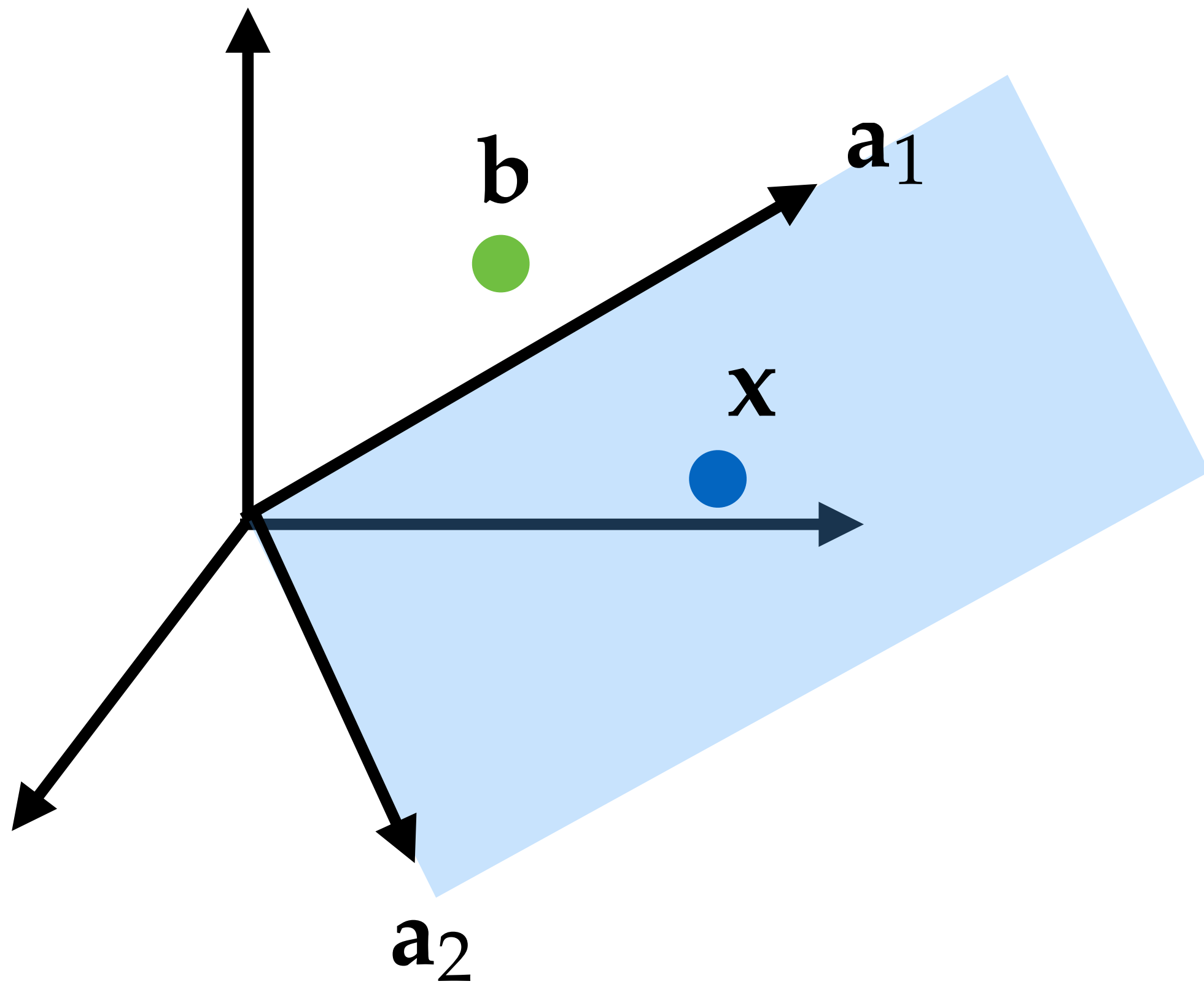
The diagram illustrates a linear system $Ax = b$ where the number of equations n is greater than the number of parameters m . The matrix A is shown as a vertical stack of rows, each representing an equation. The first four rows are explicitly labeled with coefficients a_1, a_2, a_3, a_4 , and the fifth row is labeled with an ellipsis \dots . The vector x is shown as a single column. The vector b is shown as a column of values b_1, b_2, b_3, b_4 , and an ellipsis \dots . The system is represented as $A \cdot x = b$. A vertical line is drawn through the vector b , and a horizontal line is drawn through the vector x , intersecting at the point where the least squares solution is found. The vector b is labeled with a large 0 to the right, indicating that the system is overdetermined and does not have an exact solution.

n constraint
equations (many)

m parameters
(few)

Geometric intuition

For almost all values of b , $Ax = b$ is not solvable. Try to get as close as possible.



$$A = \left(\begin{array}{c|c} & \\ \hline a_1 & a_2 \\ \hline & \end{array} \right) \left. \vphantom{\begin{array}{c|c} & \\ \hline a_1 & a_2 \\ \hline & \end{array}} \right\} 3 \text{ rows}$$

Compromise: minimize the residual

Want: $\|\mathbf{r}(\mathbf{x})\| = \|\mathbf{Ax} - \mathbf{b}\|$ to be as *small as possible*

Game plan:

1. Expand the residual equation
2. Take its derivative
3. Set its derivative to zero

Ingredient: the *squared* length of a vector is just the inner product with itself:

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = \langle \mathbf{Ax} - \mathbf{b}, \mathbf{Ax} - \mathbf{b} \rangle$$

Compromise: minimize the residual

Want: $\|\mathbf{r}(\mathbf{x})\| = \|\mathbf{Ax} - \mathbf{b}\|$ to be as *small as possible*

Game plan:

1. Expand the residual equation
2. Take its derivative
3. Set its derivative to zero

$$\begin{aligned}\|\mathbf{Ax} - \mathbf{b}\|^2 &= \langle \mathbf{Ax} - \mathbf{b}, \mathbf{Ax} - \mathbf{b} \rangle \\ &= \langle \mathbf{Ax}, \mathbf{Ax} \rangle - \langle \mathbf{Ax}, \mathbf{b} \rangle - \langle \mathbf{b}, \mathbf{Ax} \rangle + \langle \mathbf{b}, \mathbf{b} \rangle \\ &= \mathbf{x}^T \mathbf{A}^T \mathbf{Ax} - 2\mathbf{b}^T \mathbf{Ax} + \|\mathbf{b}\|^2\end{aligned}$$

Compromise: minimize the residual

Want: $\|\mathbf{r}(\mathbf{x})\| = \|\mathbf{Ax} - \mathbf{b}\|$ to be as *small as possible*

Game plan:

1. Expand the residual equation
2. Take its derivative
3. Set its derivative to zero

$$\frac{\partial}{\partial \mathbf{x}} \left[\mathbf{x}^T \mathbf{A}^T \mathbf{Ax} - 2\mathbf{b}^T \mathbf{Ax} + \|\mathbf{b}\|^2 \right]$$
$$= 2\mathbf{A}^T \mathbf{Ax} - 2\mathbf{A}^T \mathbf{b}$$

Compromise: minimize the residual

Want: $\|\mathbf{r}(\mathbf{x})\| = \|\mathbf{Ax} - \mathbf{b}\|$ to be as *small as possible*

Game plan:

1. Expand the residual equation
2. Take its derivative
3. Set its derivative to zero



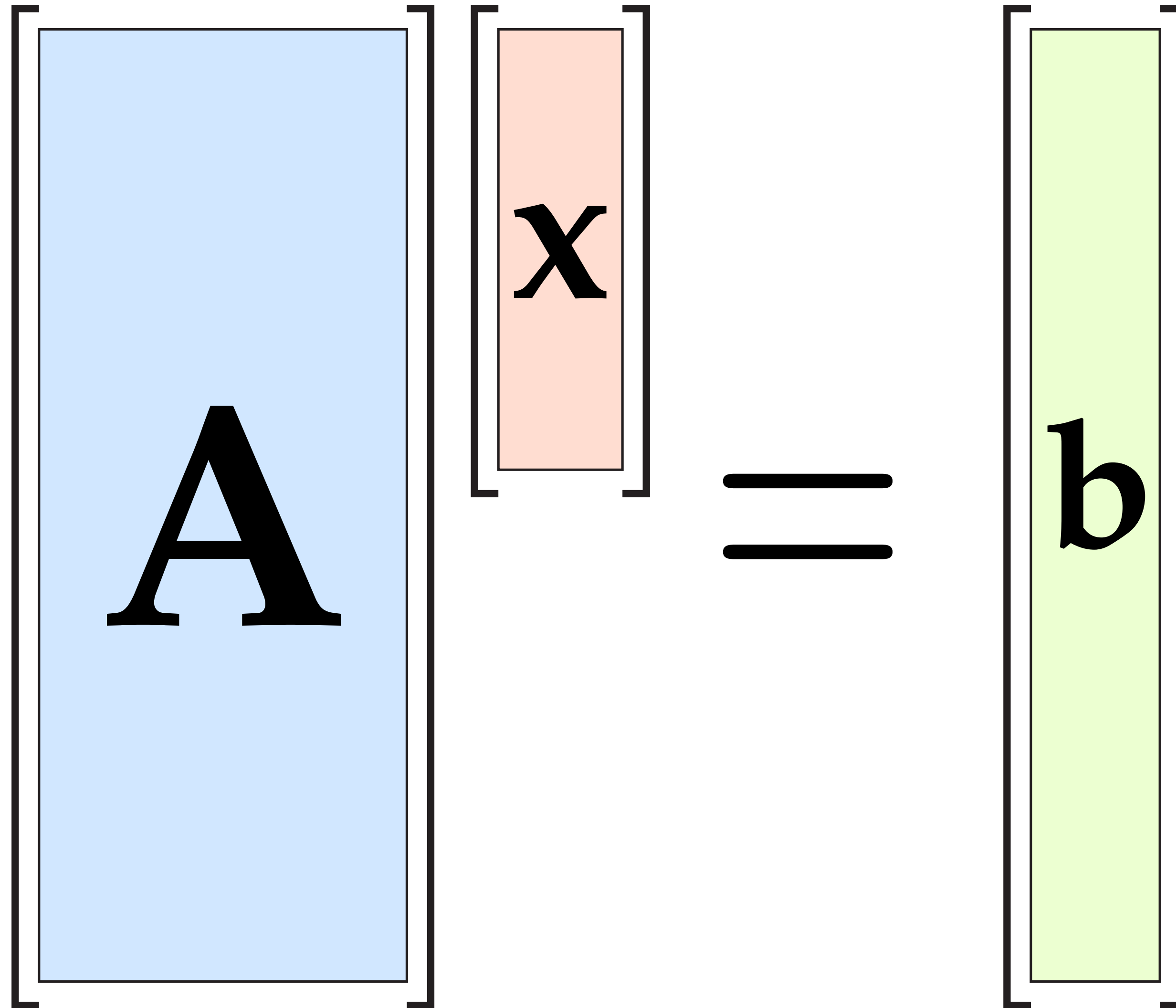
Carl Friedrich Gauss

$$\cancel{2}\mathbf{A}^T \mathbf{Ax} - \cancel{2}\mathbf{A}^T \mathbf{b} = 0$$

Normal Equations $\mathbf{A}^T \mathbf{b}$

Tall linear system

Generally ill-defined



The diagram illustrates a tall linear system $Ax = b$. It consists of three main components arranged horizontally: a large blue rectangle labeled A , a smaller orange rectangle labeled x , and a green rectangle labeled b . Each rectangle is enclosed in a black frame with a white border. The A rectangle is significantly taller than the x and b rectangles. An equals sign is positioned between the x and b rectangles, indicating the equation $Ax = b$.

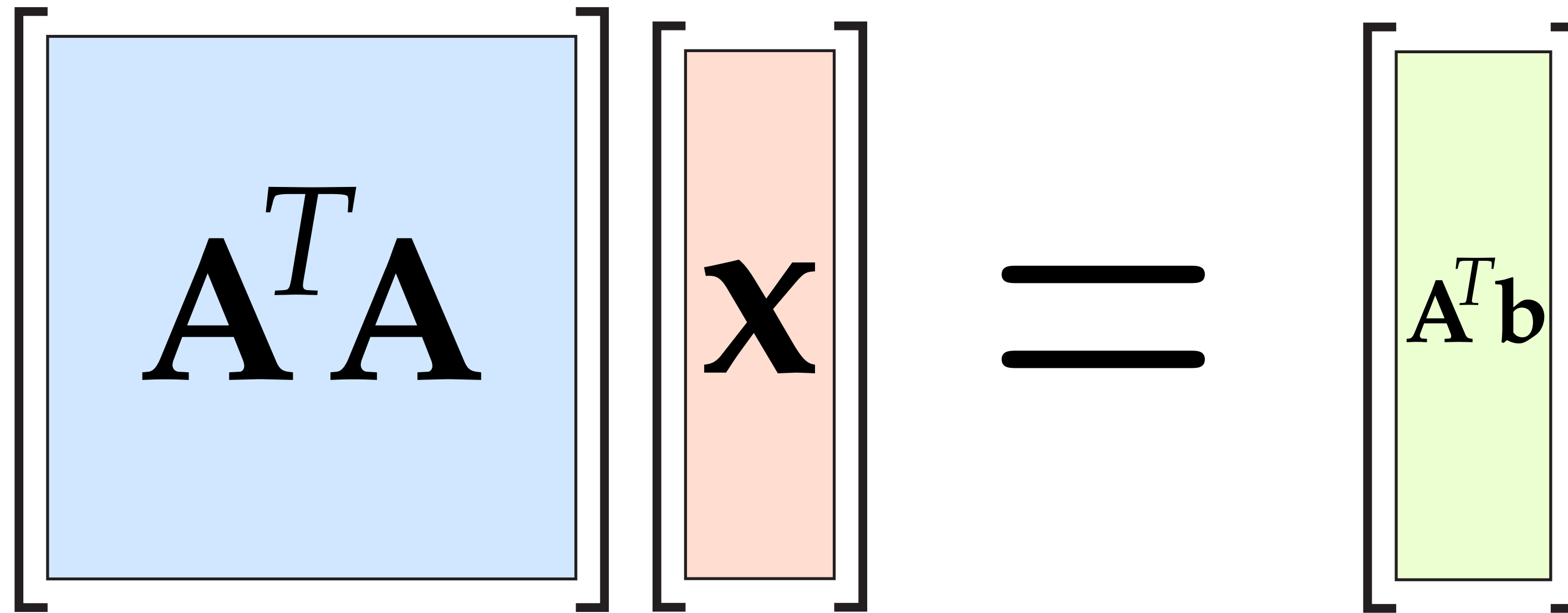
Linear least squares via the normal equations

The “best” solution (in the L_2 sense) is given by:

$$\begin{bmatrix} A^T \end{bmatrix} \begin{bmatrix} A \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} A^T \end{bmatrix} \begin{bmatrix} b \end{bmatrix}$$

Linear least squares via the normal equations

.. which is equivalent to a smaller square linear system

$$\begin{bmatrix} A^T A \end{bmatrix} \begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} A^T b \end{bmatrix}$$


Why are they called normal equations?

