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**

$$A = L\,U$$

3. **Conditioning & least squares basics**

$$\mathbf{A}^T\mathbf{A}\,\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

4. **Least squares, QR**

$$A = Q\,R$$

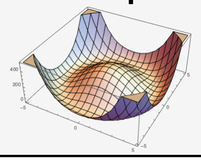
5. **Writing efficient numerical code**

```
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)
vcmpgt_oqps ymm6, ymm6, ymm7 ; Packed FP comparison -- c
```

6. **Eigenanalysis, SVD**

$$A = U\,\Sigma\,V^T$$

7. **Nonlinear problems**

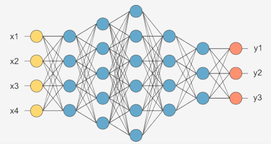


8. **Aut. differentiation**

PYTORCH

TensorFlow

9. **Neural networks**

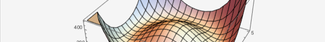


Where are we now?

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**

$$A = LU$$

3. **Conditioning & least squares basics**

$$A^TA\,x = A^Tb$$

4. **Least squares, QR**

$$A = QR$$

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)
vcmpgt_oqps ymm6, ymm6, ymm7 ; Packed FP comparison -- c
```

6. **Eigenanalysis, SVD**

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

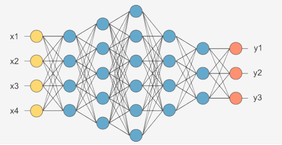
7. **Nonlinear problems**

8. **Aut. differentiation**

PYTORCH

TensorFlow

9. **Neural networks**



Solving a linear system using Gaussian Elimination

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

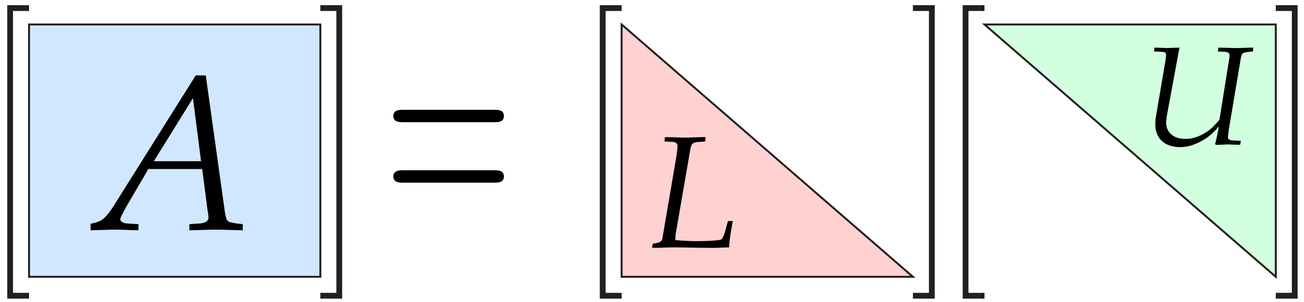
Observation: None of the decisions taken by the algorithm depend on the right hand side.

Solving a linear system using Gaussian Elimination

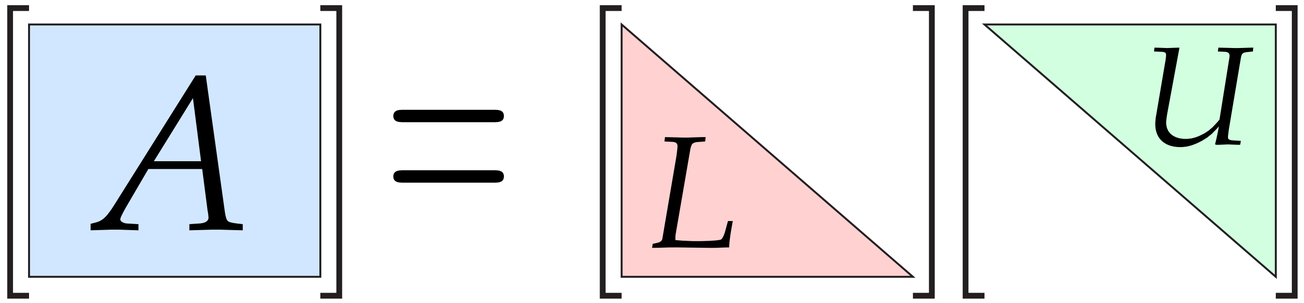
$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{b} \end{array}\right] = \left[\begin{array}{cccc|c} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array}\right]$$

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

Recap: LU factorization



Recap: LU factorization



$$\mathbf{U} = \underbrace{\mathbf{E}_6\mathbf{E}_5\mathbf{E}_4\mathbf{E}_3\mathbf{E}_2\mathbf{E}_1\mathbf{A}}$$

$$\mathbf{A} = \mathbf{LU}$$

$$\underbrace{(\mathbf{E}_6\mathbf{E}_5\mathbf{E}_4\mathbf{E}_3\mathbf{E}_2\mathbf{E}_1)^{-1}}$$

$$= \mathbf{E}_1^{-1}\mathbf{E}_2^{-1}\mathbf{E}_3^{-1}\mathbf{E}_4^{-1}\mathbf{E}_5^{-1}\mathbf{E}_6^{-1}$$

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

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

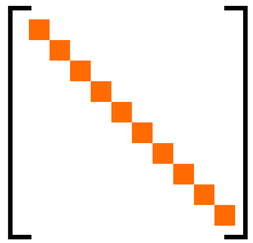
What kind of linear system solver?

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

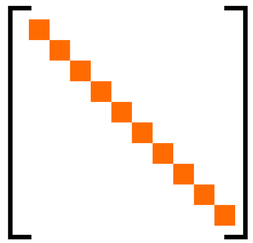
```
for i in range(n):
    x[i] = b[i] / A[i, i]
```

Recap: don't invert (big) matrices

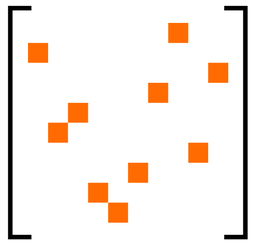
$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b}$$

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



Permutation

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

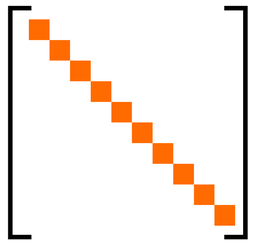
```
for i in range(n):
    x[i] = b[perm[i]]
```

Recap: don't invert (big) matrices

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

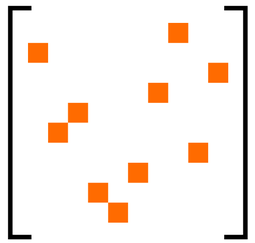
Usually this **does not mean** "compute an inverse of A". Read this as "call a linear system solver to solve $\mathbf{Ax} = \mathbf{b}$"

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

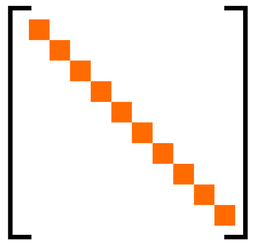
Forward/backward substitution

Recap: don't invert (big) matrices

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

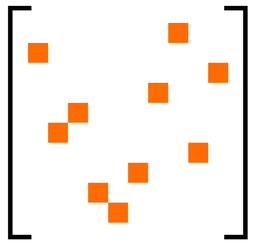
Usually this **does not mean** "compute an inverse of A". Read this as "call a linear system solver to solve $\mathbf{Ax} = \mathbf{b}$"

What kind of linear system solver?



Diagonal

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

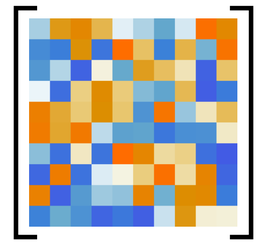


Permutation

```
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

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

A worked out example

LU factorization of a 2x2 matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \begin{matrix} \\ *\,\text{-}2 \end{matrix}$$

A worked out example

LU factorization of a 2x2 matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \begin{matrix} \\ *\,\text{-}2 \end{matrix}$$

(Elimination matrix that performs this operation)

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

A worked out example

LU factorization of a 2x2 matrix

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

(Elimination matrix that performs this operation)

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

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

A worked out example

LU factorization of a 2x2 matrix

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

(Elimination matrix that performs this operation)

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

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

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

A worked out example

LU factorization of a 2x2 matrix

(Elimination matrix that performs this operation)

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix} \quad * \text{-}2$$

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

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

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

$$\longrightarrow \mathbf{A} = \mathbf{LU}$$ ✅

Cost of matrix operations

1. Matrix-vector multiplication

Cost of matrix operations

1. Matrix-vector multiplication

```
b = A @ x
```

Cost of matrix operations

1. Matrix-vector multiplication

```
b = A @ x
```

```
b = np.zeros(n)
for i in range(n):
    b[i] = A[i, :] @ x
```

Cost of matrix operations

1. Matrix-vector multiplication

```
b = A @ x
```

```
b = np.zeros(n)
for i in range(n):
    b[i] = A[i, :] @ x
```

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

Cost of matrix operations

1. Matrix-vector multiplication

```
b = A @ x
```

```
b = np.zeros(n)
for i in range(n):
    b[i] = A[i, :] @ x
```

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

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

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
```

```
b = np.zeros(n)
for i in range(n):
    b[i] = A[i, :] @ x
```

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

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

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):
    b[i] = A[i, :] @ x
```

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

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

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):
    b[i] = A[i, :] @ x
```

66 *ms* ~6x slower

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

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

Cost of matrix operations

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

100 million matrix entries!

1. Matrix-vector multiplication

```
b = A @ x
```

11 *ms* (measured on an M1 Mac)

```
b = np.zeros(n)
for i in range(n):
    b[i] = A[i, :] @ x
```

66 *ms* ~6x slower

```
b = np.zeros(n)
for j in range(n):
    b += A[:, j] * x[j]
```

309 *ms* ~28x slower

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

Cost of matrix operations

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

100 million matrix entries!

1. Matrix-vector multiplication

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

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

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

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

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):
    b[i] = A[i, :] @ x
```
66 *ms* ~6x slower

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

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

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

Cost of matrix operations

2. Matrix-matrix multiplication

Cost of matrix operations

2. Matrix-matrix multiplication

```
C = A @ B
```

Cost of matrix operations

2. Matrix-matrix multiplication

```
C = A @ B
```

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

Cost of matrix operations

2. Matrix-matrix multiplication

```
C = A @ B
```

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

```
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]
```

Cost of matrix operations

2. Matrix-matrix multiplication

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

1 million matrix entries!

```
C = A @ B
```

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

```
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]
```

Cost of matrix operations

2. Matrix-matrix multiplication

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

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]
```

```
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]
```

Cost of matrix operations

2. Matrix-matrix multiplication

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

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]
```

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]
```

Cost of matrix operations

2. Matrix-matrix multiplication

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

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]
```

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

2. Matrix-matrix multiplication

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

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]
```

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

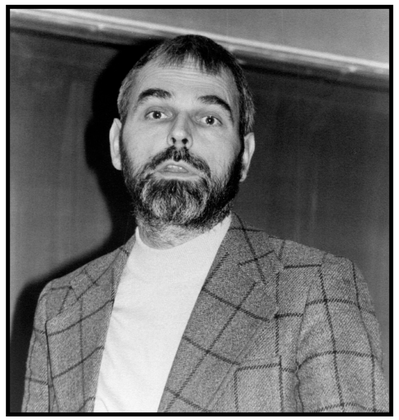
236 *ms* ~6x slower

```
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?

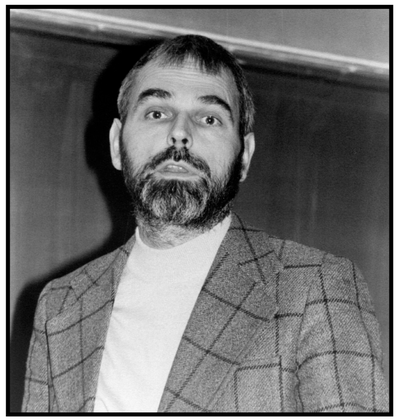


Volker Strassen, *1969*
"Matrix multiplication has $O(n^{2.8074})$ time complexity."

MidJourney: *a galactic computer*

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

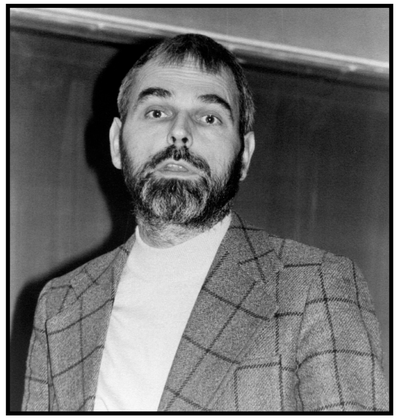
Virginia Williams et al., *2023*

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

MidJourney: *a galactic computer*

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*

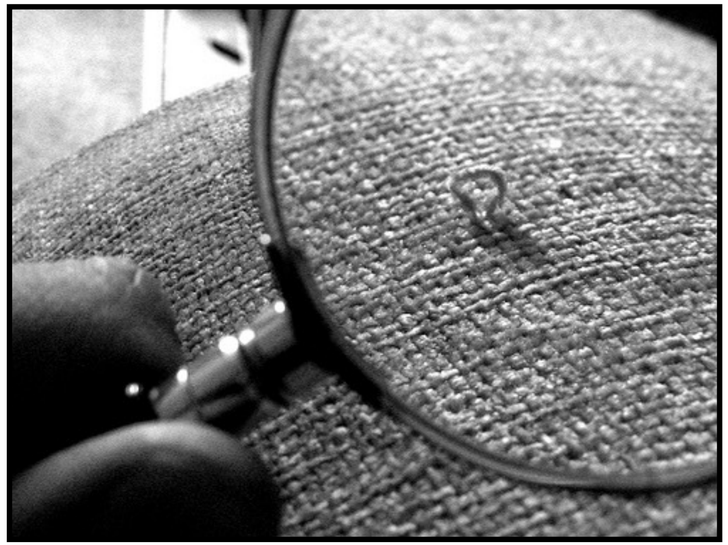
Quite possible that the true cost is $O(n^2)$. *Nobody knows!*

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

Cost of matrix operations

Forward/backward substitution

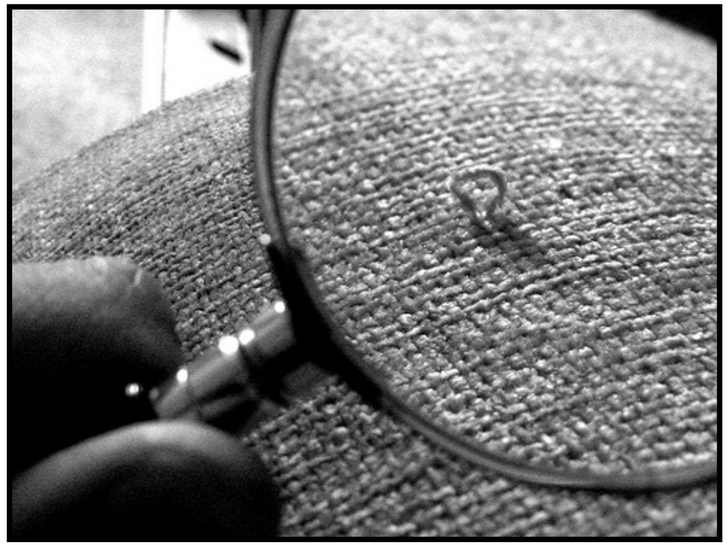
$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{b} \end{array}\right] = \left[\begin{array}{cccc|c} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{array}\right]$$



Cost of matrix operations

Forward/backward substitution

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



Cost of matrix operations

Triangular system solving *aka.* *"substitution"*.

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

Cost of matrix operations

Triangular system solving *aka. "substitution"*.

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

Cost of matrix operations

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

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

100 million matrix entries!

Triangular system solving *aka. "substitution"*.

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

Cost of matrix operations

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

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

100 million matrix entries!

Triangular system solving *aka. "substitution"*.

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

2'755 *ms* (measured on an M1 Mac)

Cost of matrix operations

Triangular system solving *aka. "substitution"*.

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

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

100 million matrix entries!

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

2'755 *ms*

recommended

30 *ms*

(measured on an M1 Mac)

Cost of matrix operations

Triangular system solving *aka. "substitution"*.

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

100 million matrix entries!

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

2'755 *ms* (measured on an M1 Mac)

recommended

30 *ms*

```
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)
```

Cost of matrix operations

Triangular system solving *aka.* *"substitution"*.

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

100 million matrix entries!

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

recommended: 2'755 *ms* (measured on an M1 Mac) + 30 *ms*

```
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)
```

manual: 102 *ms* ~3.4x slower

Cost of matrix operations

Triangular system solving *aka. "substitution"*.

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

100 million matrix entries!

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

recommended — 2'755 *ms* (measured on an M1 Mac) / 30 *ms*

```
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)
```

manual — 102 *ms* — ~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
```

Cost of matrix operations

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

100 million matrix entries!

Triangular system solving *aka.* *"substitution"*.

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{P}\mathbf{x}_1 = \mathbf{b}$ (permute), solve $\mathbf{L}\mathbf{x}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{U}\mathbf{x}_3 = \mathbf{x}_2$ (backward subst.)

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

recommended — 2'755 *ms* (measured on an M1 Mac) / 30 *ms*

```
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)
```

manual — 102 *ms* — ~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
```

not invented here syndrome! — 150 *ms* — 5x slower

Cost of matrix operations

Triangular system solving *aka.* *"substitution"*.

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

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

100 million matrix entries!

LU permutation produces $\mathbf{PLU} = \mathbf{A}$

Usage: solve $\mathbf{Px}_1 = \mathbf{b}$ (permute), solve $\mathbf{Lx}_2 = \mathbf{x}_1$ (forward subst.), solve $\mathbf{Ux}_3 = \mathbf{x}_2$ (backward subst.)

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

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)

# As above, but using the following method 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
```

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

recommended	275 *ms*	(measured on an M1 Mac)
	30 *ms*	
manual	102 *ms*	3.4x slower
not invented here syndrome!	150 *ms*	5x slower

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

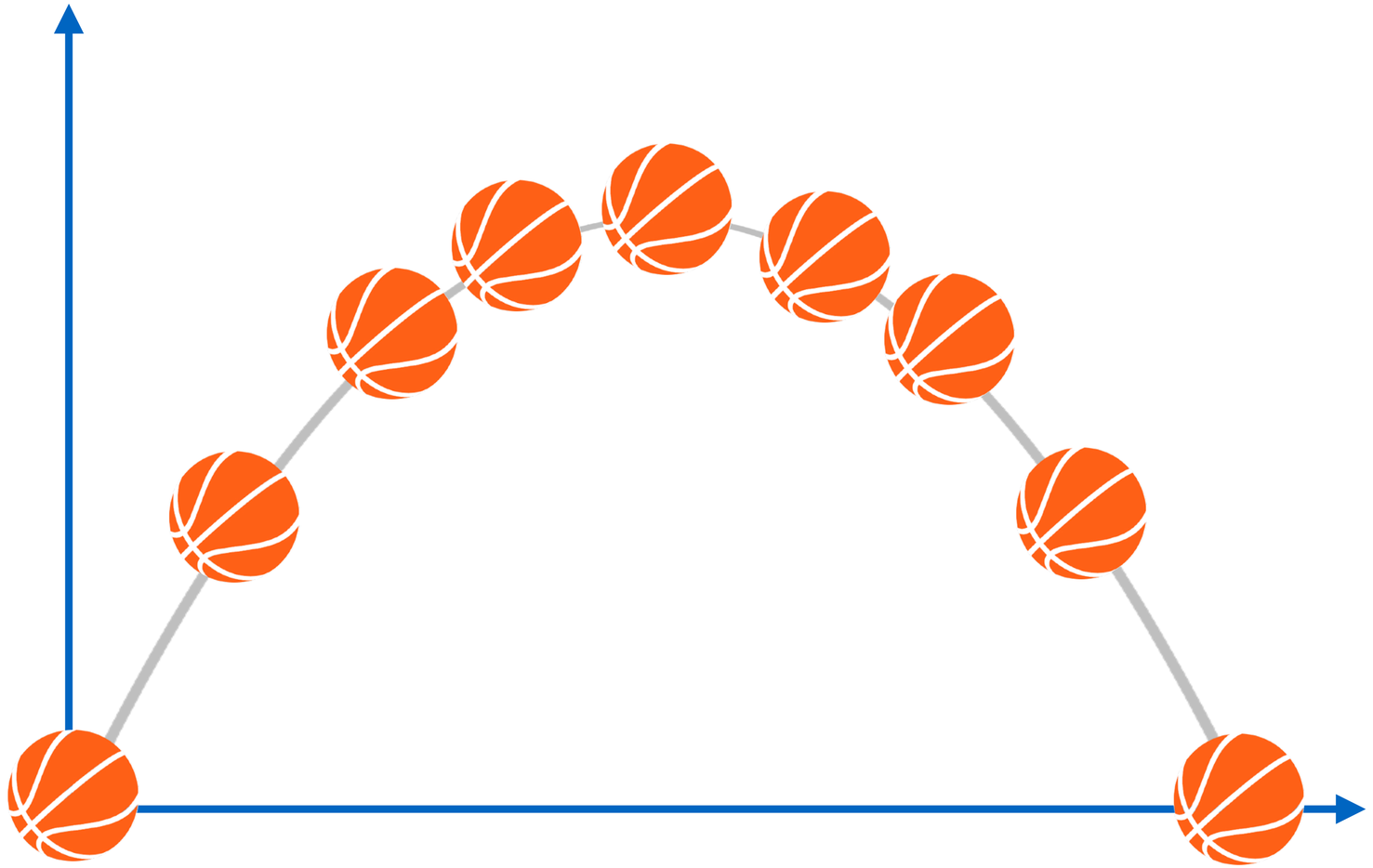
Examples of linear systems

Examples of linear systems

Examples of linear systems

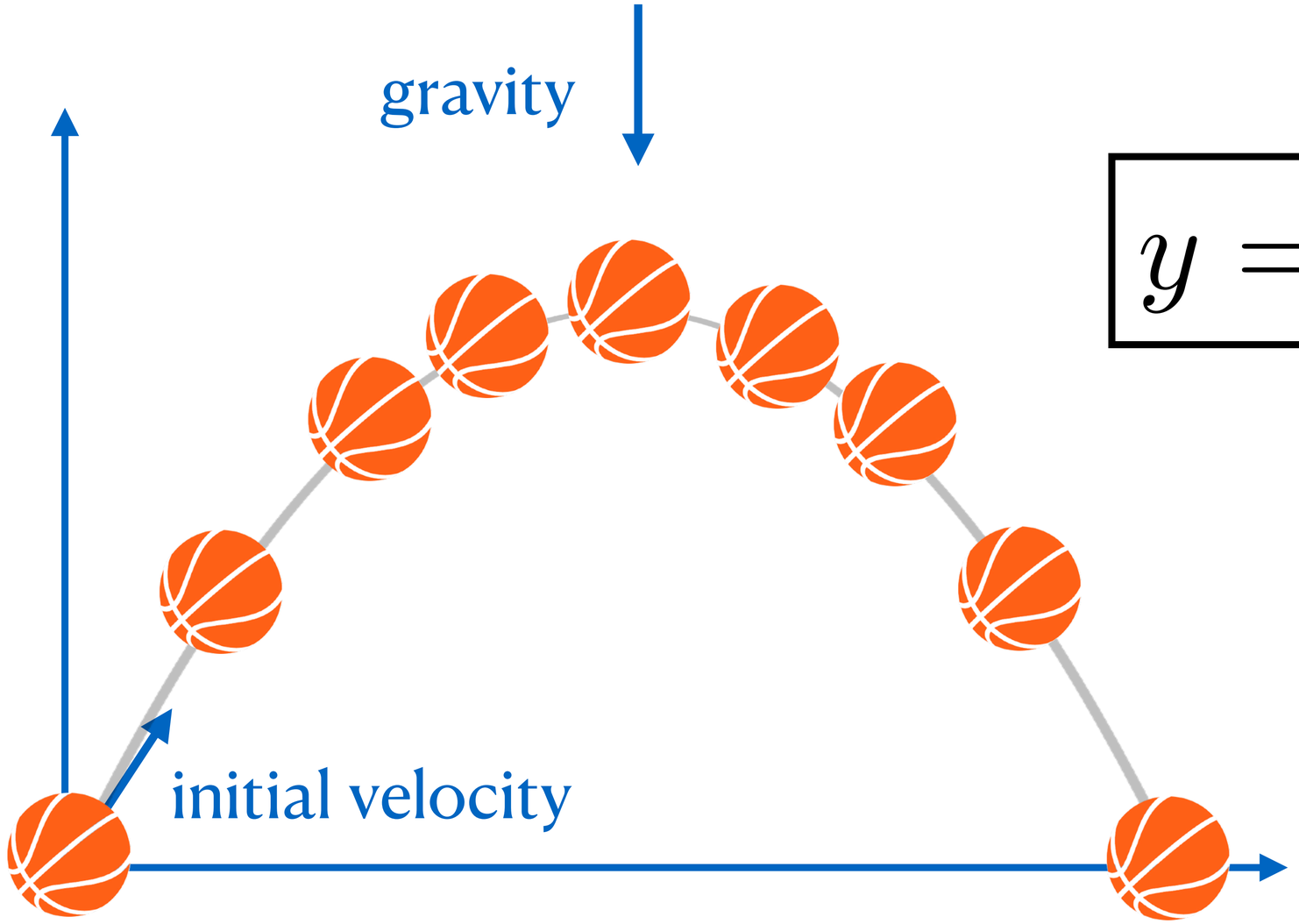
How can we model this?

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



How can we model this?

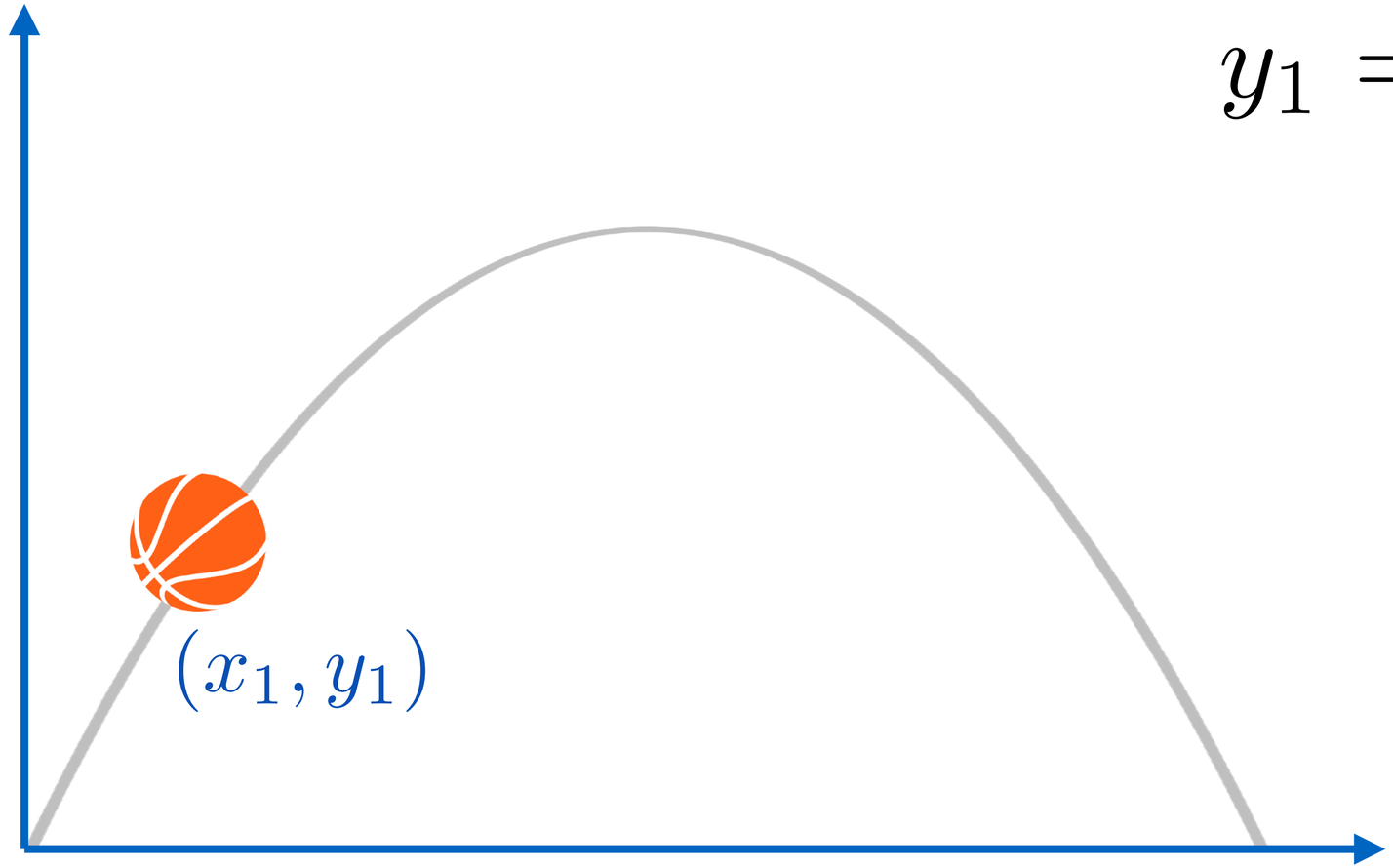
This kind of motion is well-understood! **(Ideal ballistic motion)**



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

Ideal Ballistic Motion

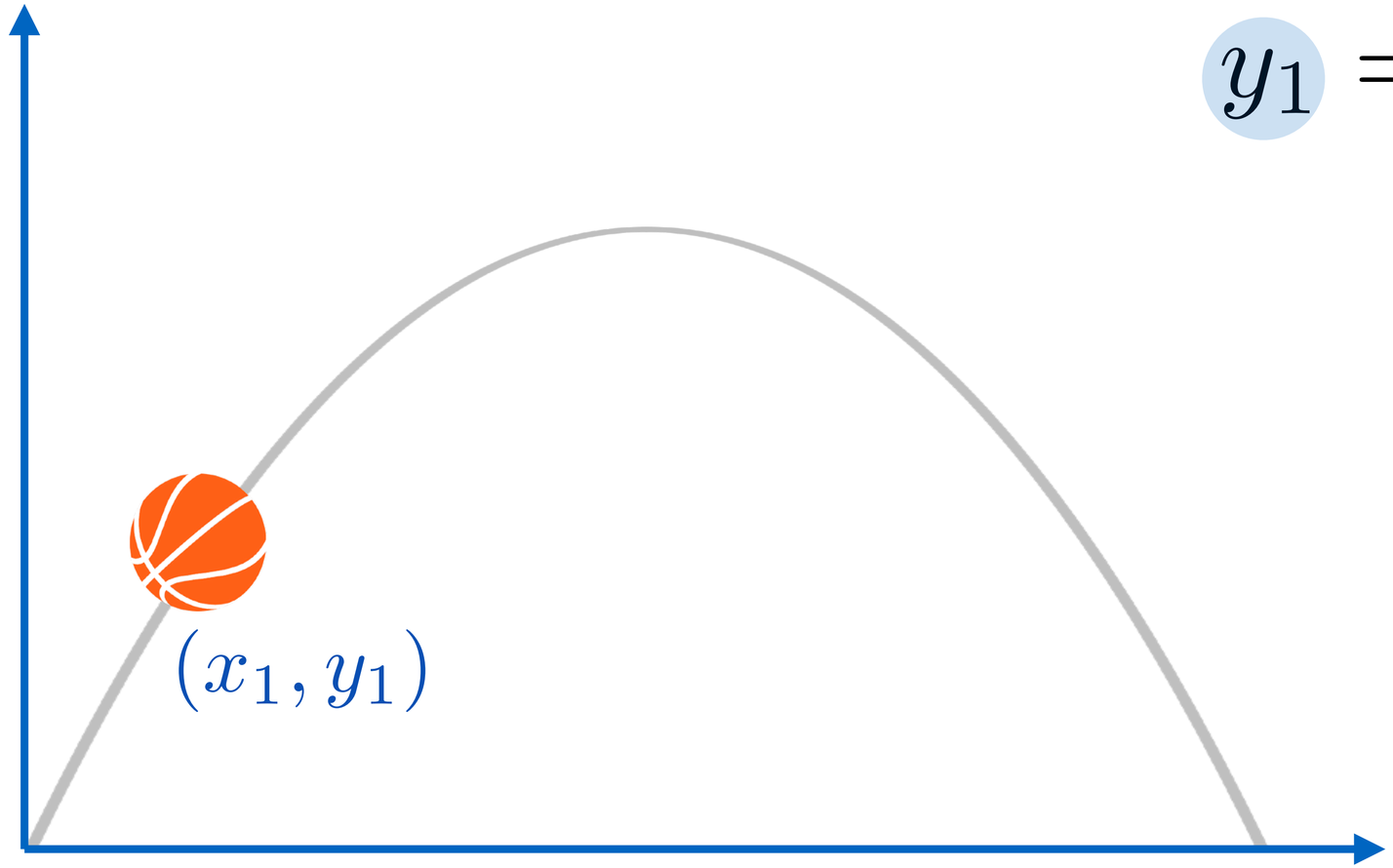
Step 1: Think about observables and unknowns



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

Ideal Ballistic Motion

Step 1: Think about observables and unknowns

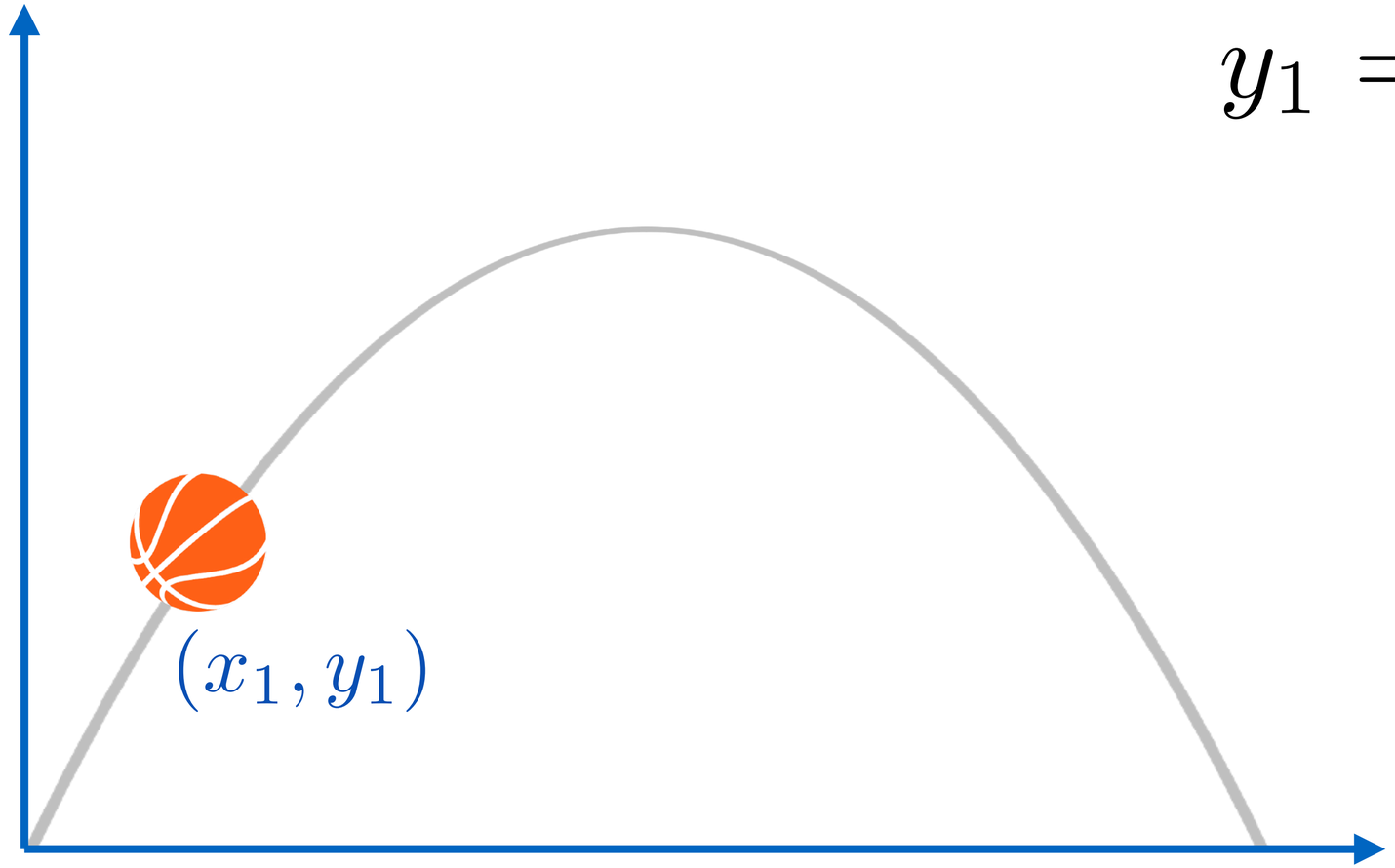


$$y_1 = ax_1^2 + bx_1 + c$$

known

Ideal Ballistic Motion

Step 1: Think about observables and unknowns



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

unknown

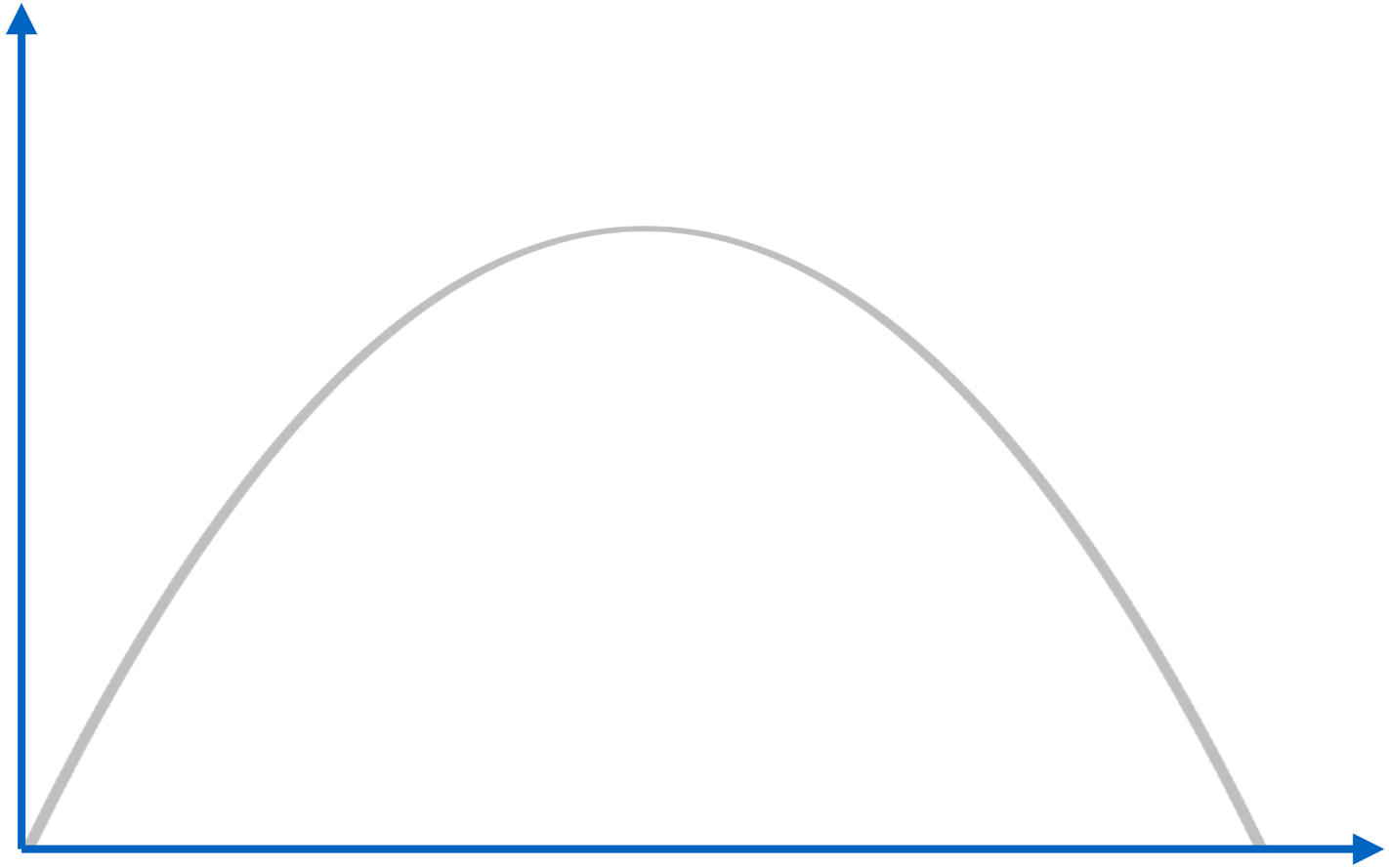
Regression

- 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

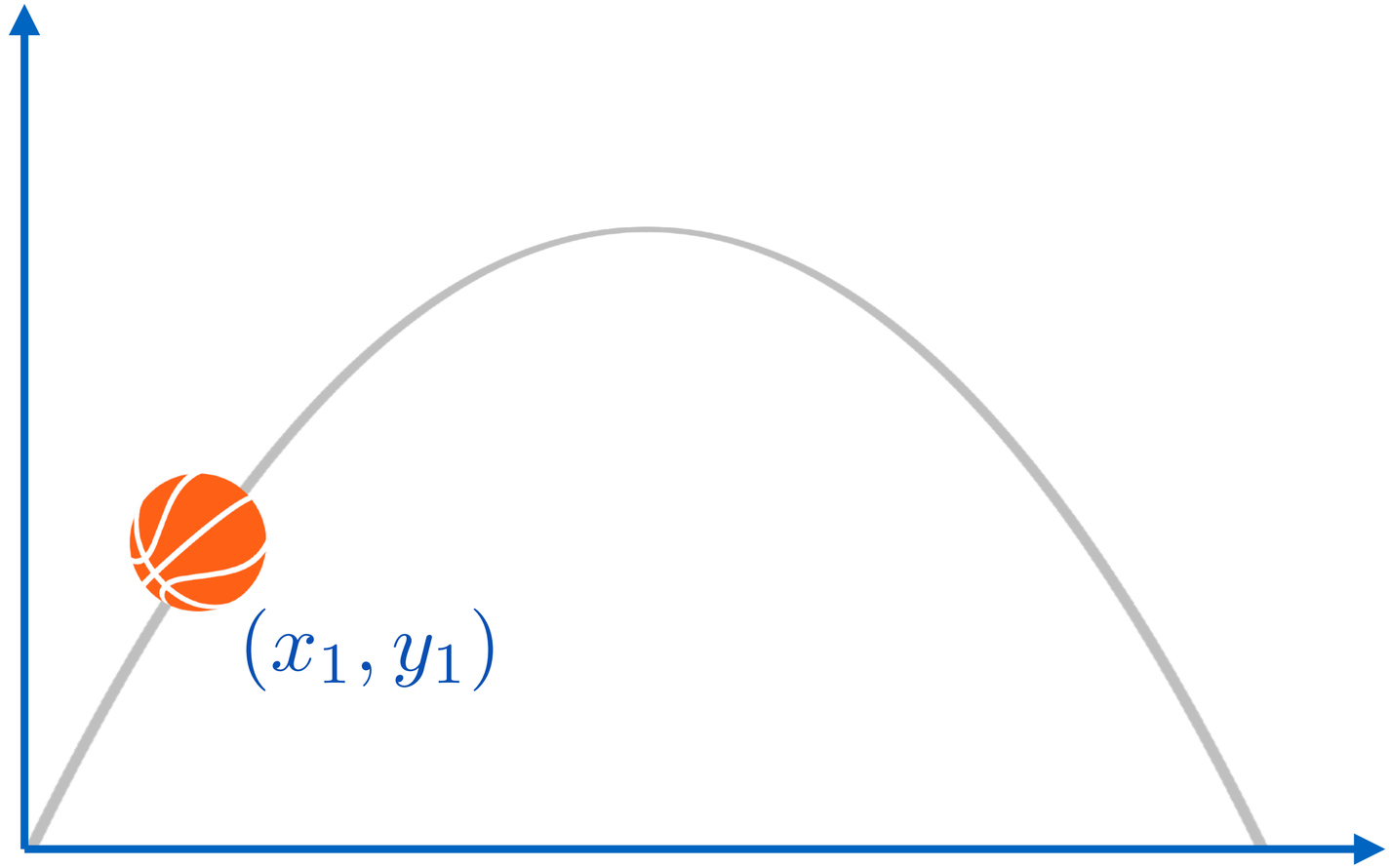
How many data points (i.e., *measurements*) do we need to solve this system?



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

Ideal Ballistic Motion

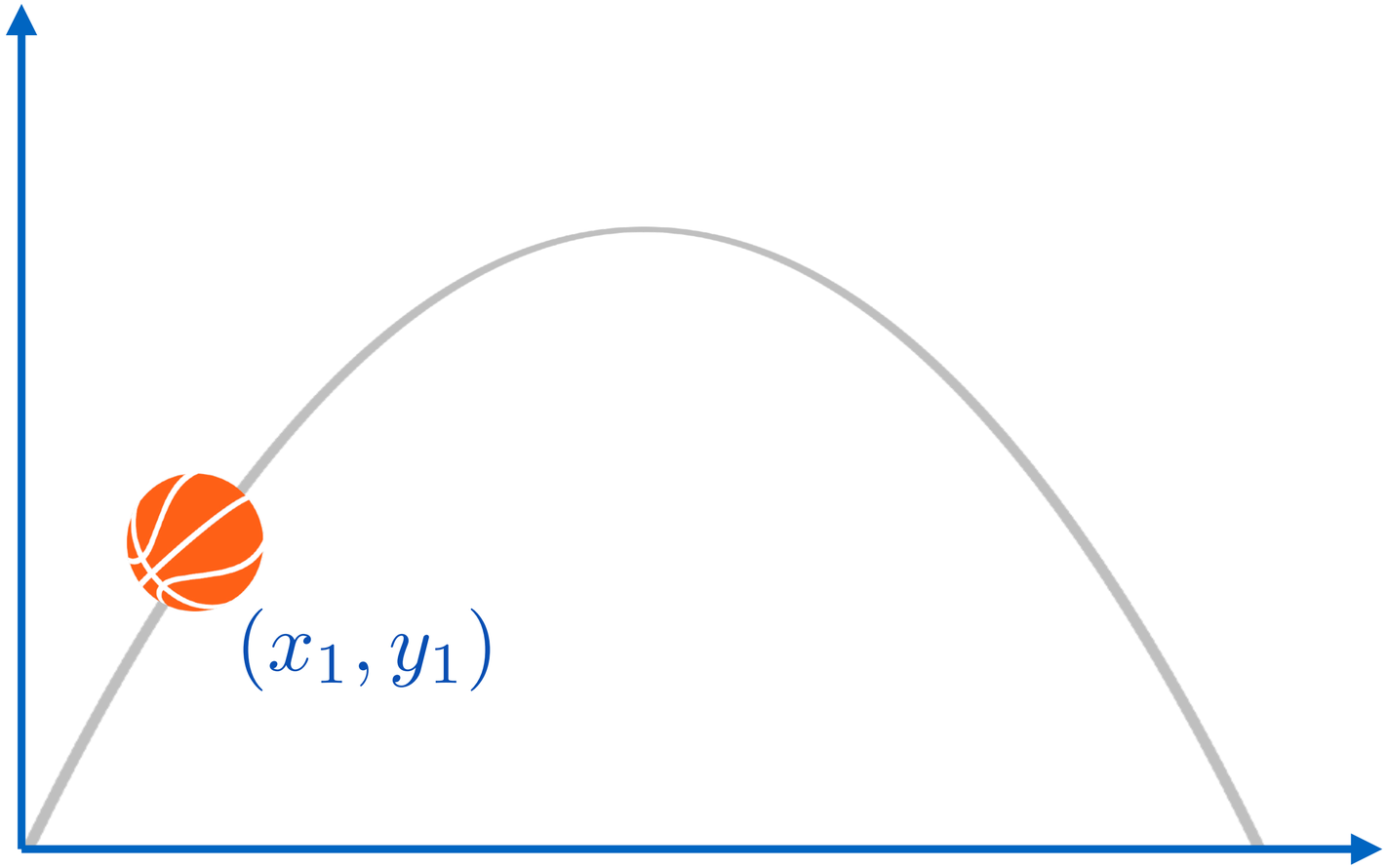
How many data points (i.e., *measurements*) do we need to solve this system?



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

Ideal Ballistic Motion

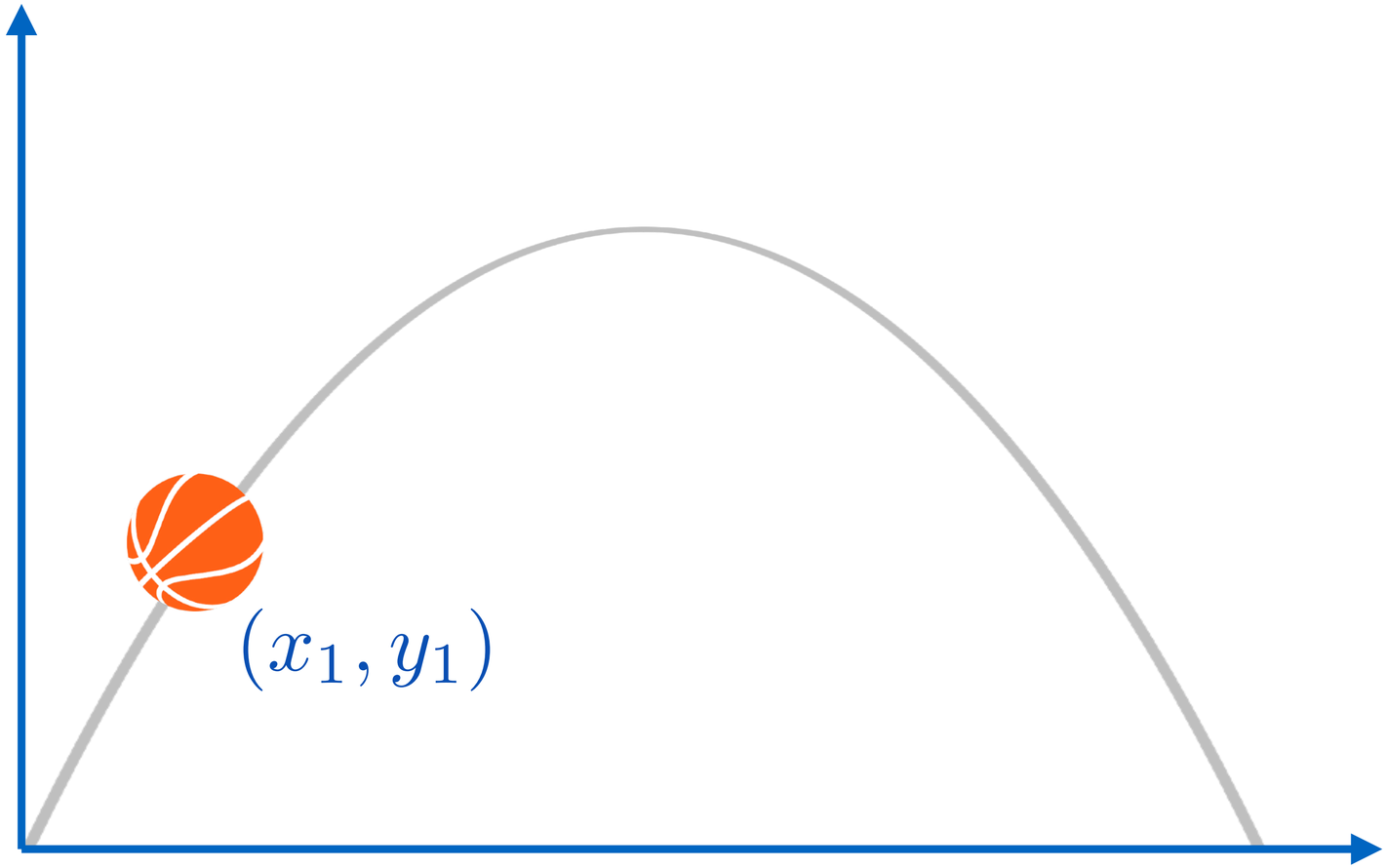
How many data points (i.e., *measurements*) do we need to solve this system?



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

Ideal Ballistic Motion

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

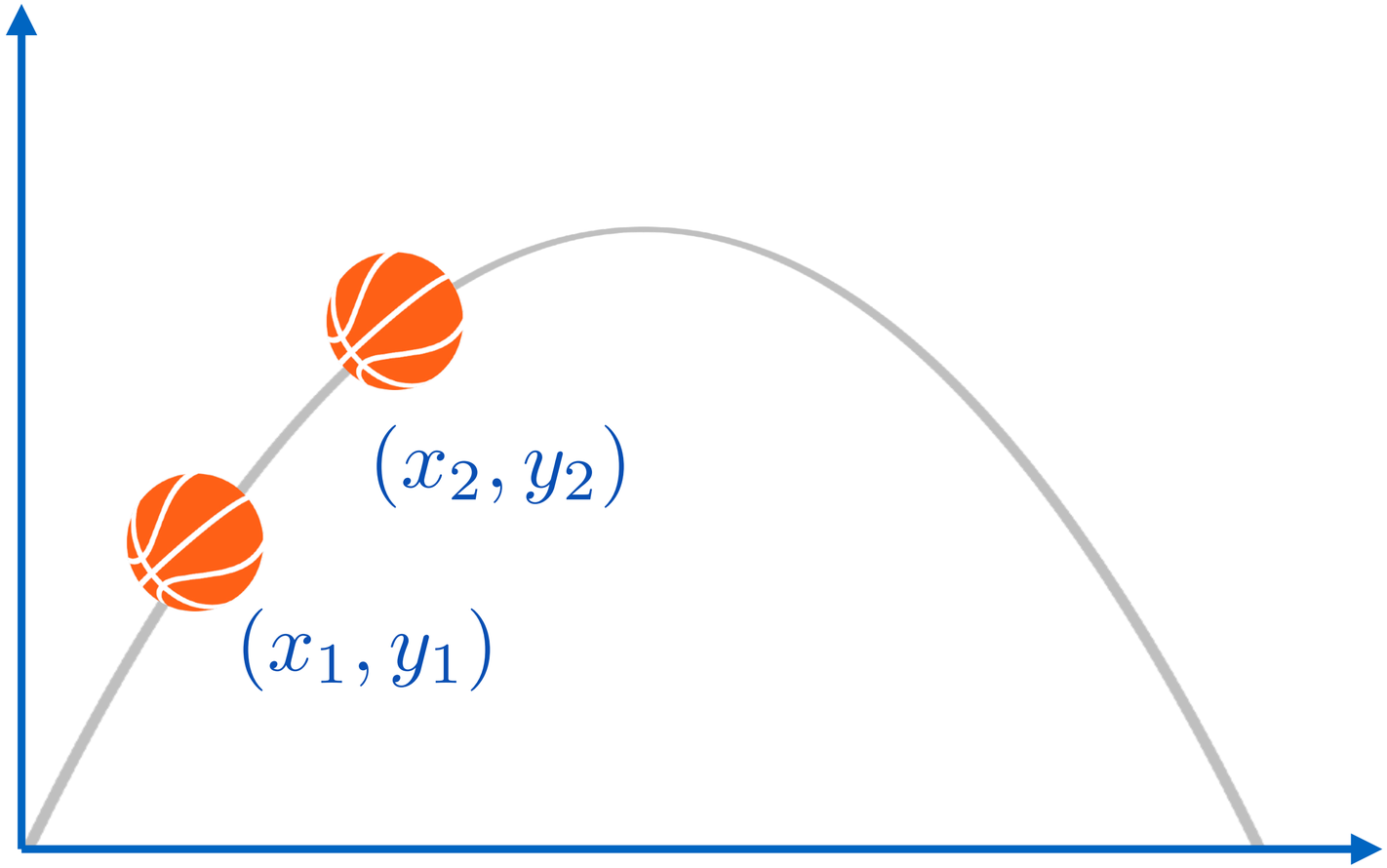


The equation is **linear** in the unknowns.

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

Ideal Ballistic Motion

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



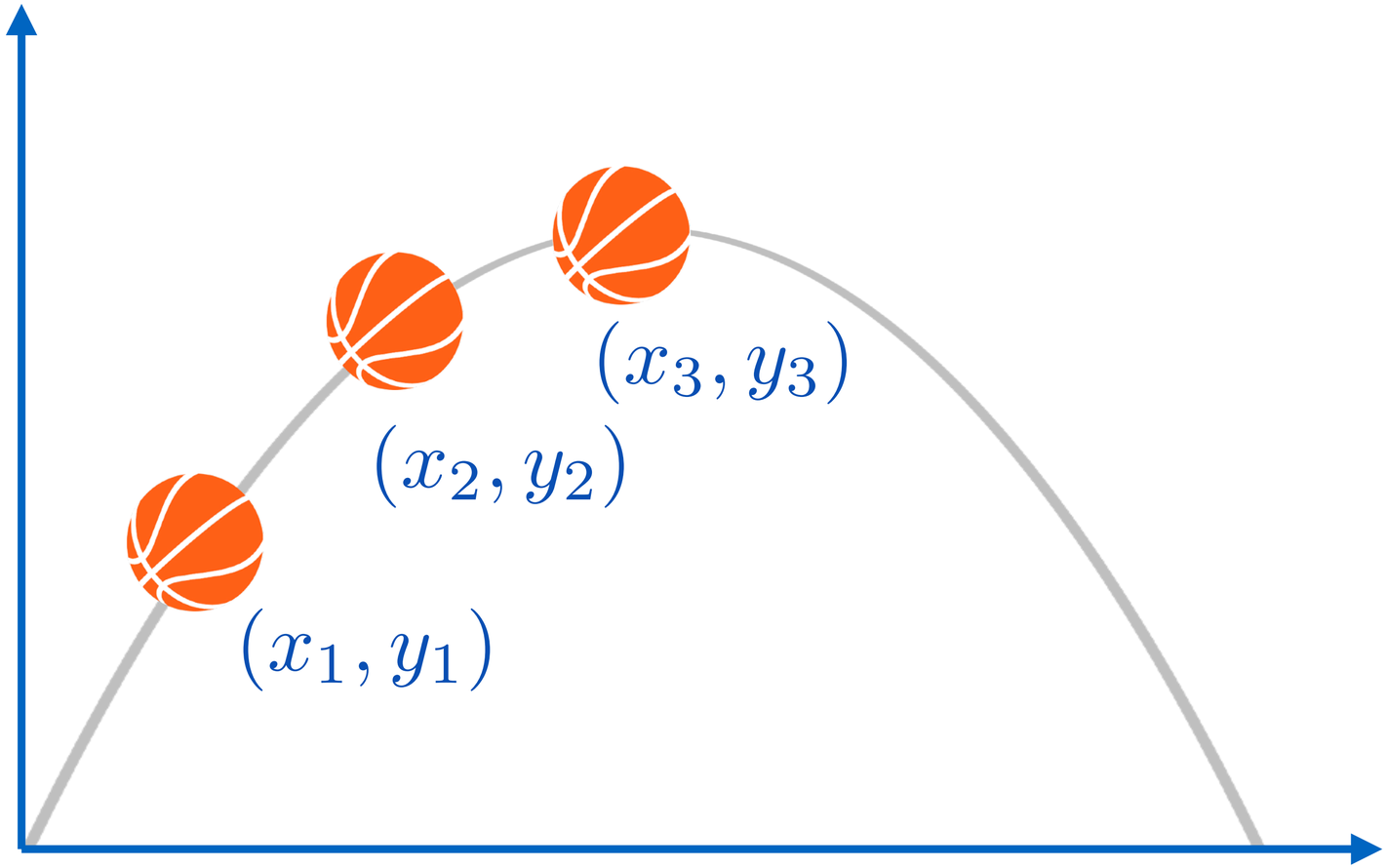
The equation is **linear** in the unknowns.

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

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

Ideal Ballistic Motion

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

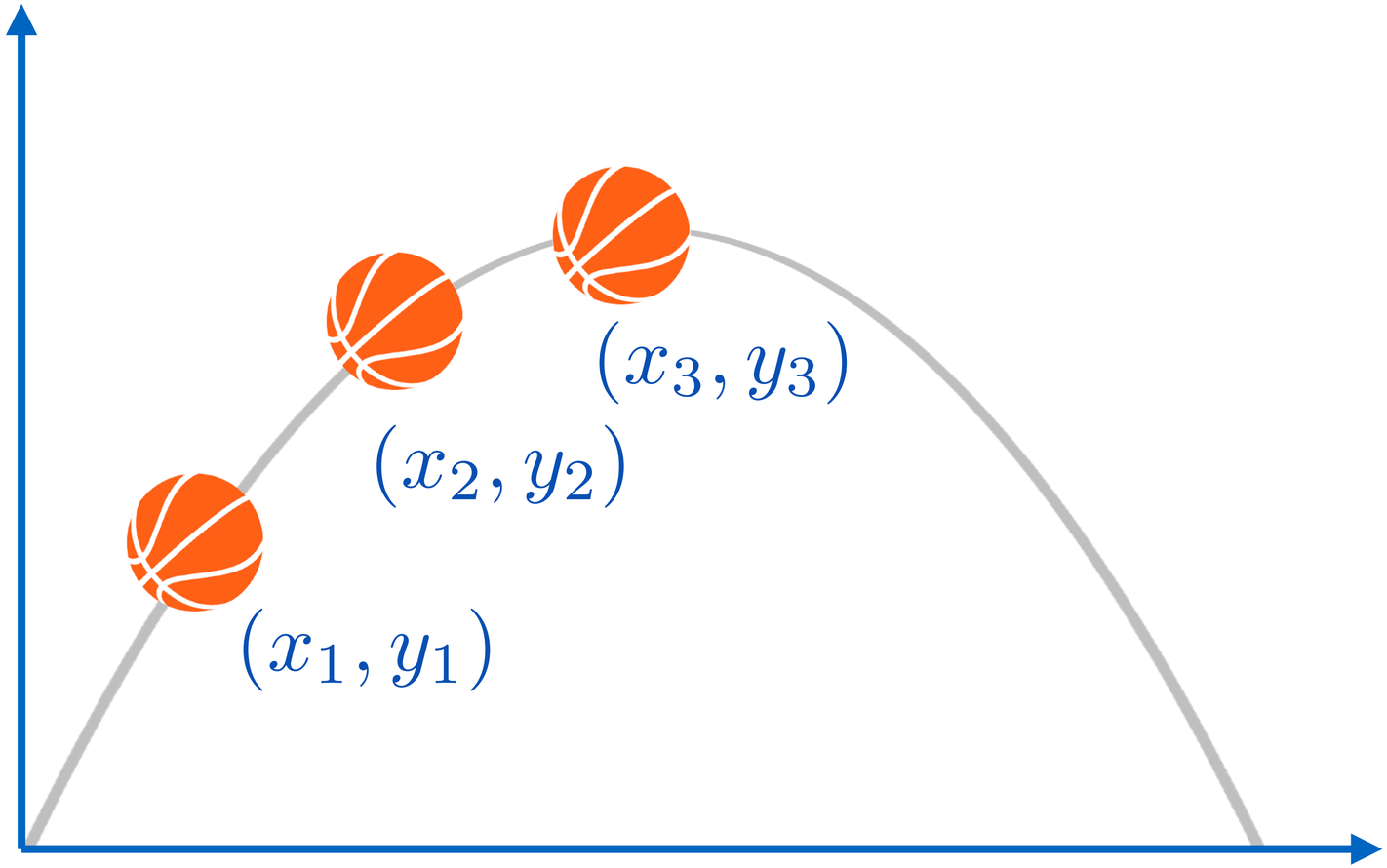


The equation is **linear** in the unknowns.

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_3 = ax_3^2 + bx_3 + c$$

Ideal Ballistic Motion

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

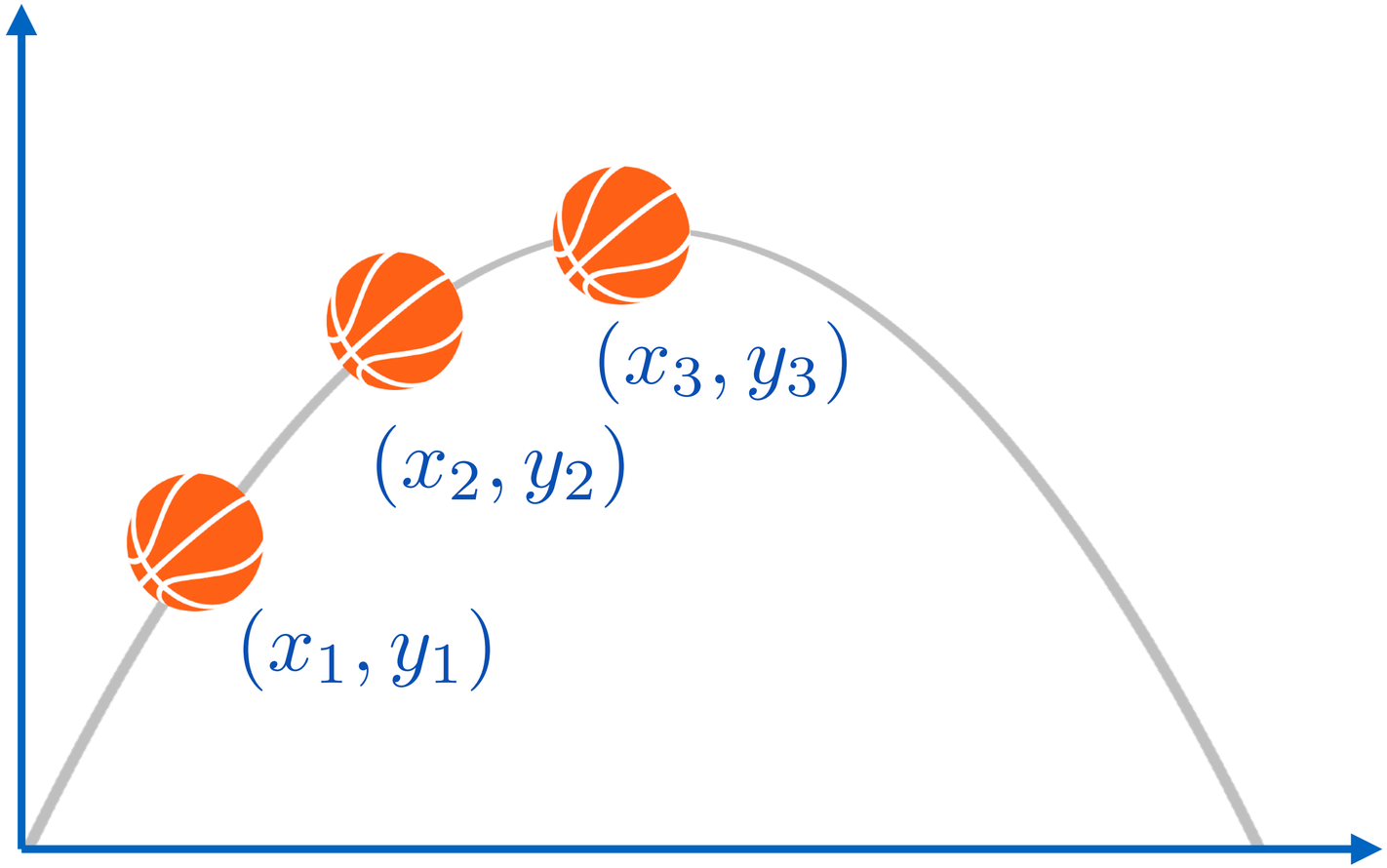


The equation is **linear** in the unknowns.

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_3 = ax_3^2 + bx_3 + c$$

Ideal Ballistic Motion

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



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

$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_3 = ax_3^2 + bx_3 + c$$

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

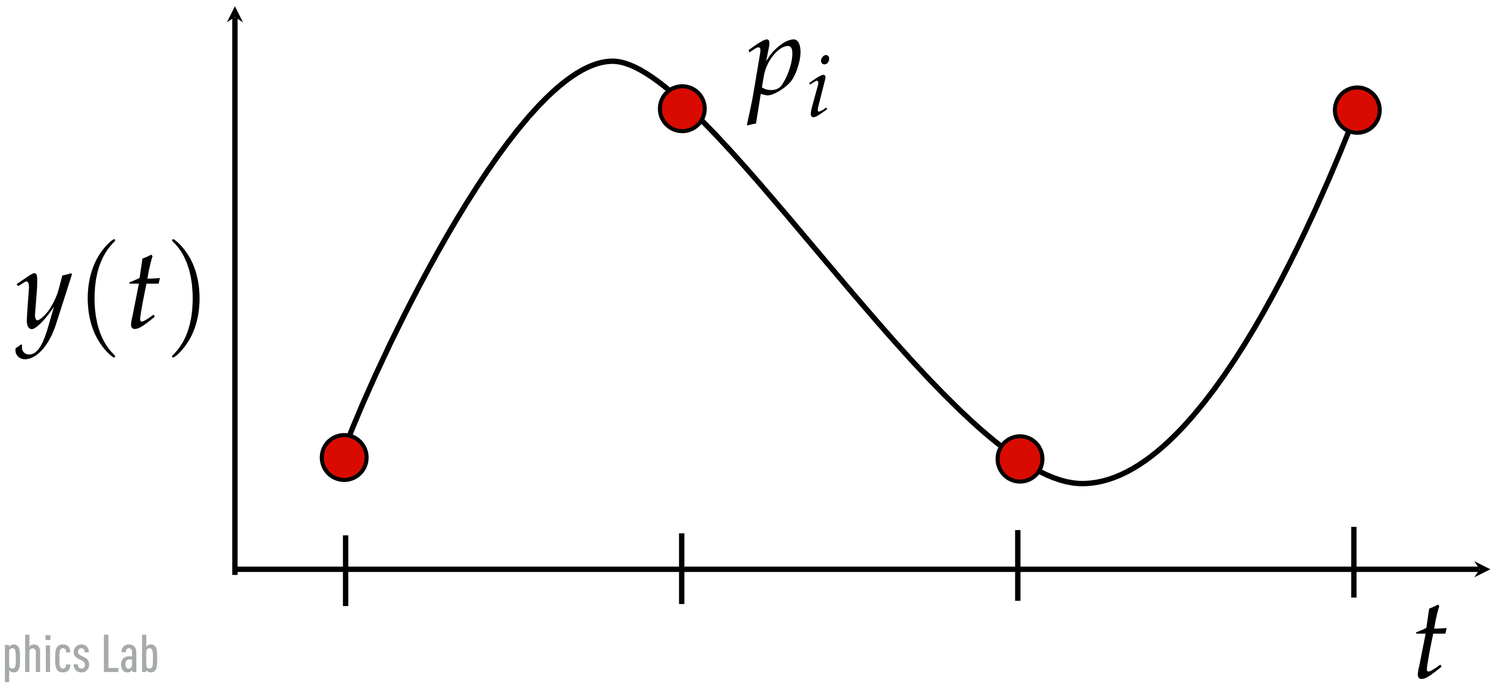
(identical, but using matrix-vector notation)

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

$$y(t_i) = p_i$$

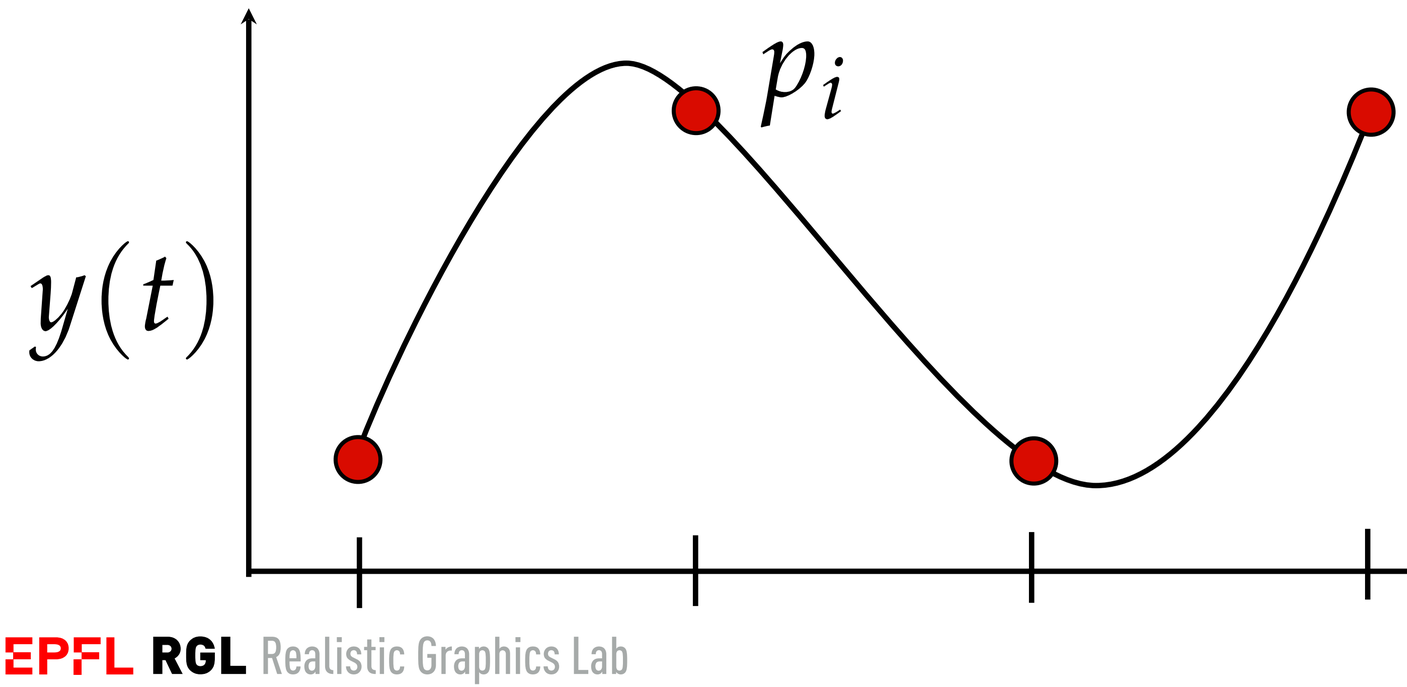


Polynomial regression

The basketball example is a representative of this larger class

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

$$y(t_i) = p_i$$

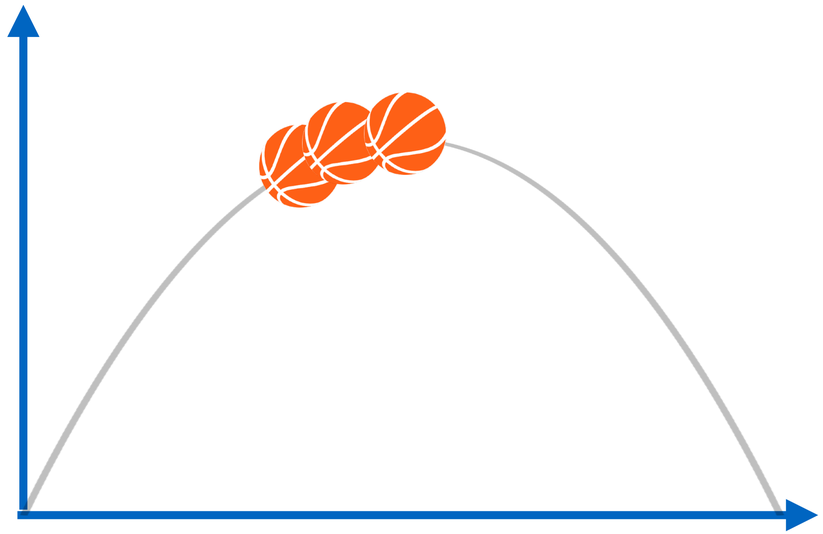


$$\begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ t_3^3 & t_3^2 & t_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

"Vandermonde matrix"

Food for thought

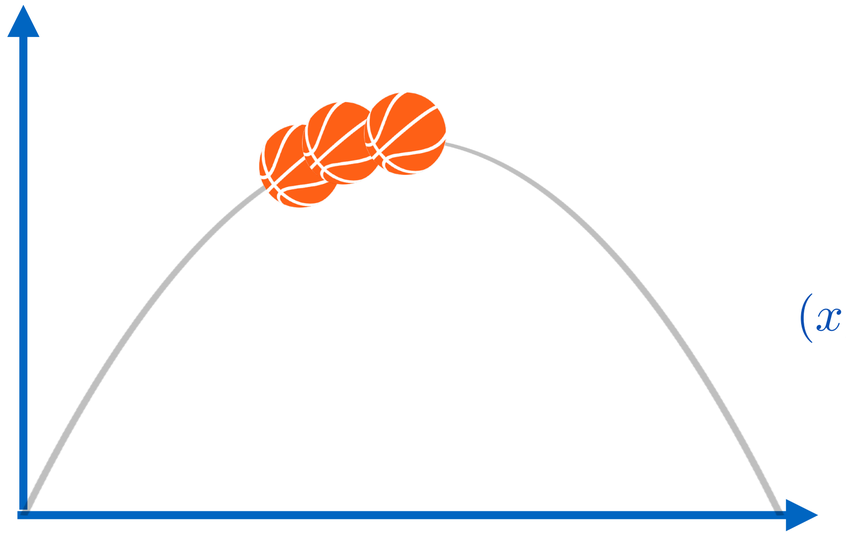
What could possibly go wrong?



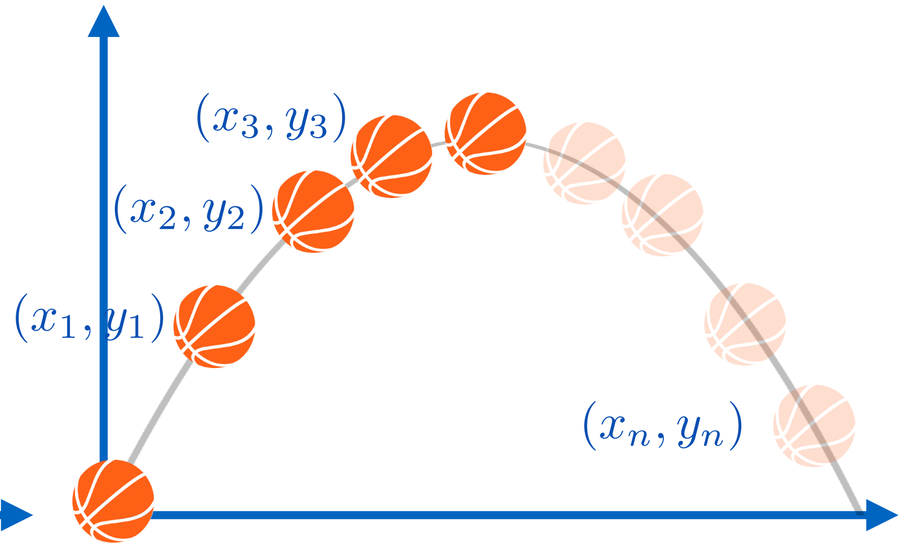
What if the observations are *weak*?

Food for thought

What could possibly go wrong?



What if the observations are *weak*?

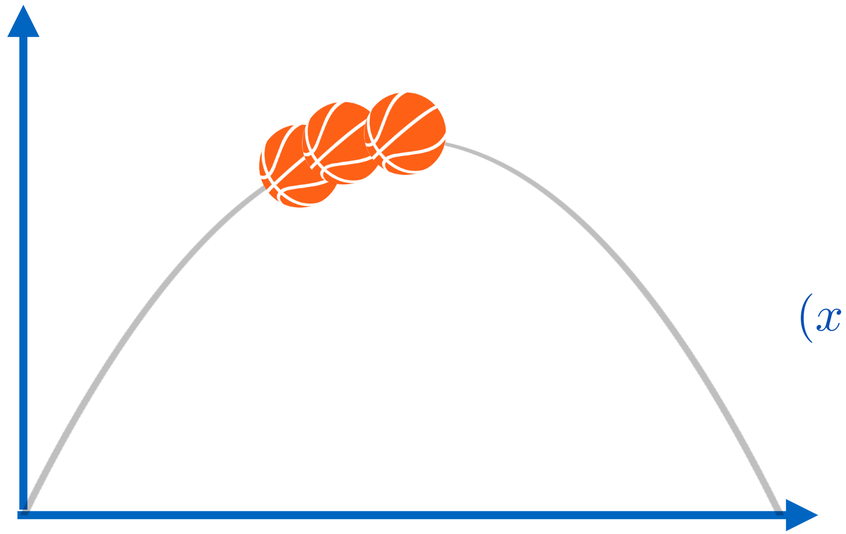


What if there are > 3 observations?

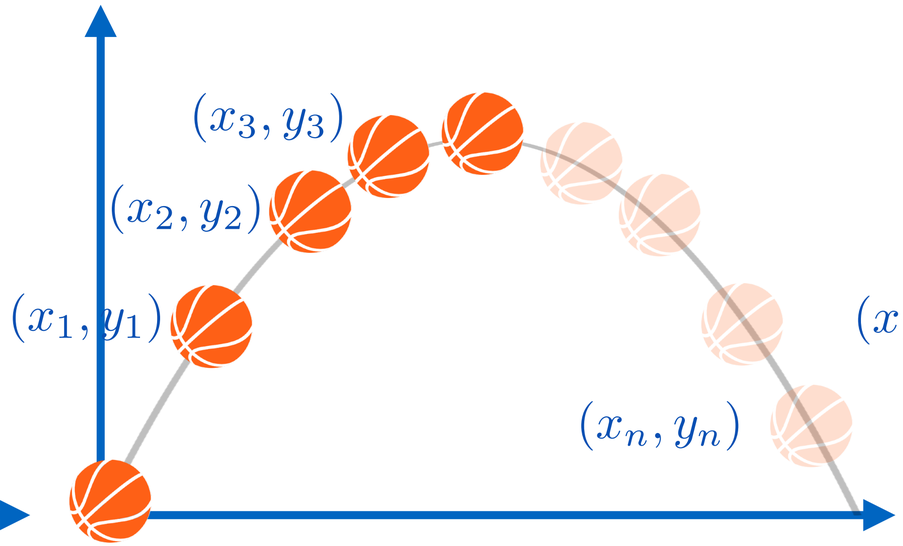
Use subset? Which ones? Use all somehow?

Food for thought

What could possibly go wrong?

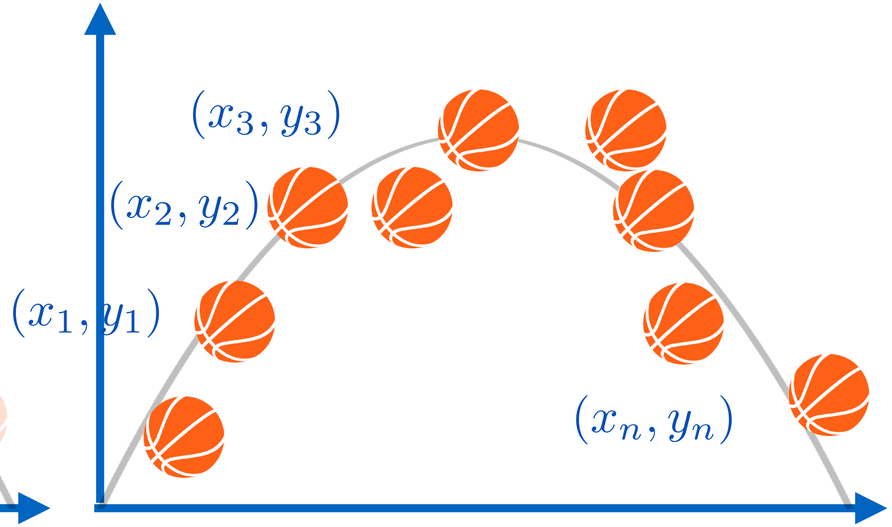


What if the observations are *weak*?



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

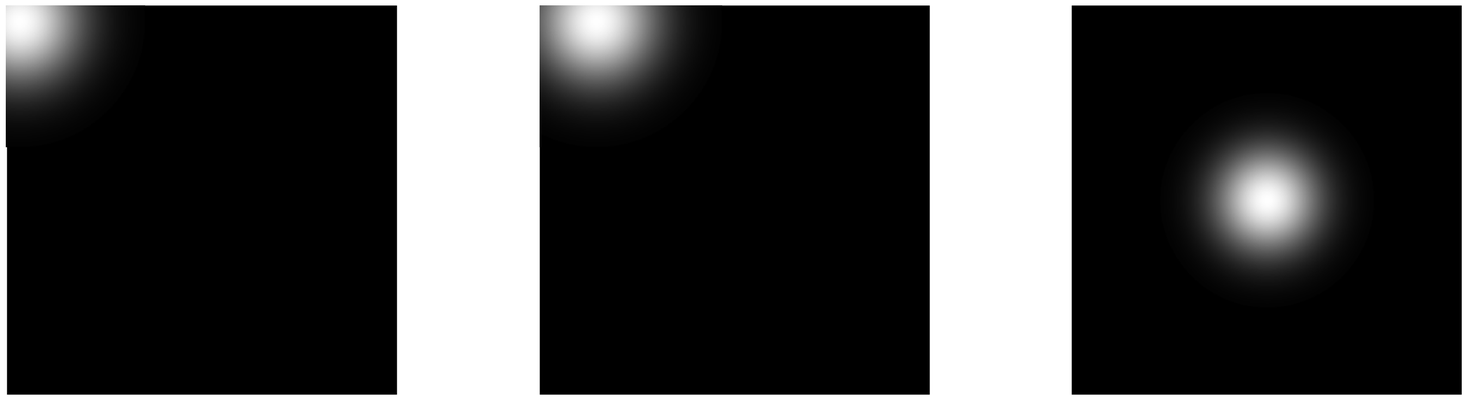

KABOOM!

Another classical example of a "bad" linear system

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

Blurry

$$\mathbf{b} = \begin{pmatrix} | & | & \cdots & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ | & | & \cdots & | \end{pmatrix} \mathbf{x}$$

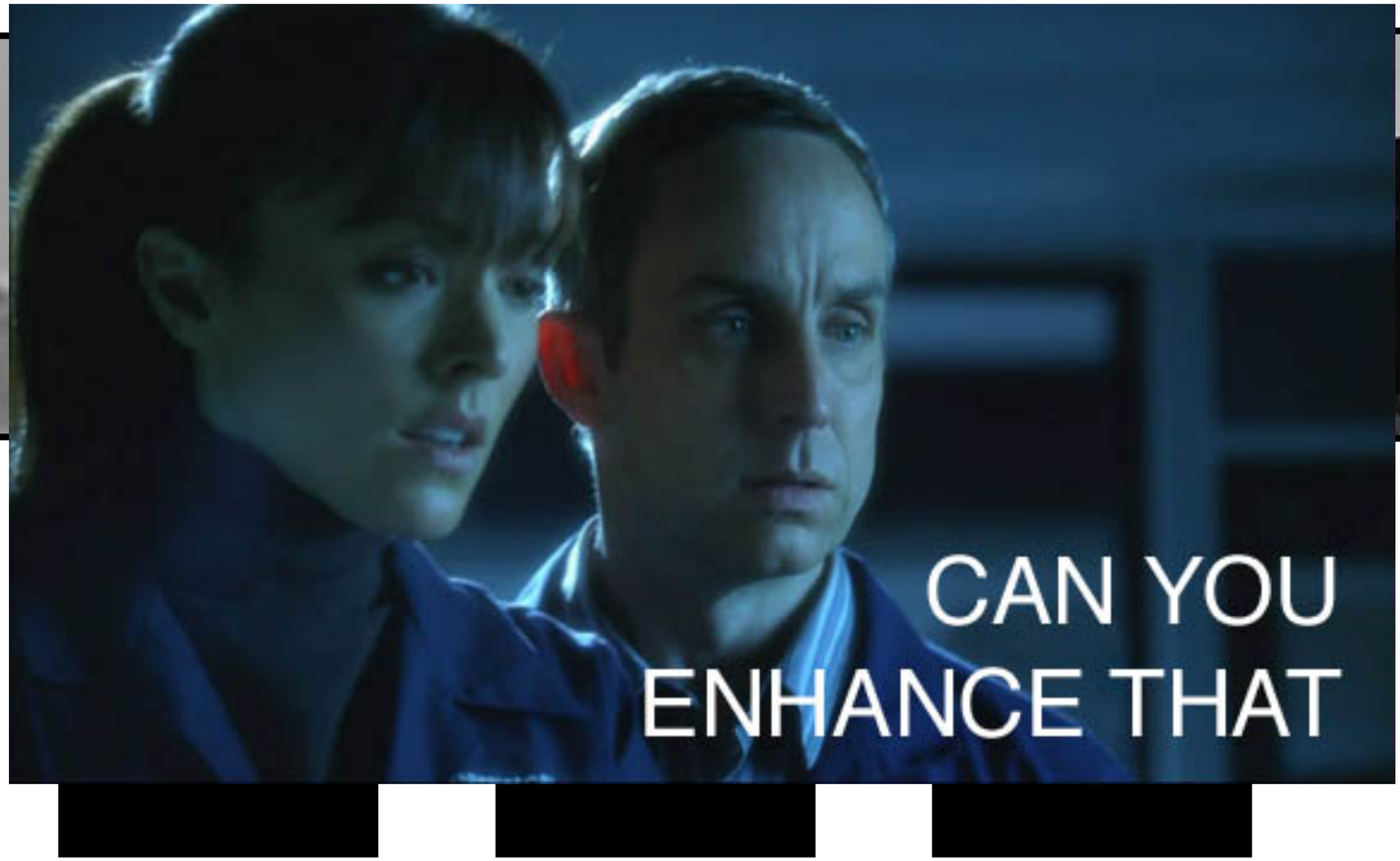


Original

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

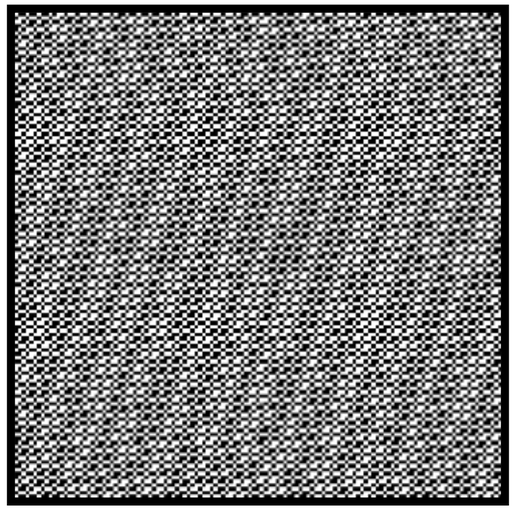
Blurry

Another classical example of a "bad" linear system

Original

Blurry

Recovered

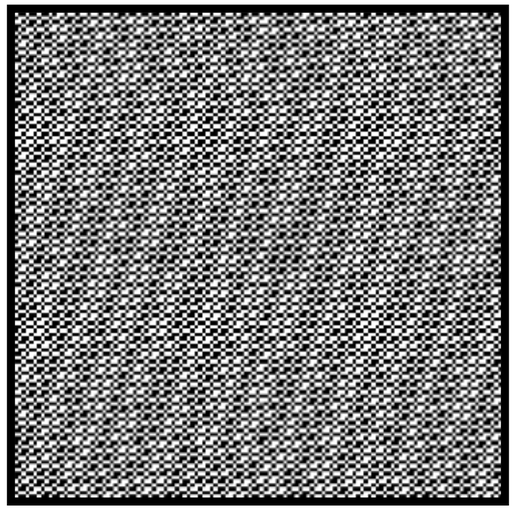


Another classical example of a "bad" linear system

Original

Blurry

Recovered



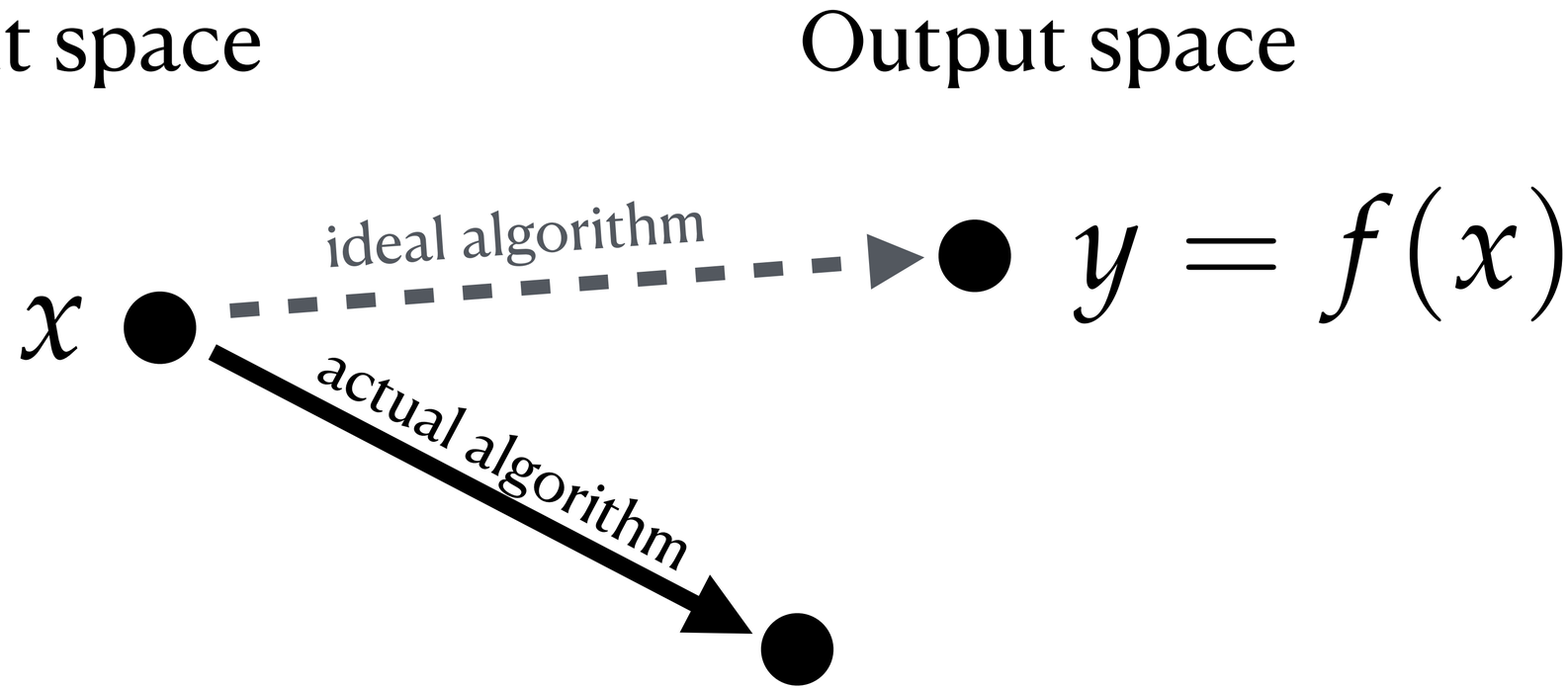
Regularized

Forward and Backward Error

Recap

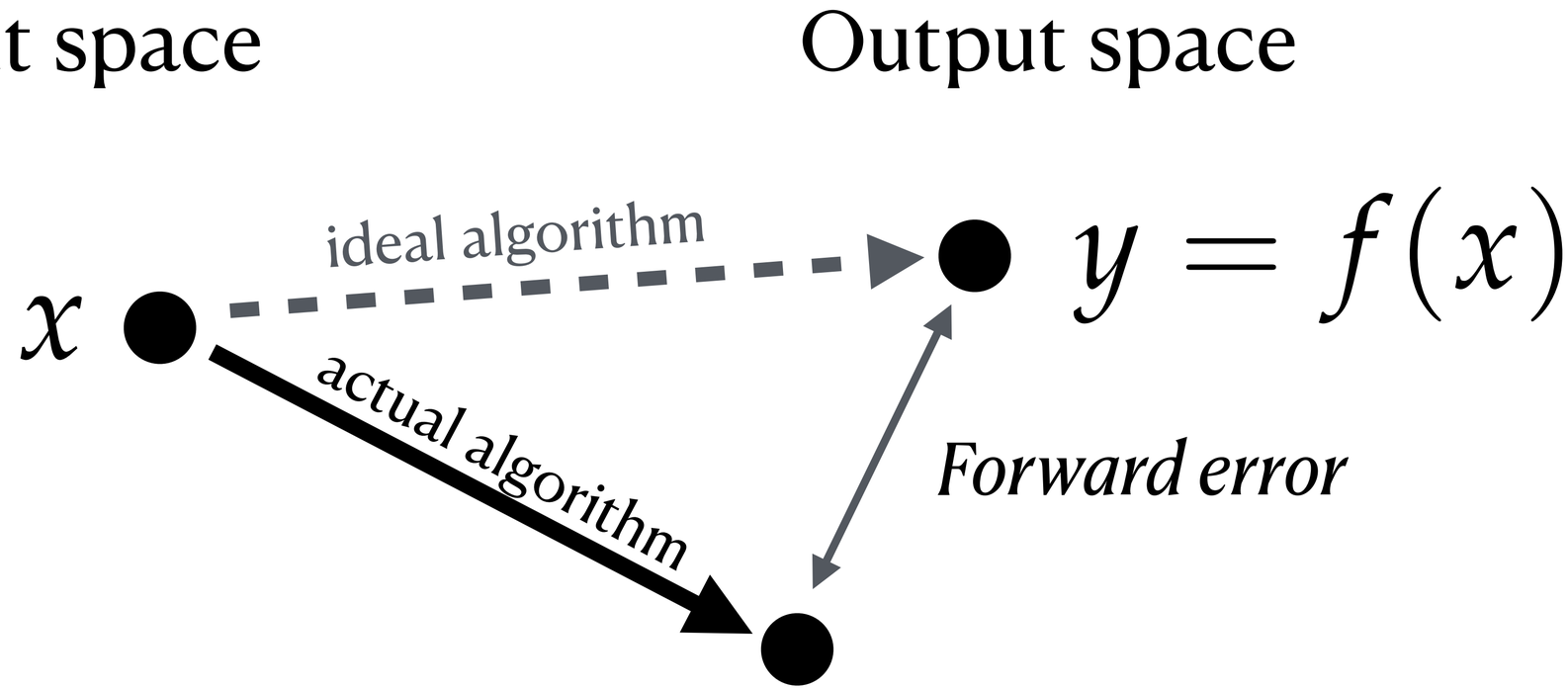
Forward and Backward Error

Recap



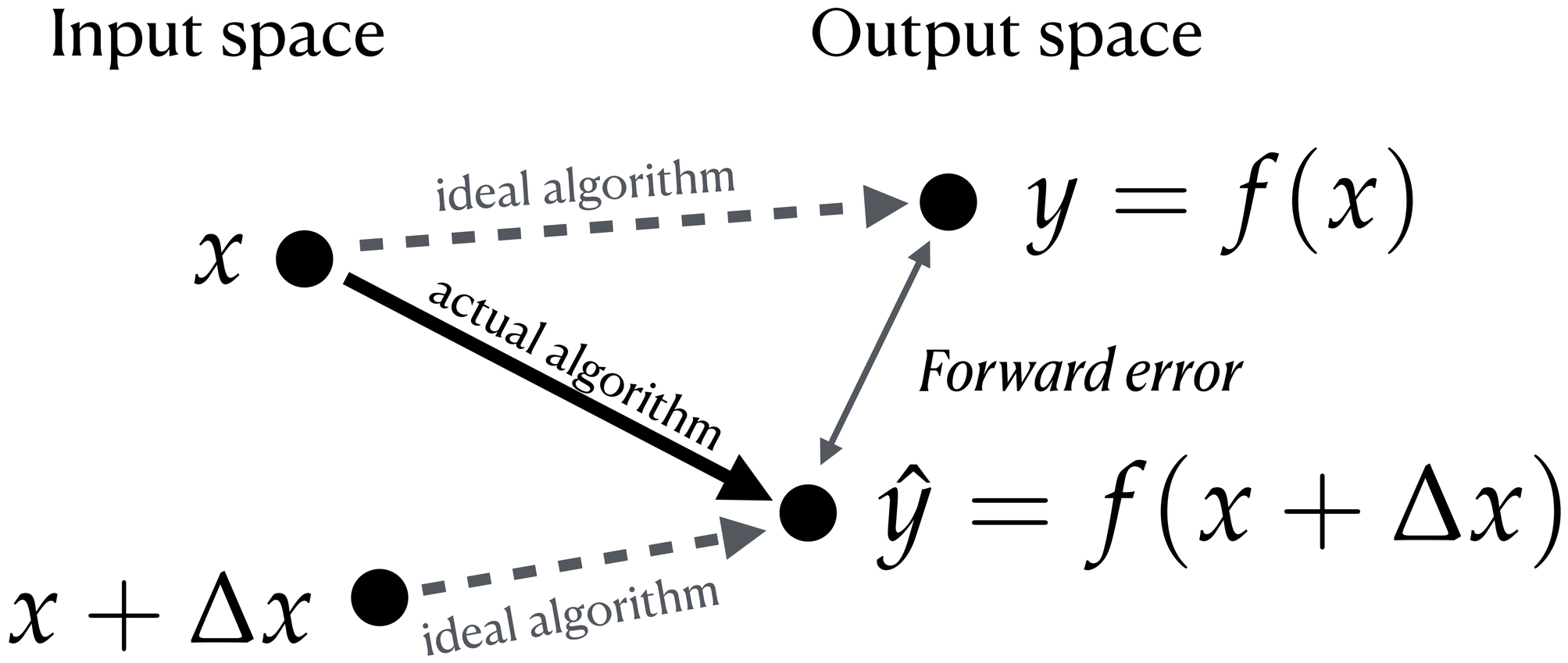
Forward and Backward Error

Recap



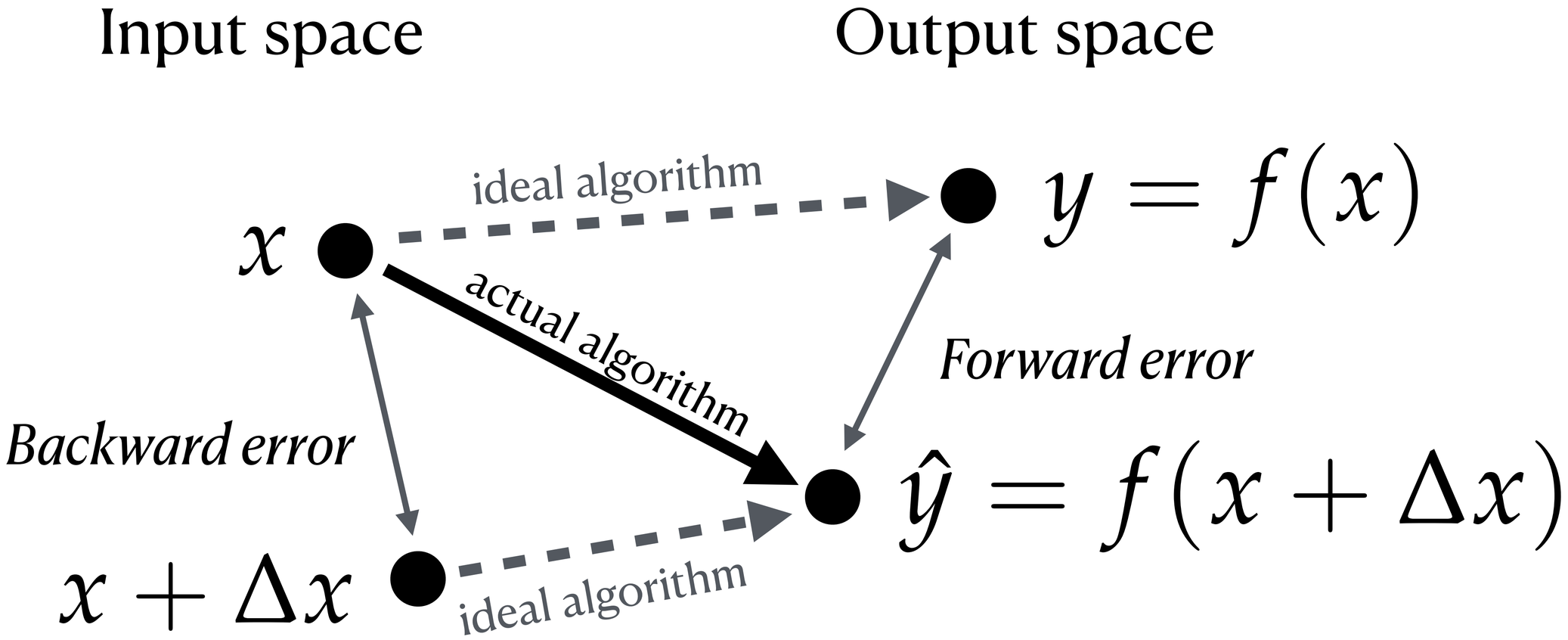
Forward and Backward Error

Recap

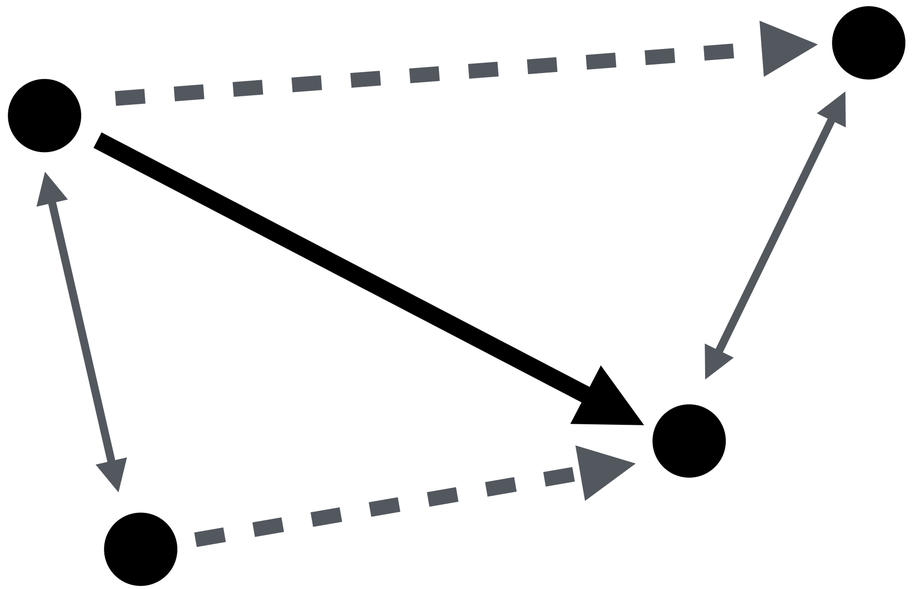


Forward and Backward Error

Recap



Conditioning of linear systems

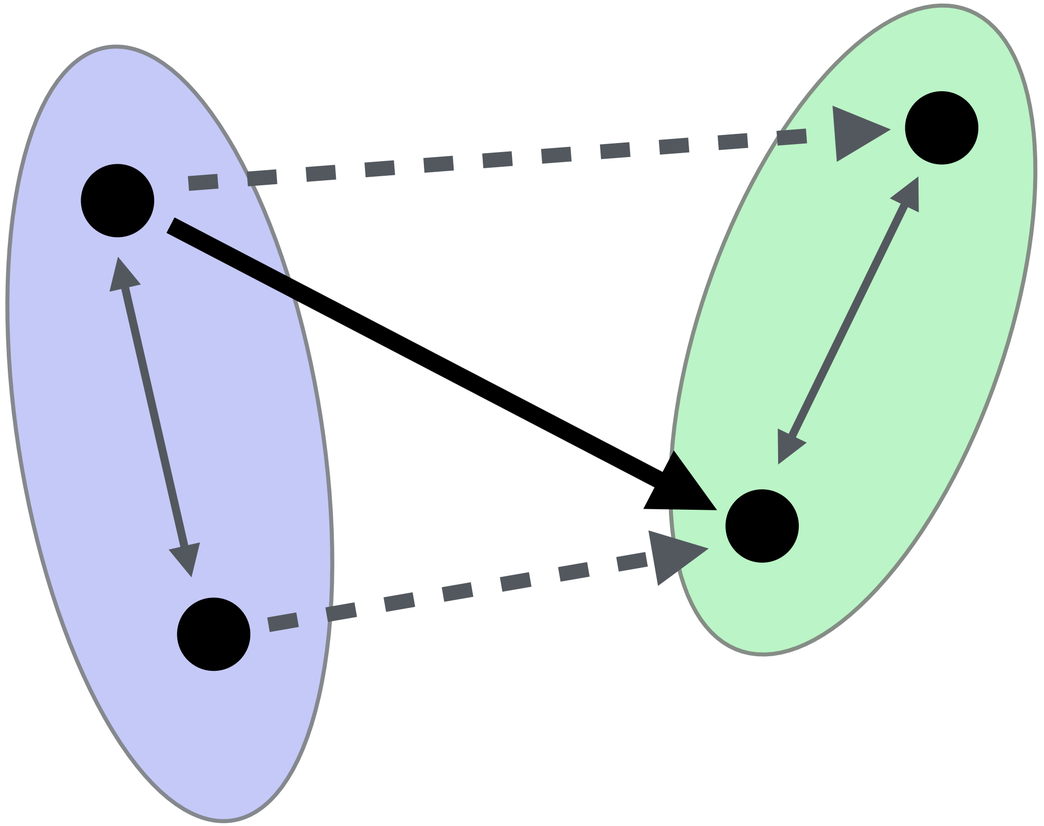


Backward error

Forward error

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

Conditioning of linear systems



Backward error

Forward error

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

Forward and Backward error

.. applied to linear system solving

Theoretical infinite precision solver

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

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!

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

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

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

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

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 $\hat{\mathbf{x}}$?

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

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

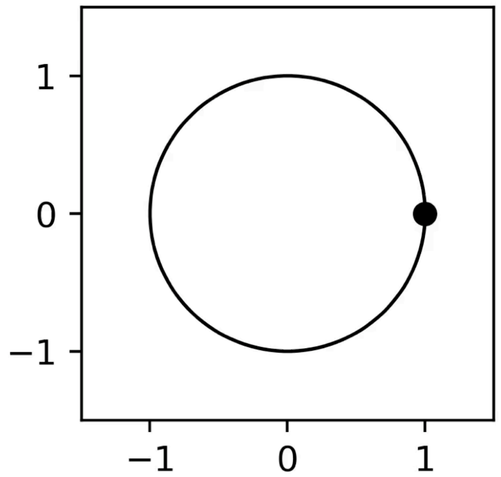
.. but how accurate is $\hat{\mathbf{x}}$?

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

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

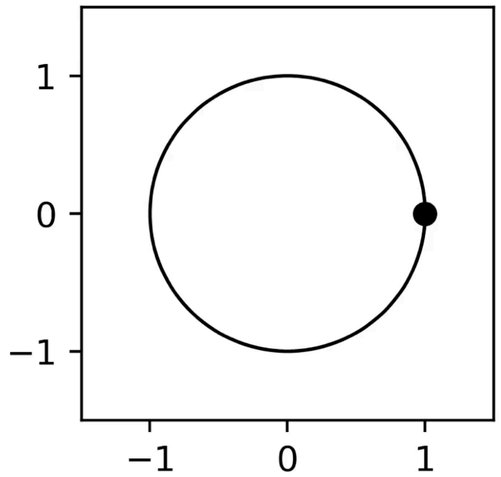
Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

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



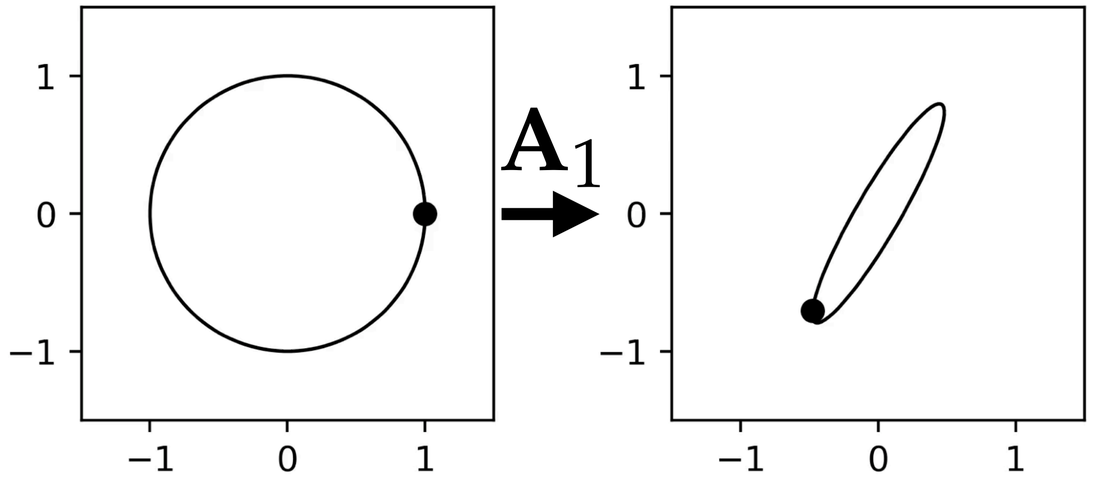
Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

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



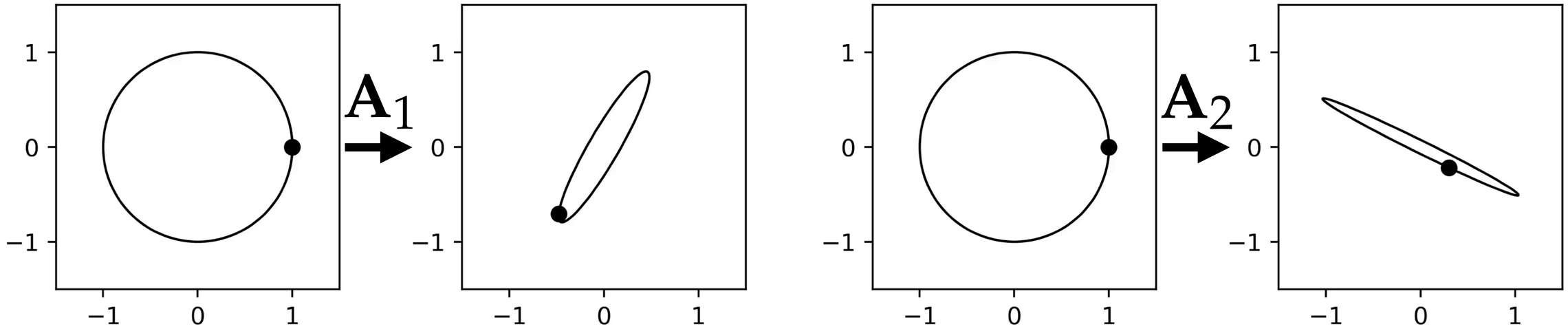
Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

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



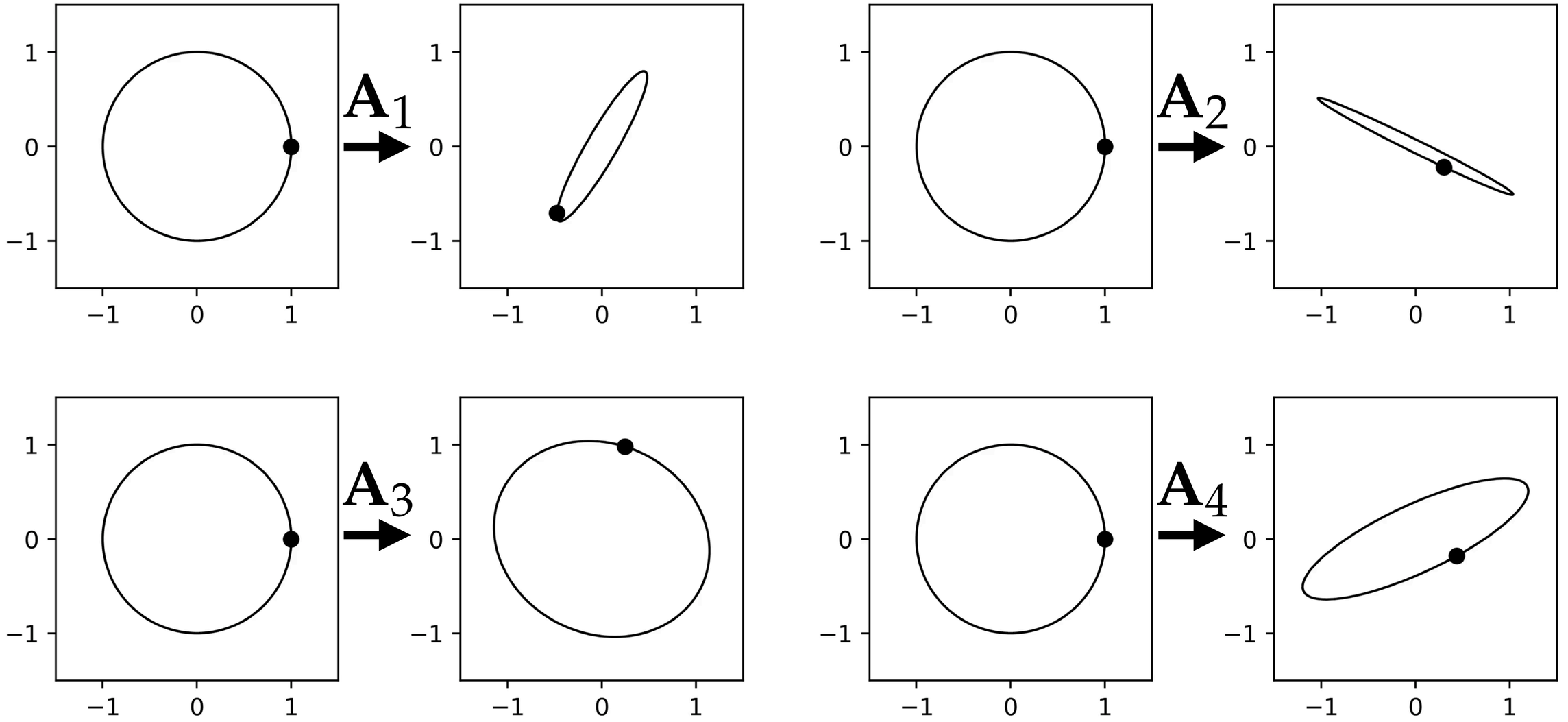
Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

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



Interlude: Matrix-vector multiplication, ellipses, and ellipsoids

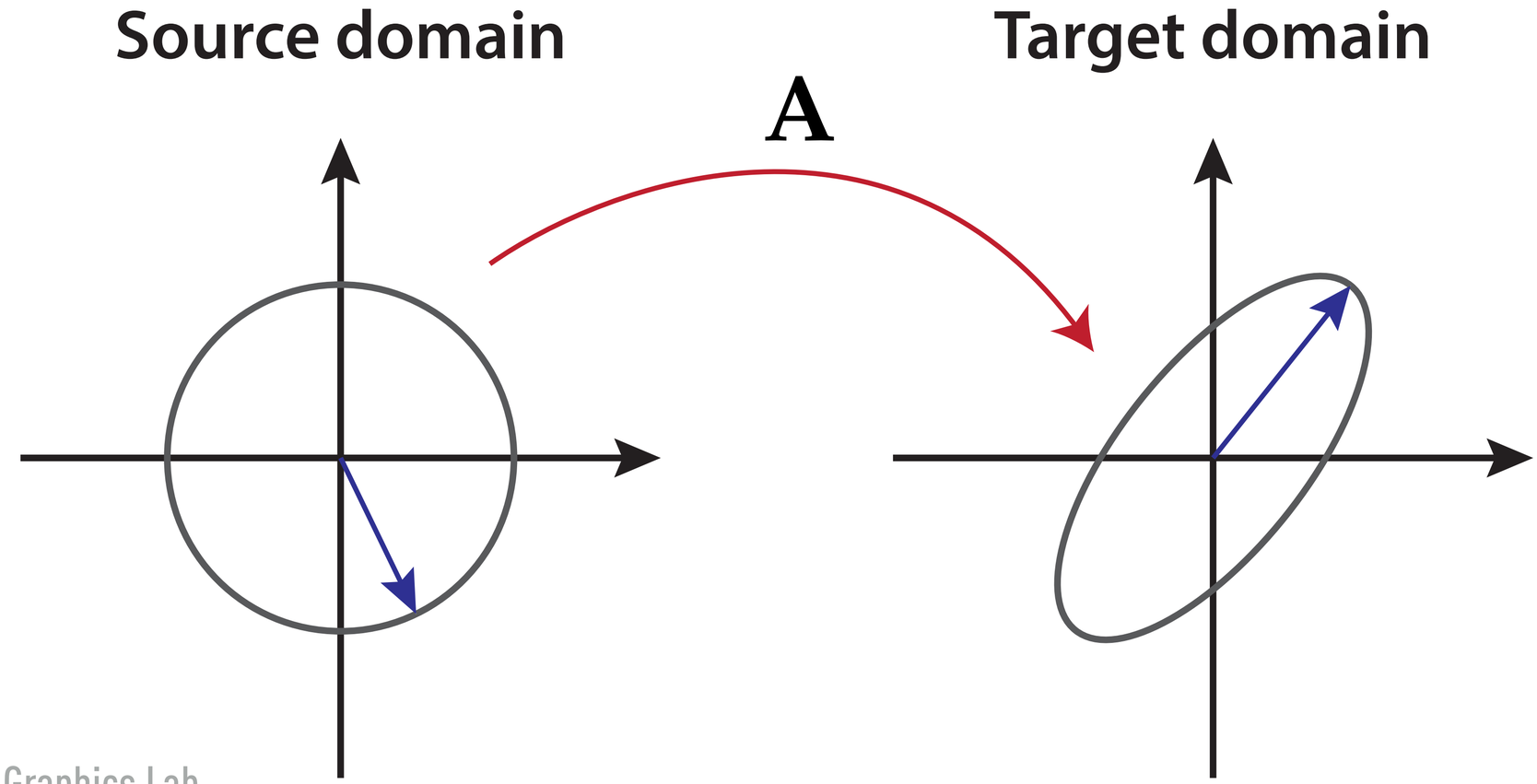
Four randomly generated matrices — how do they transform points on a circle?



The matrix norm (aka. operator norm)

Determines the maximum radius of this ellipse

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



Some inequalities

$$||\mathbf{Ax}|| \leq ||\mathbf{A}||||\mathbf{x}||$$

Some inequalities

$$||\mathbf{Ax}|| \leq ||\mathbf{A}||\,||\mathbf{x}||$$

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

Some inequalities

$$\|\mathbf{Ax}\| \leq \|\mathbf{A}\|\|\mathbf{x}\|$$

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

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$$

Some inequalities

$$||\mathbf{Ax}|| \leq ||\mathbf{A}||\,||\mathbf{x}||$$

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

$$||\mathbf{AB}|| \leq ||\mathbf{A}||\,||\mathbf{B}||$$

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

Some inequalities

$$\|\mathbf{Ax}\| \leq \|\mathbf{A}\|\|\mathbf{x}\|$$

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"

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$$

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

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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)*

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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: $\|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}\|$

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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: $\|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|$

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

Ingredient 2: $$\|\Delta\mathbf{x}\| = \|\mathbf{A}^{-1}\Delta\mathbf{b}\|$$

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

Ingredient 2: $$\|\Delta\mathbf{x}\| = \|\mathbf{A}^{-1}\Delta\mathbf{b}\| \leq \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{b}\|$$

Solving a perturbed linear system

Original system $\mathbf{A}\mathbf{x} = \mathbf{b}$

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

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

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

Solving a perturbed linear system

The condition number

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

Solving a perturbed linear system

The condition number

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

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

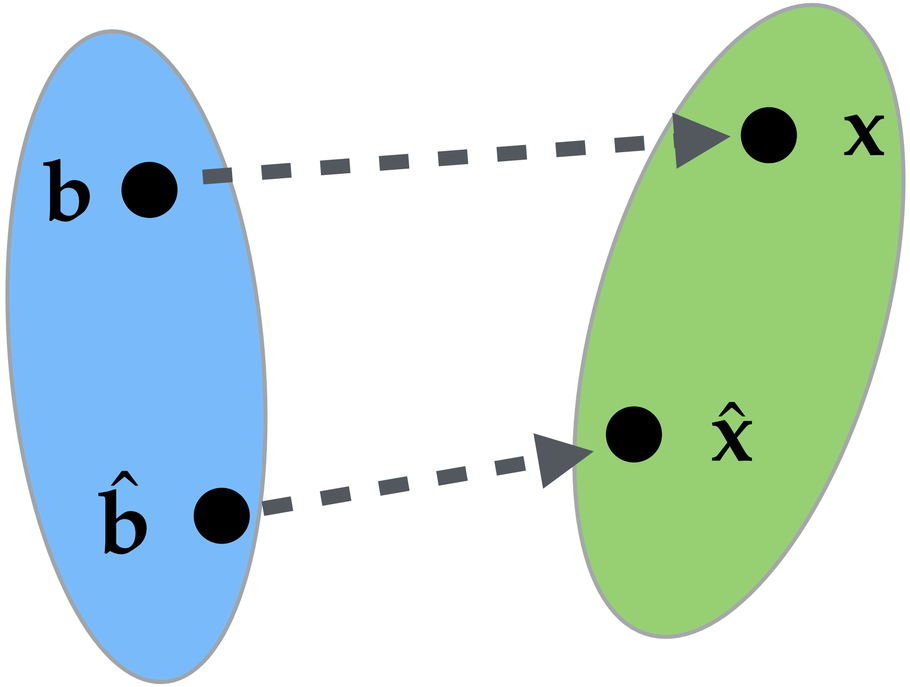
Solving a perturbed linear system

The condition number

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

Solving a perturbed linear system

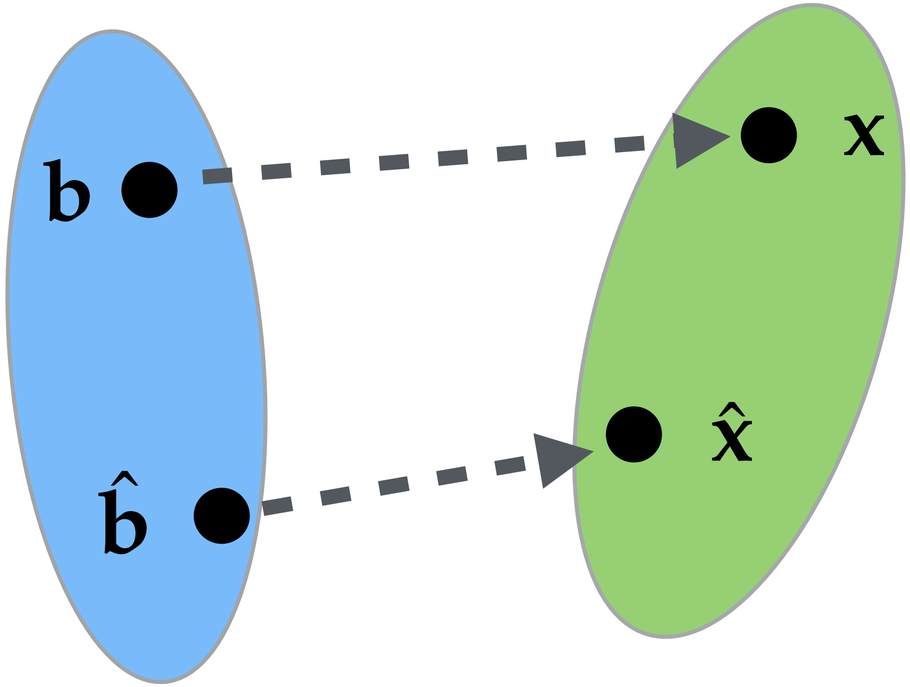
The condition number



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

Solving a perturbed linear system

The condition number



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

The condition number

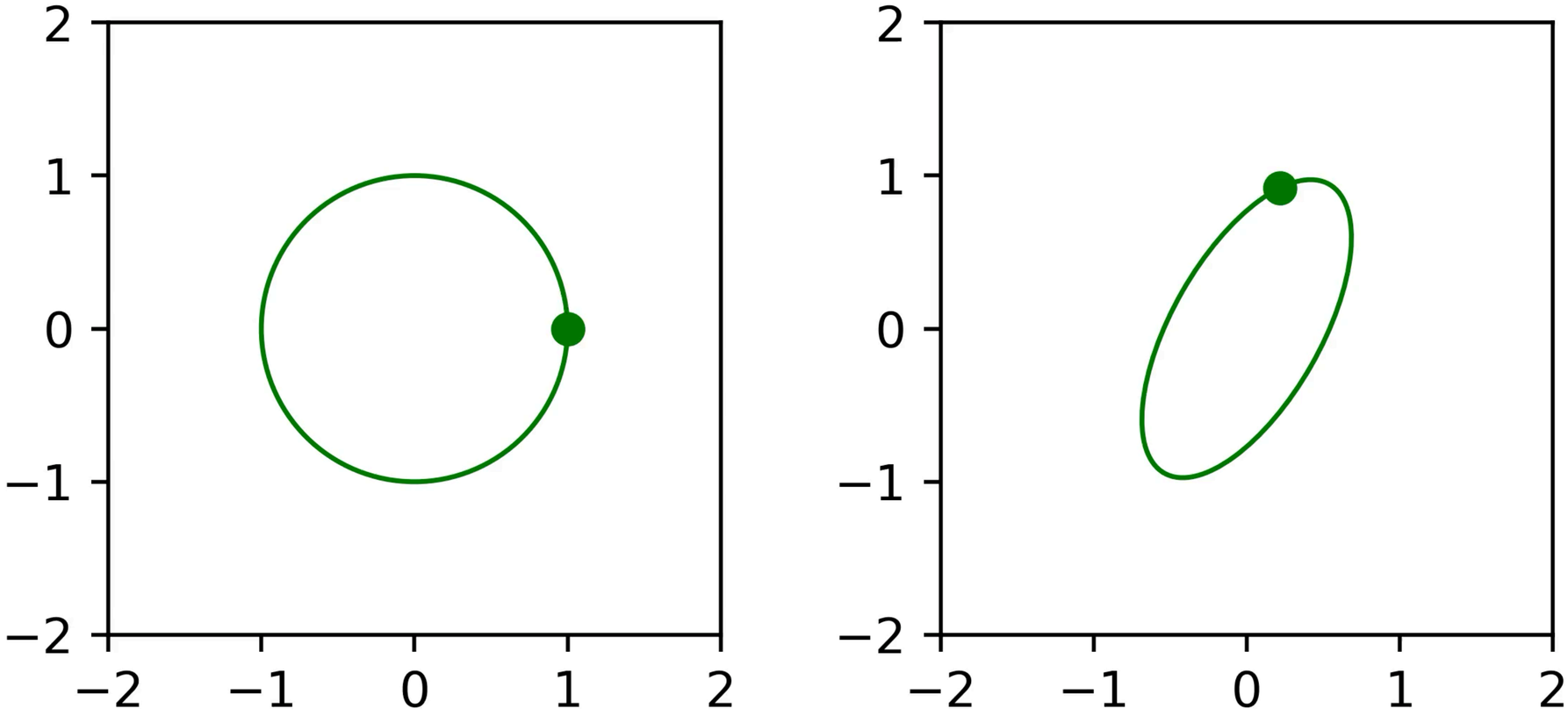
What does this number measure?

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

The condition number

What does this number measure?

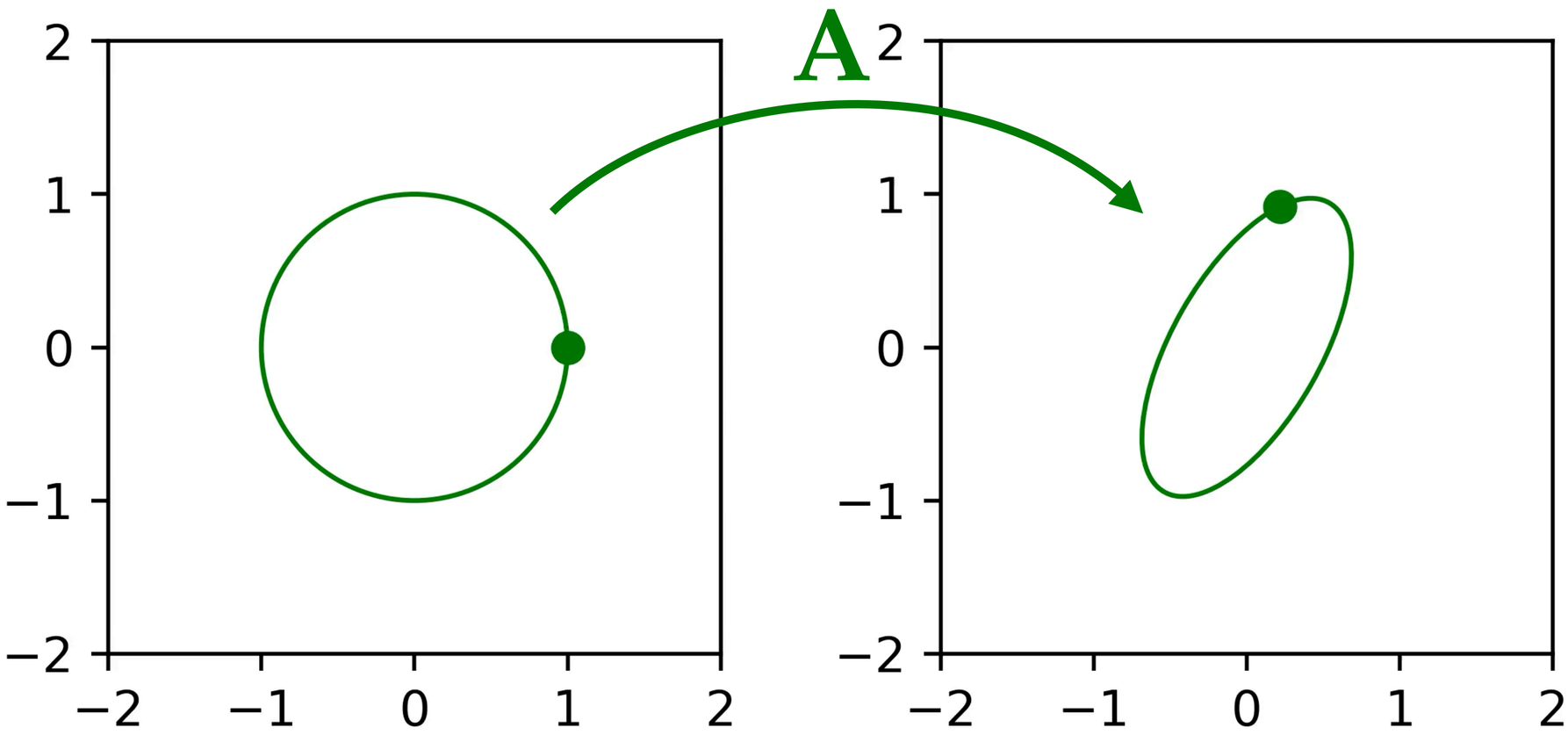
$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$



The condition number

What does this number measure?

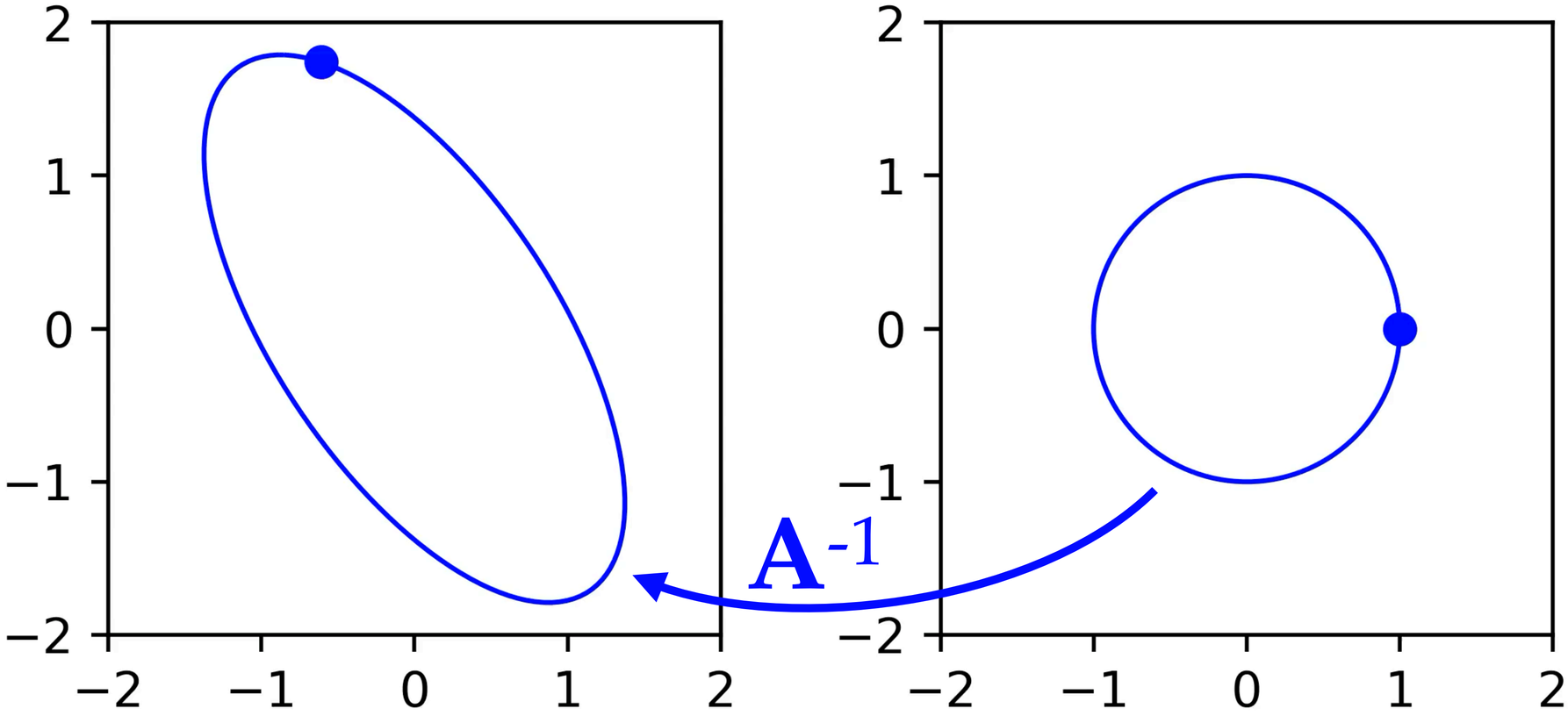
$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$



The condition number

What does this number measure?

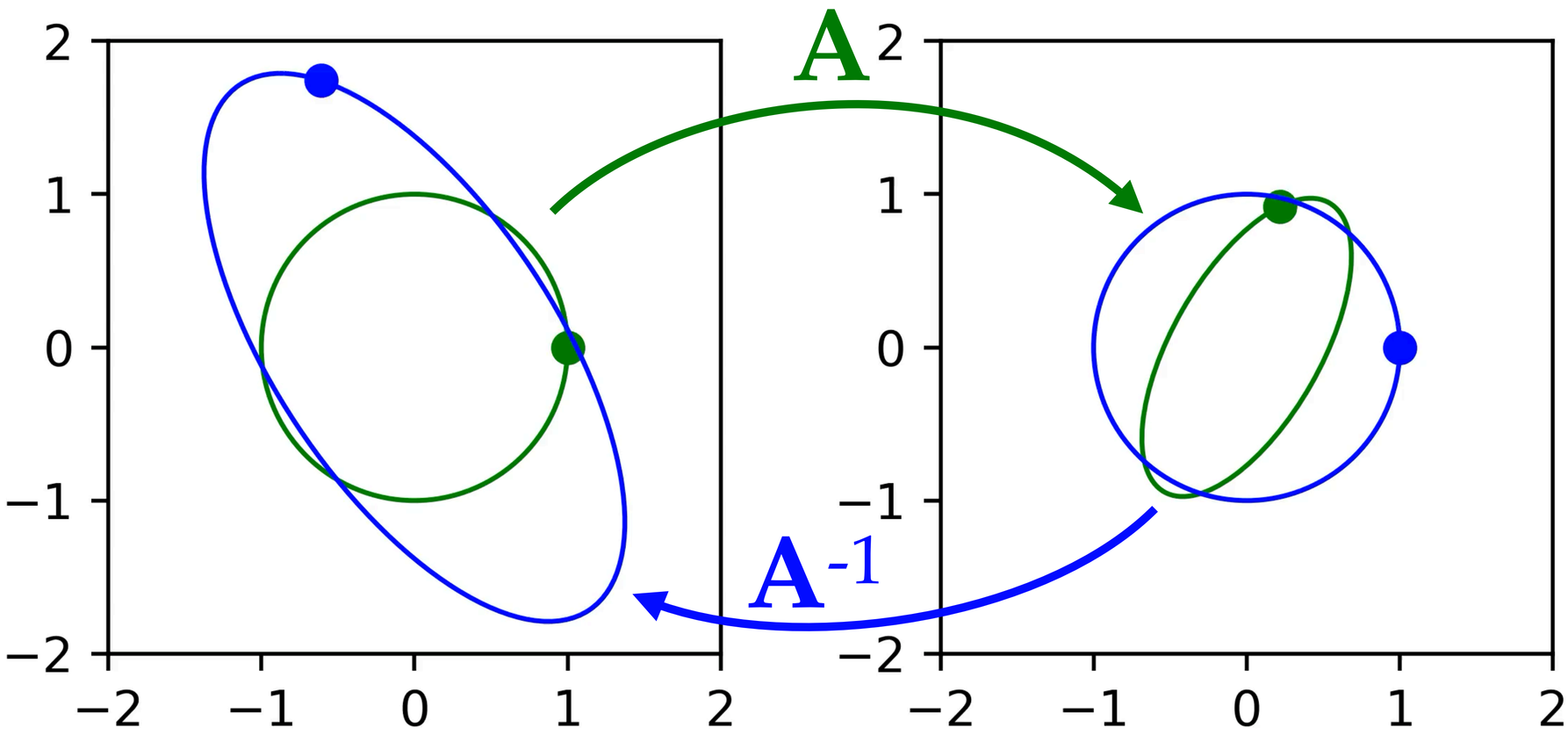
$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\|$$



The condition number

What does this number measure?

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



The condition number

Some useful properties

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

- $\mathrm{cond}(\mathbf{A}) = \infty$ iff $\mathbf{A}$ is singular
- $\mathrm{cond}(\mathbf{A}) = 1$ if $\mathbf{A}$ is the identity matrix
- $\mathrm{cond}(\gamma\mathbf{A}) = \mathrm{cond}(\mathbf{A}) \quad \gamma \in \mathbb{R}$
- $\mathrm{cond}(\mathbf{QA}) = \mathrm{cond}(\mathbf{A})$ 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.

.. 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 $\det(A) = 0$ *iff* A is singular.

.. 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 $\det(A) = 0$ *iff* A is singular.
- Let's try to set $\mathbf{A} = \frac{1}{10}\mathbf{I} \in \mathbb{R}^{1000 \times 1000}$

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

.. 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 $\det(A) = 0$ *iff* A is singular.
- Let's try to set $\mathbf{A} = \frac{1}{10}\mathbf{I} \in \mathbb{R}^{1000 \times 1000}$
- Then $\det(A) = 10^{-1000}$. 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

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

(*rel. error of solution*)

Implications

Stated without proof.

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

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

(*rel. error of solution*)

Where 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?*

Midjourney: *chicken and egg problem*

- Our definition of $\mathrm{cond}(\mathbf{A})$ requires inverting a matrix and computing a matrix norm (*how*?!)

A Chicken-and-Egg problem

- Inverting matrices is expensive.
- We want to know the condition number to find out if this is *even possible?*

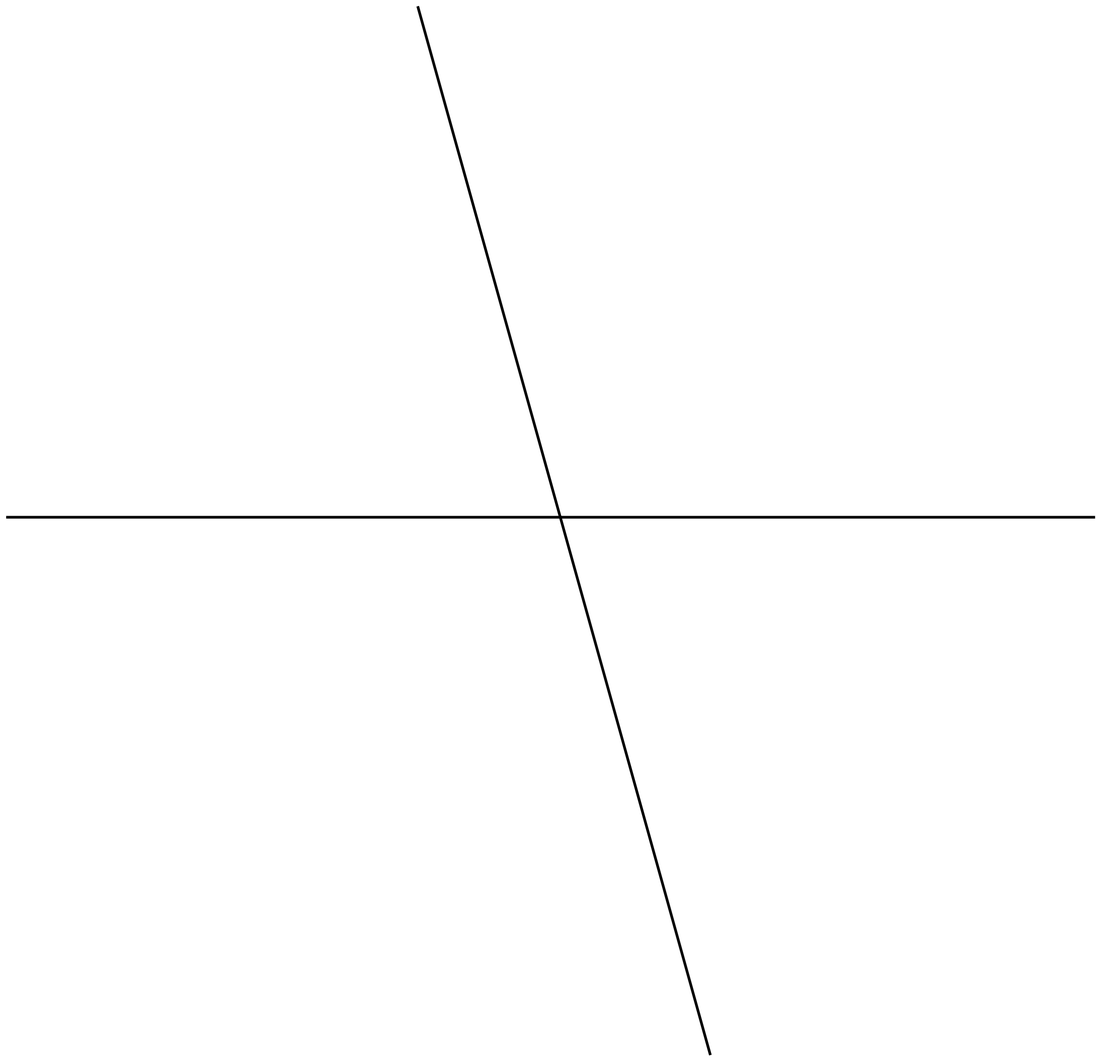
Midjourney: *chicken and egg problem*

- Our definition of $\mathrm{cond}(\mathbf{A})$ requires inverting a matrix and computing a matrix norm (*how*?!)

Stay tuned!

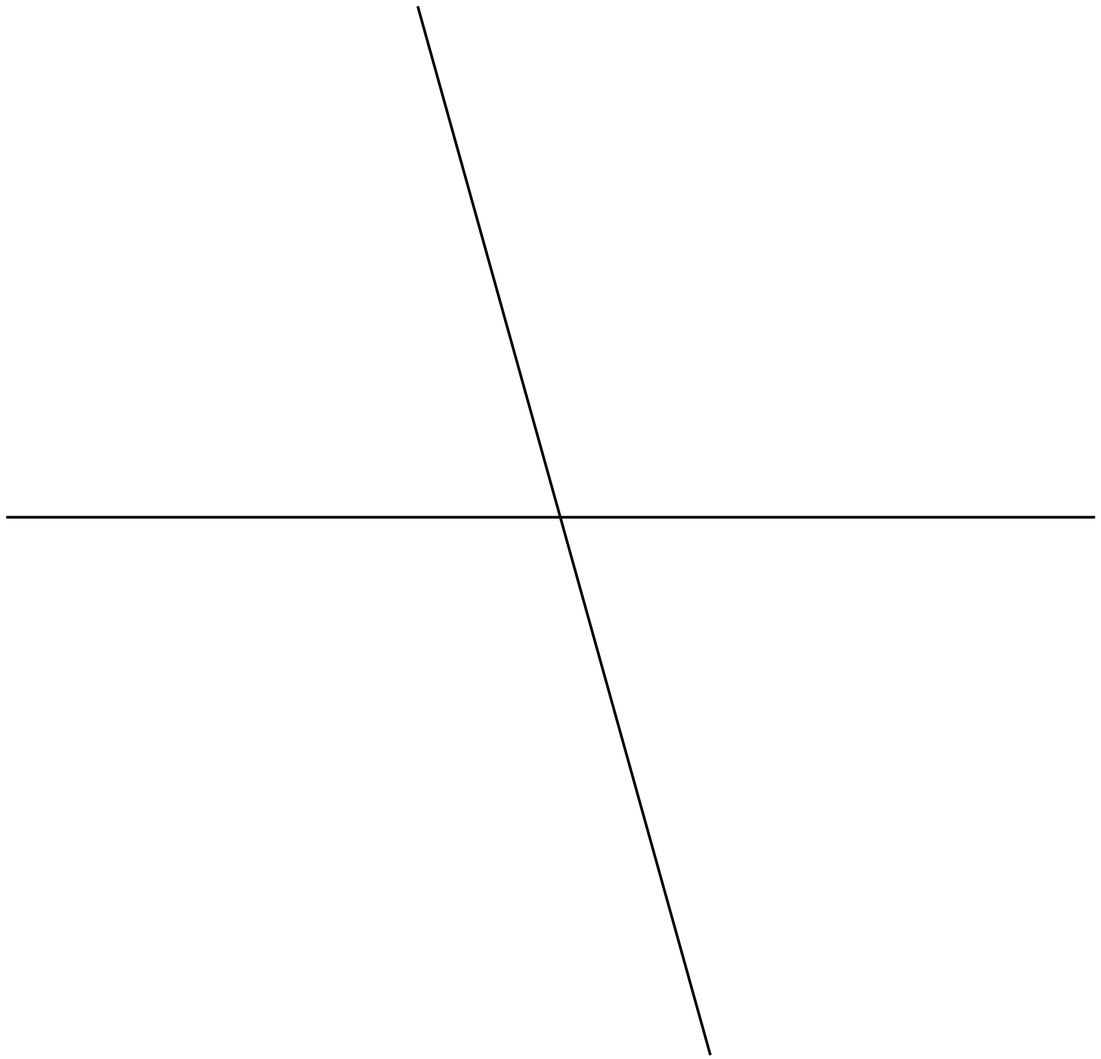
Geometry of badly conditioned systems

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



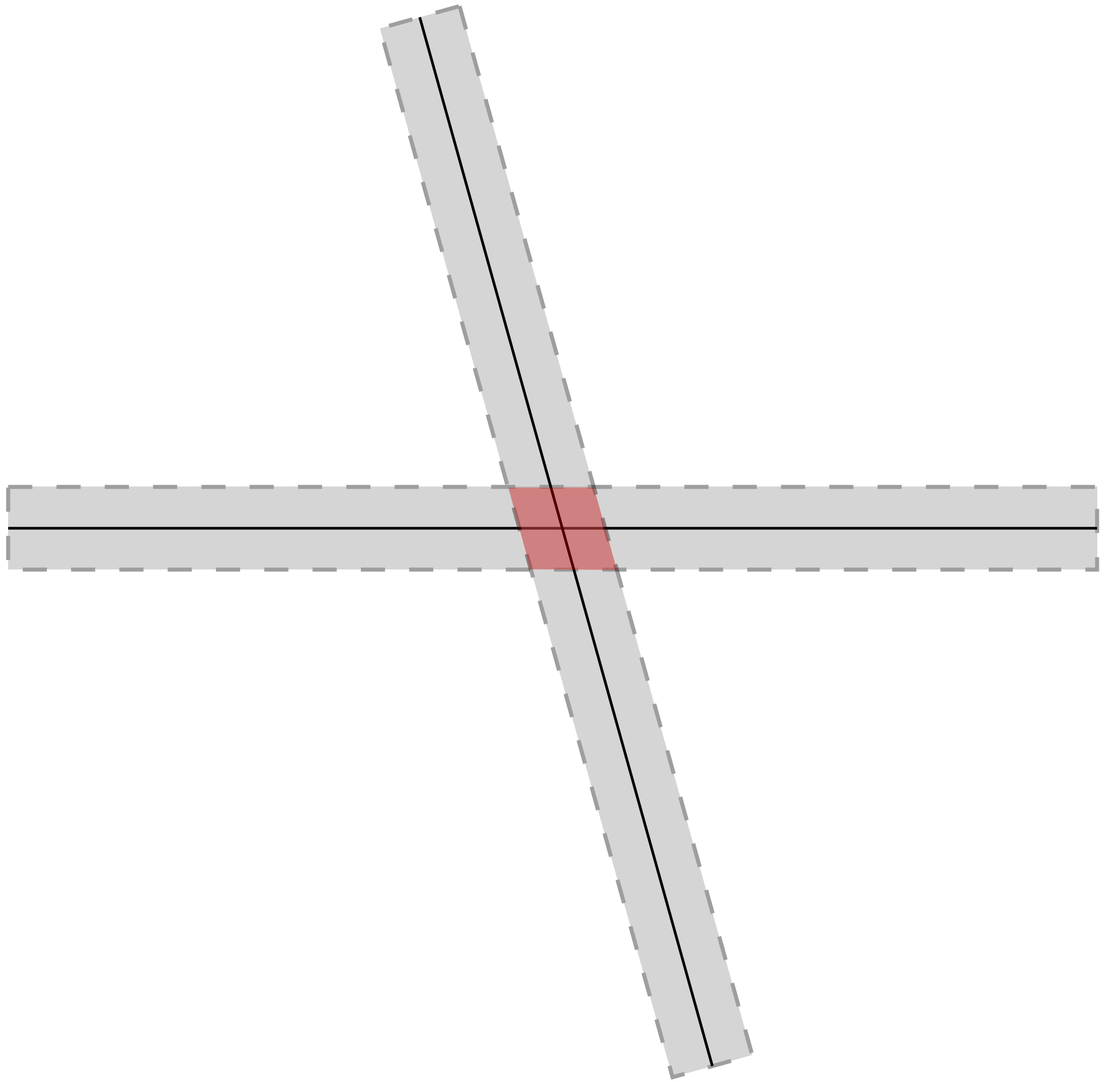
Geometry of badly conditioned systems

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



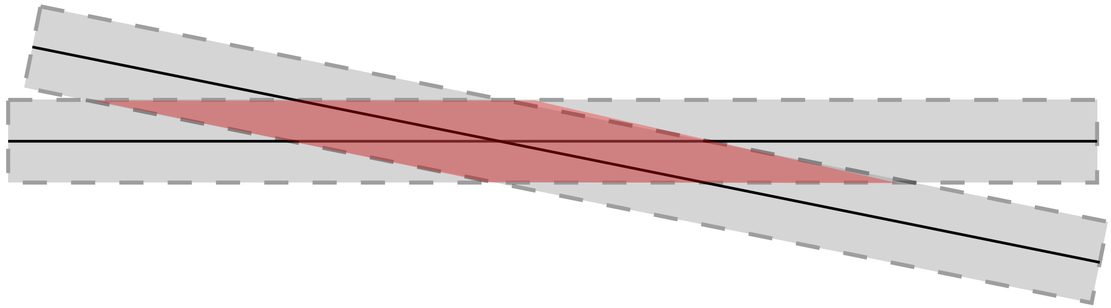
Geometry of badly conditioned systems

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



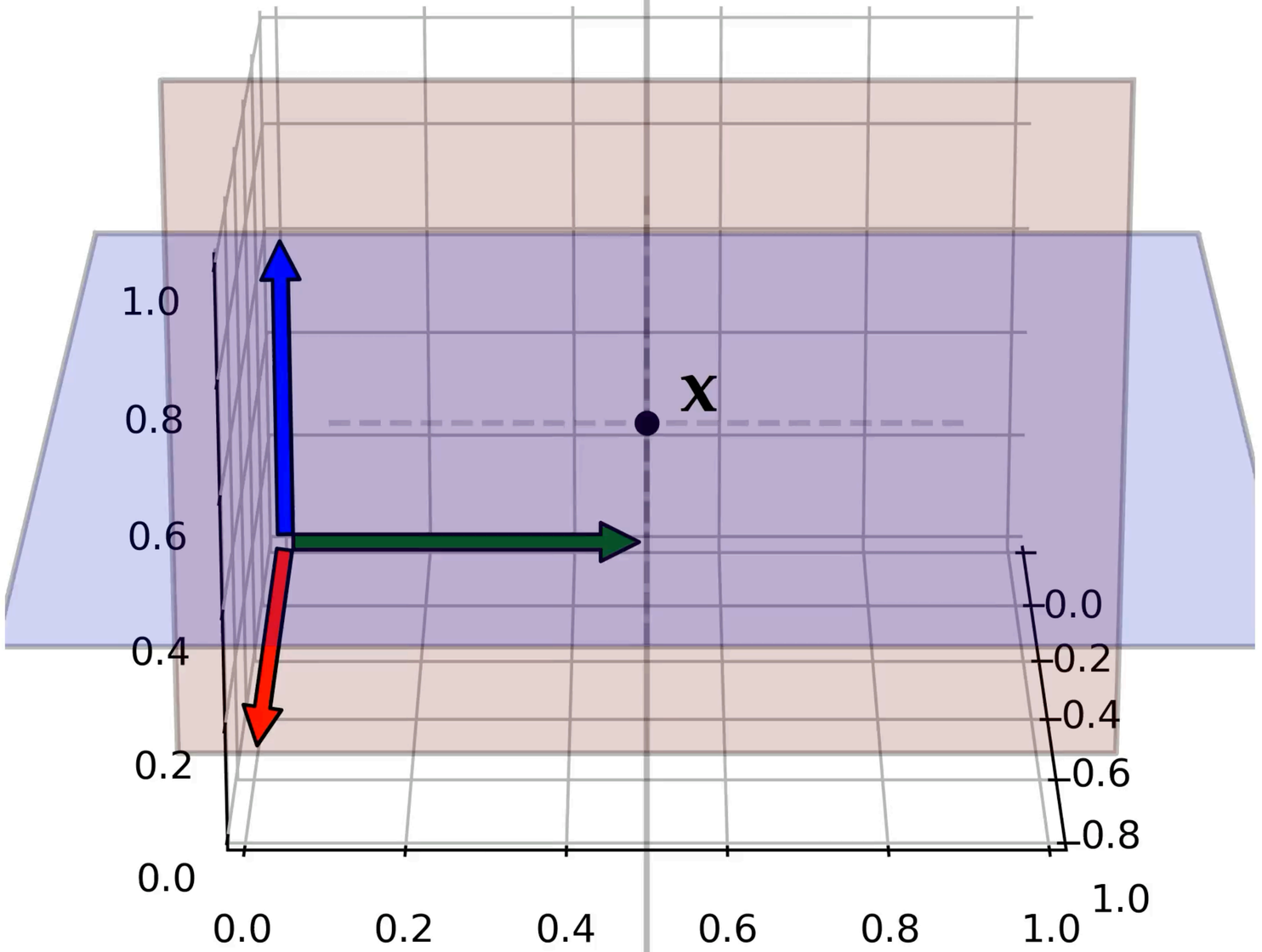
Geometry of badly conditioned systems

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



Geometric interpretation of linear systems

Row-based view



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

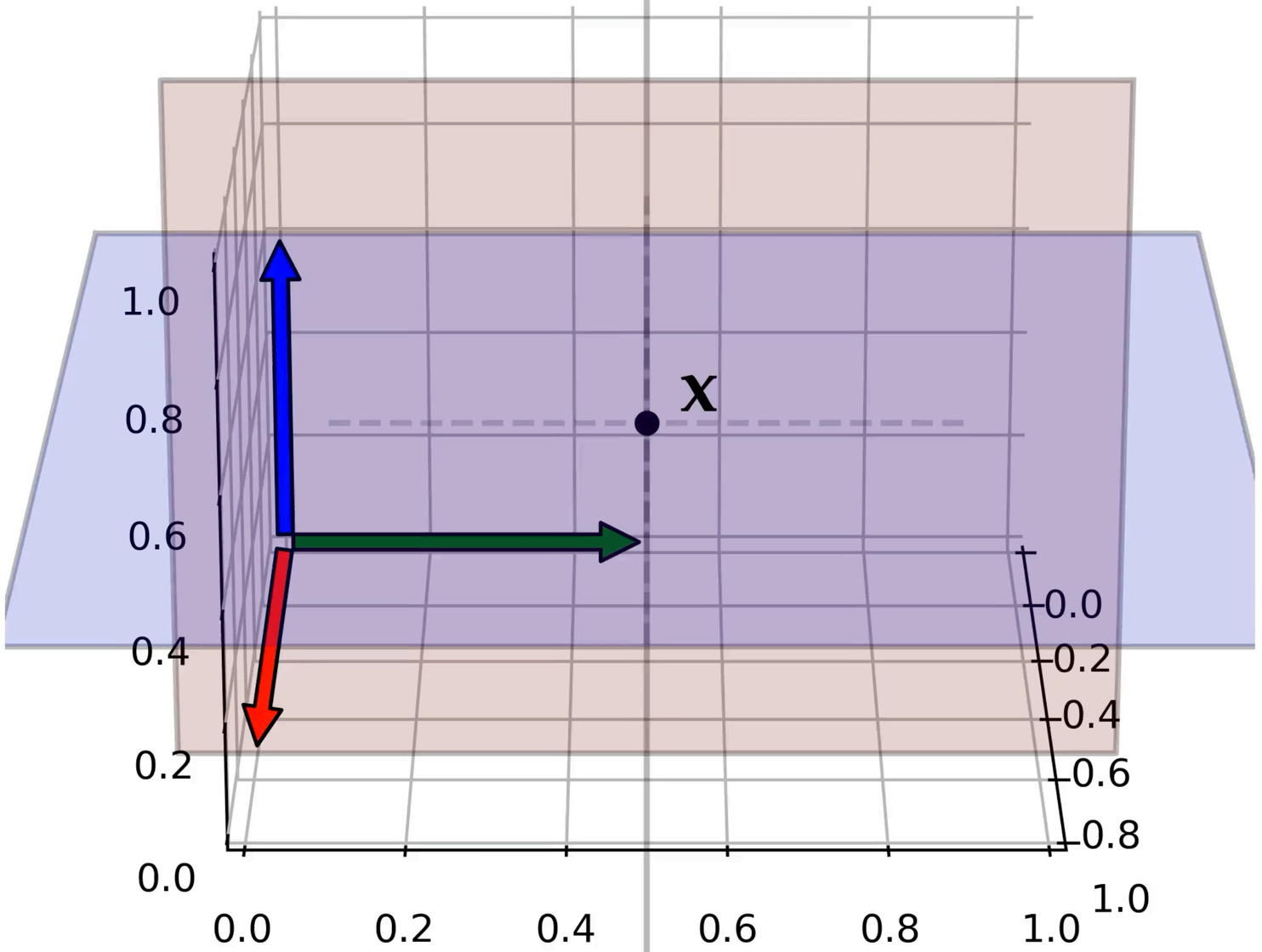
Interpretation: Compute the intersection of planes with normals $\mathbf{a}^{(i)}$ and offset b_i

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

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

Geometric interpretation of linear systems

Row-based view



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

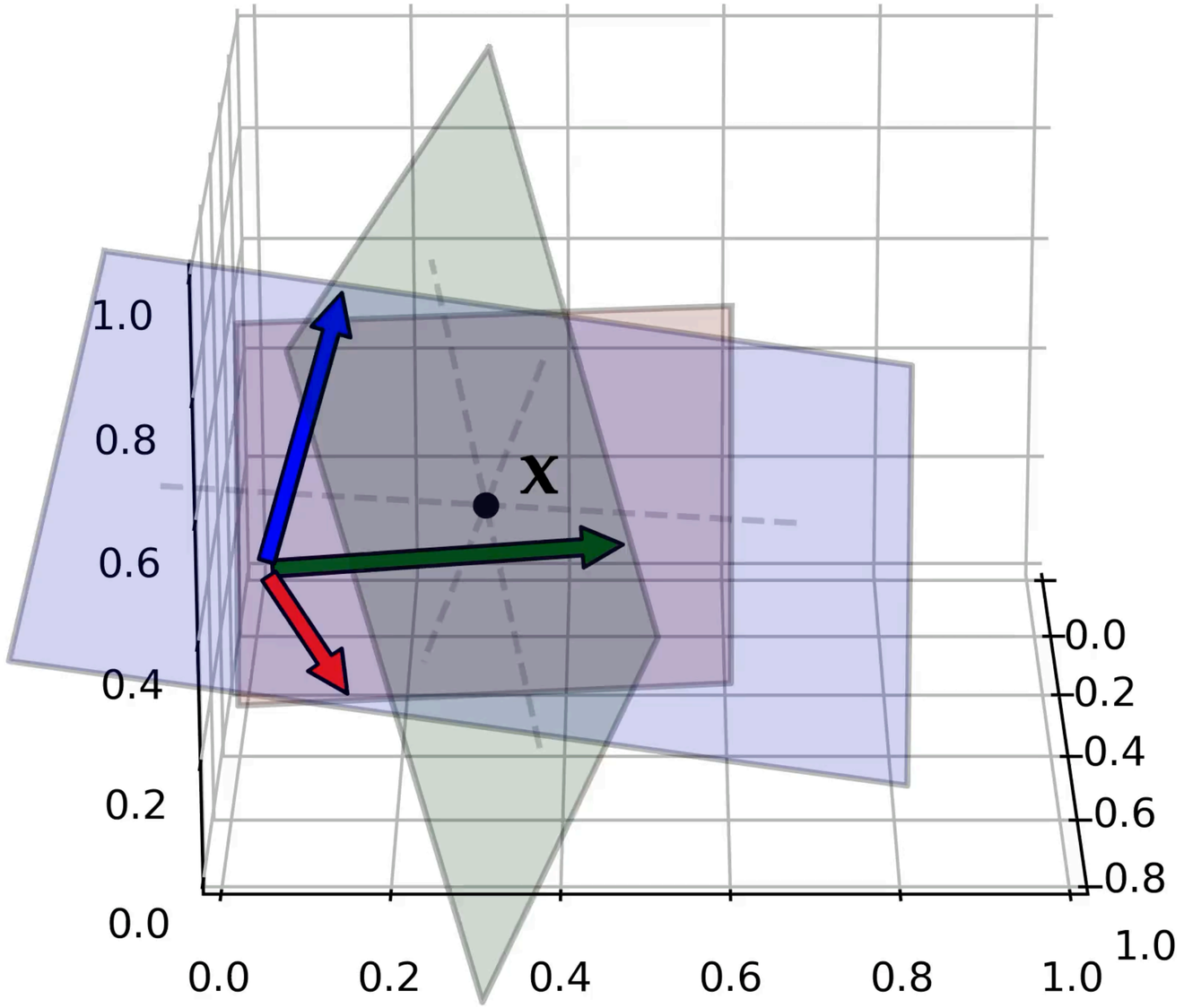
Interpretation: Compute the intersection of planes with normals $\mathbf{a}^{(i)}$ and offset b_i

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

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

Geometric interpretation of linear systems

Row-based view



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

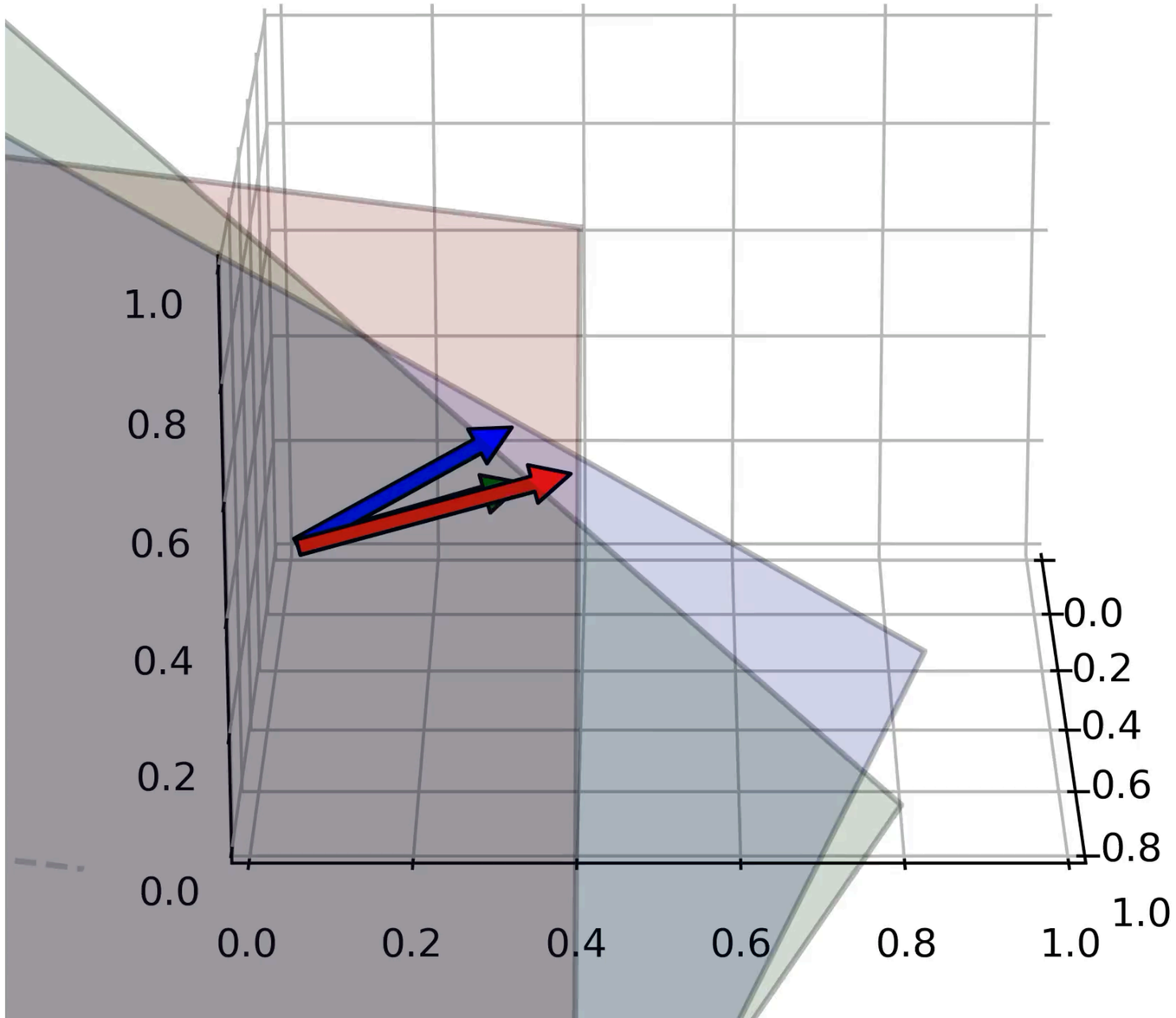
Interpretation: Compute the intersection of planes with normals $\mathbf{a}^{(i)}$ and offset b_i

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

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

Geometric interpretation of linear systems

Row-based view



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

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

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

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

Conditioning of the Vandermonde Matrix

Maps from polynomial coefficients to polynomial evaluations.

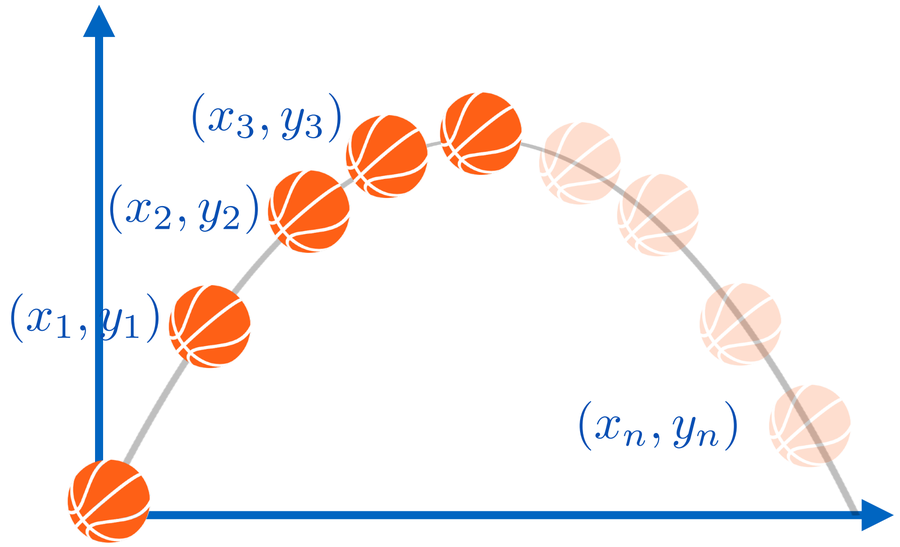
$$\mathbf{A} = \begin{pmatrix} \alpha_1^n & \dots & \alpha_1^2 & \alpha_1 & 1 \\ \alpha_2^n & \dots & \alpha_2^2 & \alpha_2 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_3^n & \dots & \alpha_3^2 & \alpha_3 & 1 \\ \alpha_m^n & \dots & \alpha_m^2 & \alpha_m & 1 \end{pmatrix}$$

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

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

Linear Least Squares

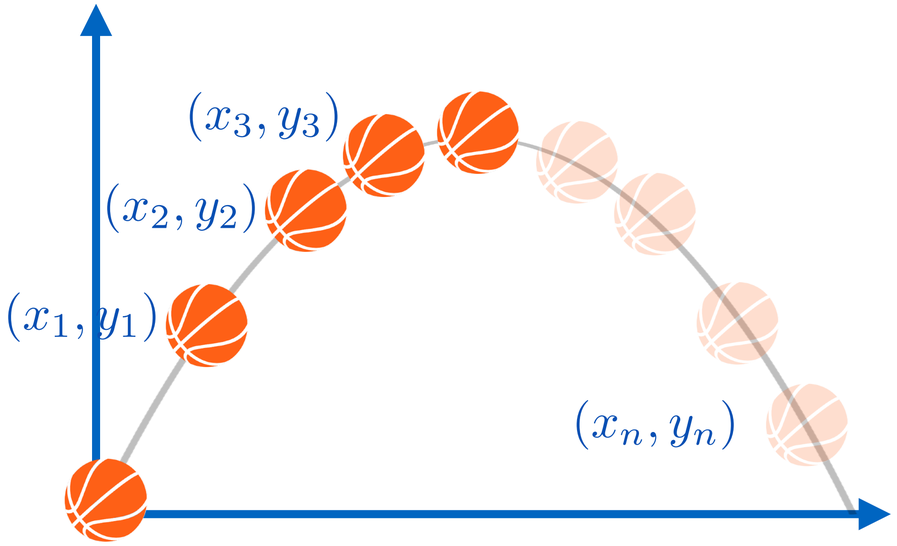
Linear Least Squares



What if there are > 3 observations?

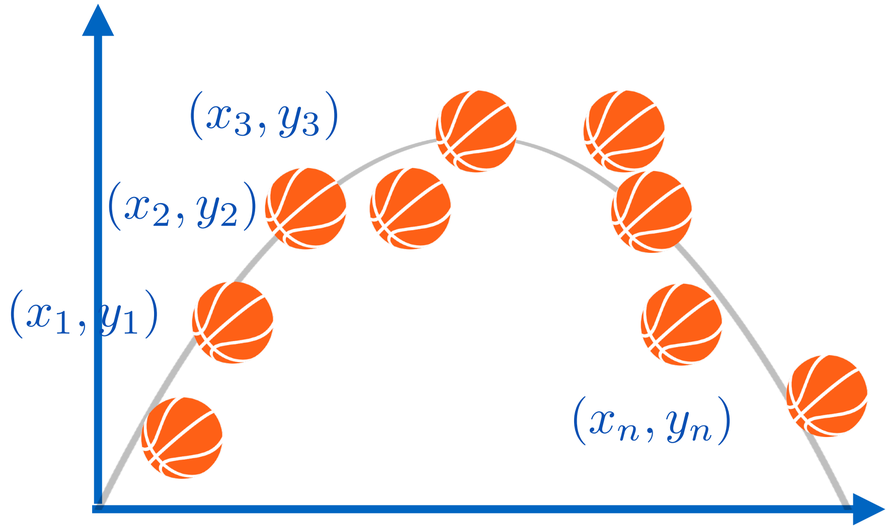
Use subset? Which ones? Use all somehow?

Linear Least Squares



What if there are > 3 observations?

Use subset? Which ones? Use all somehow?



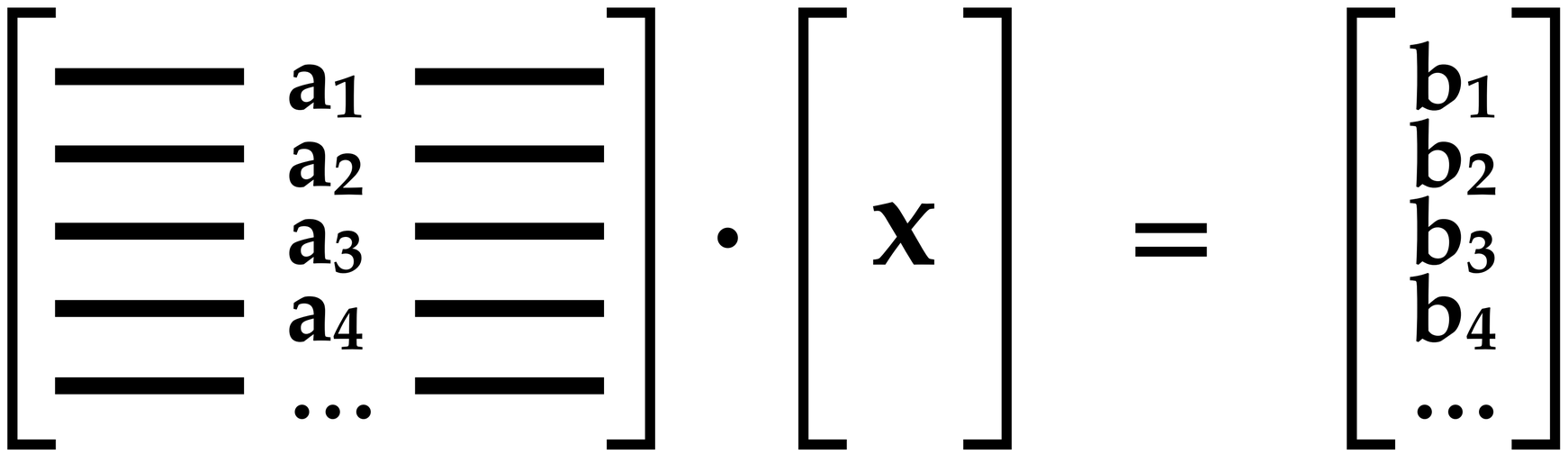
What if there is noise?

What if the model violates the assumptions?

From linear systems to least squares

$$\begin{bmatrix} & \mathbf{A} & \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \end{bmatrix}$$

From linear systems to least squares



n constraint equations (many)

m parameters (few)

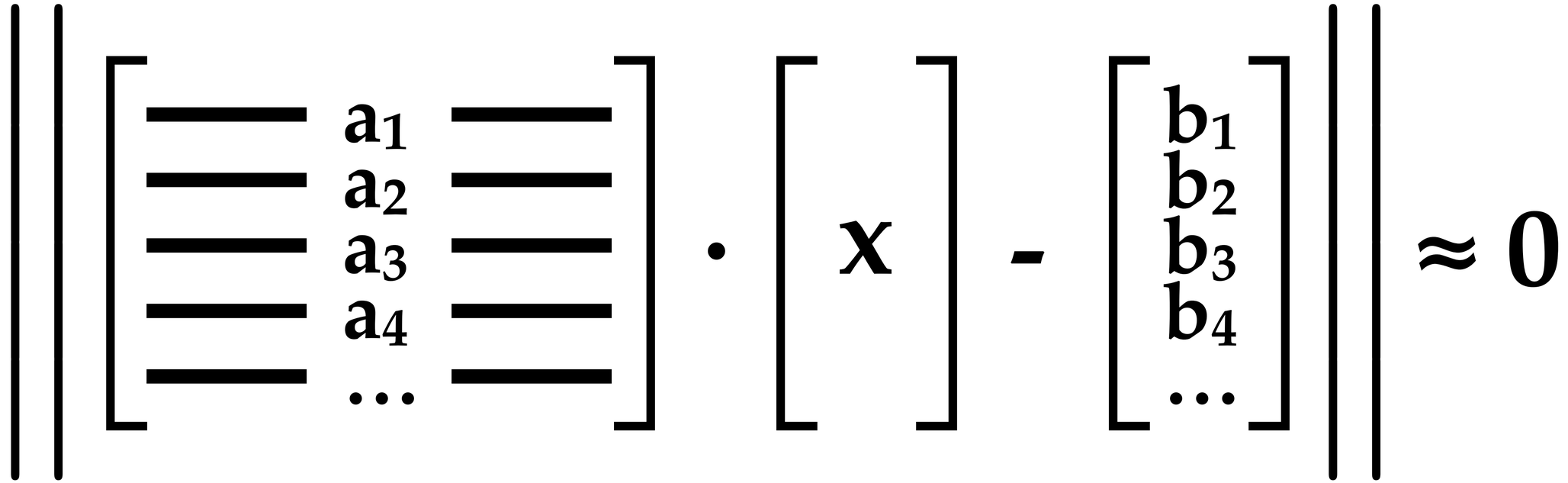
From linear systems to least squares

$$\begin{bmatrix} \text{—} & \mathbf{a}_1 & \text{—} \\ \text{—} & \mathbf{a}_2 & \text{—} \\ \text{—} & \mathbf{a}_3 & \text{—} \\ \text{—} & \mathbf{a}_4 & \text{—} \\ \text{—} & \dots & \text{—} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \end{bmatrix} - \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \\ \dots \end{bmatrix} \approx \mathbf{0}$$

n constraint equations (many)

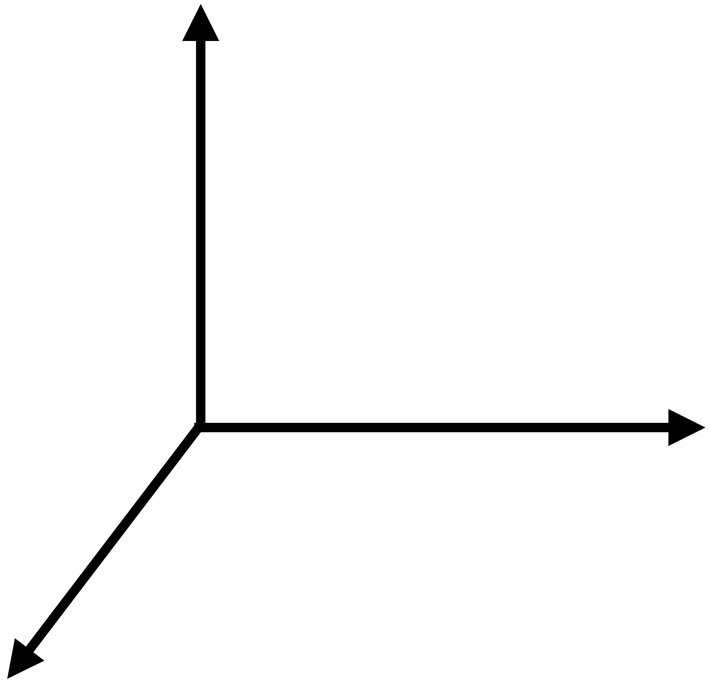
m parameters (few)

From linear systems to least squares



Geometric intuition

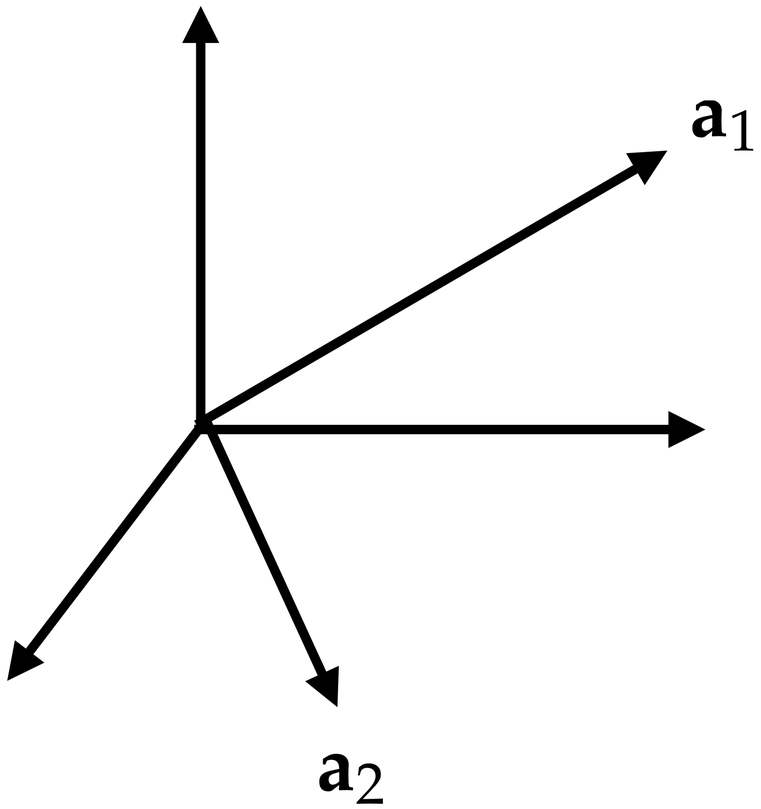
For almost all values of $\mathbf{b}$, $\mathbf{A}\mathbf{x} = \mathbf{b}$ is not solvable. Try to get as close as possible.



$$\mathbf{A} = \left(\begin{array}{cc} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{array} \right) \Bigg\} \text{ 3 rows}$$

Geometric intuition

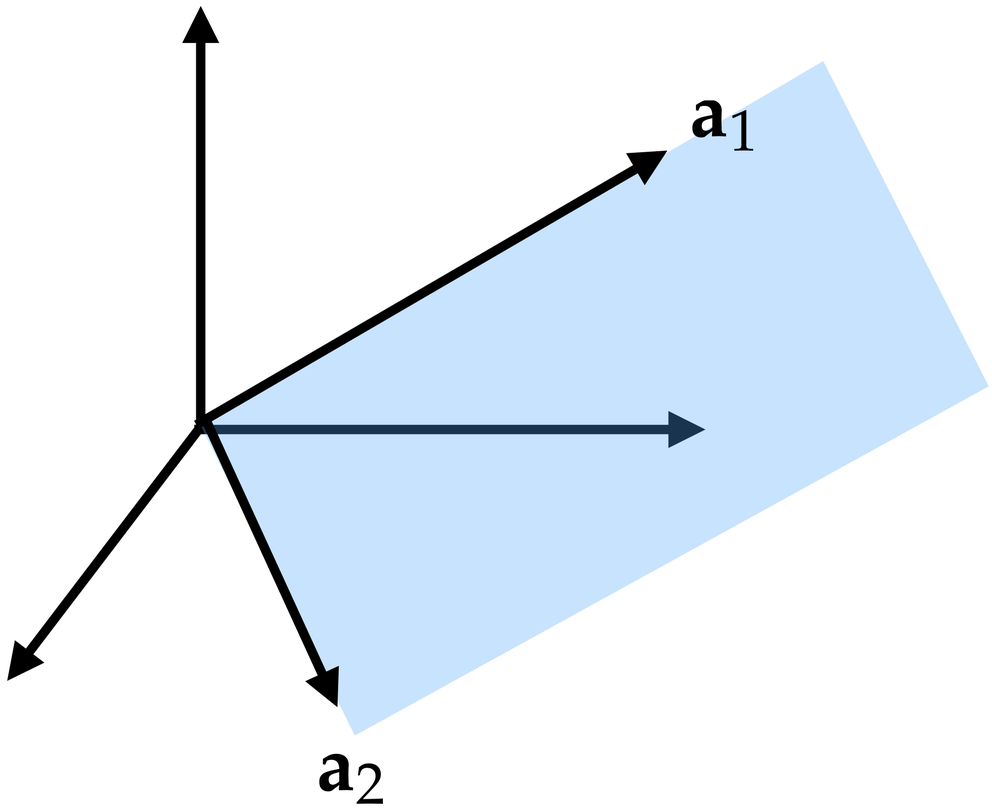
For almost all values of $\mathbf{b}$, $\mathbf{A}\mathbf{x} = \mathbf{b}$ is not solvable. Try to get as close as possible.



$$\mathbf{A} = \left(\begin{array}{cc} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{array} \right) \Bigg\} \text{ 3 rows}$$

Geometric intuition

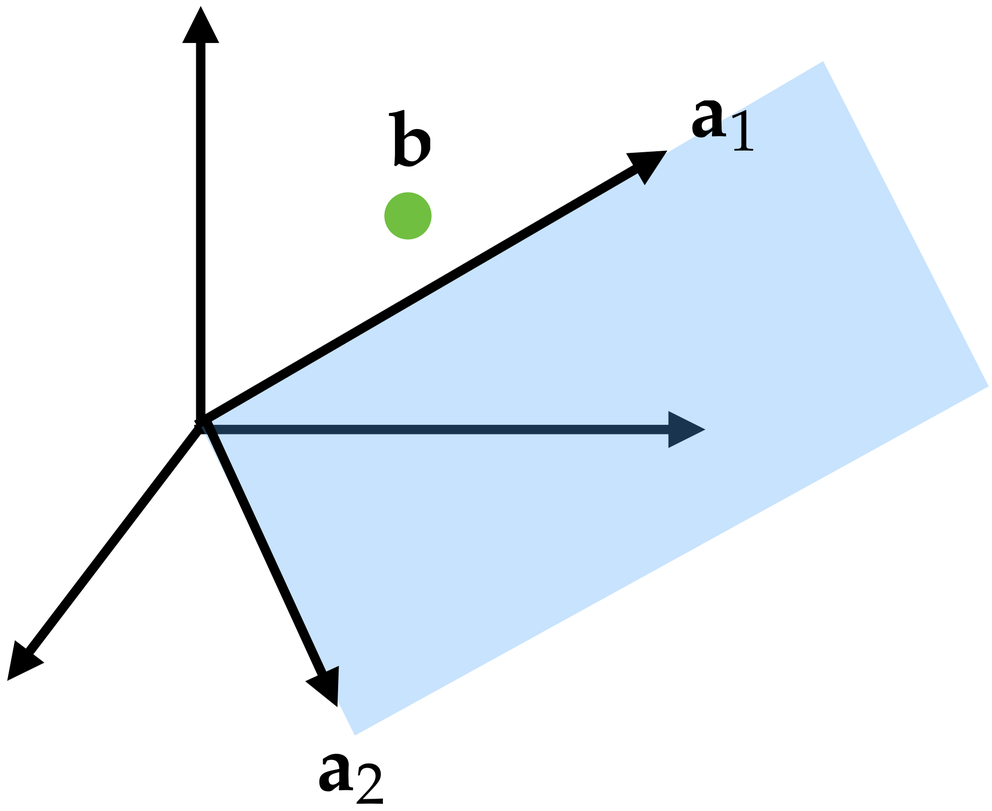
For almost all values of $\mathbf{b}$, $\mathbf{A}\mathbf{x} = \mathbf{b}$ is not solvable. Try to get as close as possible.



$$\mathbf{A} = \left(\begin{array}{cc} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{array} \right) \Bigg\} \text{ 3 rows}$$

Geometric intuition

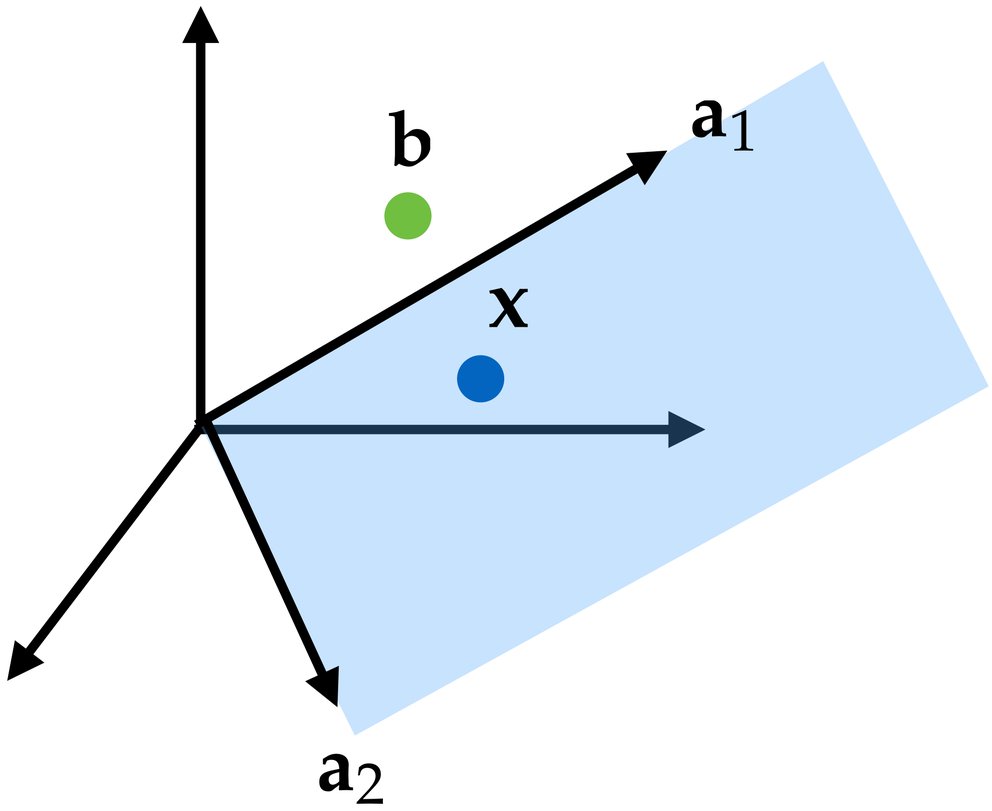
For almost all values of $\mathbf{b}$, $\mathbf{A}\mathbf{x} = \mathbf{b}$ is not solvable. Try to get as close as possible.



$$\mathbf{A} = \begin{pmatrix} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{pmatrix} \Big\} \text{ 3 rows}$$

Geometric intuition

For almost all values of $\mathbf{b}$, $\mathbf{A}\mathbf{x} = \mathbf{b}$ is not solvable. Try to get as close as possible.



$$\mathbf{A} = \begin{pmatrix} | & | \\ \mathbf{a}_1 & \mathbf{a}_2 \\ | & | \end{pmatrix} \Big\} \text{ 3 rows}$$

Compromise: minimize the residual

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

Compromise: minimize the residual

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

Game plan:

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

Compromise: minimize the residual

Want: $\|\mathbf{r}(\mathbf{x})\| = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$ 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:

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

$$= 2\mathbf{A}^T \mathbf{A}\mathbf{x} - 2\mathbf{A}^T \mathbf{b}$$

Compromise: minimize the residual

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

Game plan:

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

$$2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}$$

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

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

Compromise: minimize the residual

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

Game plan:

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

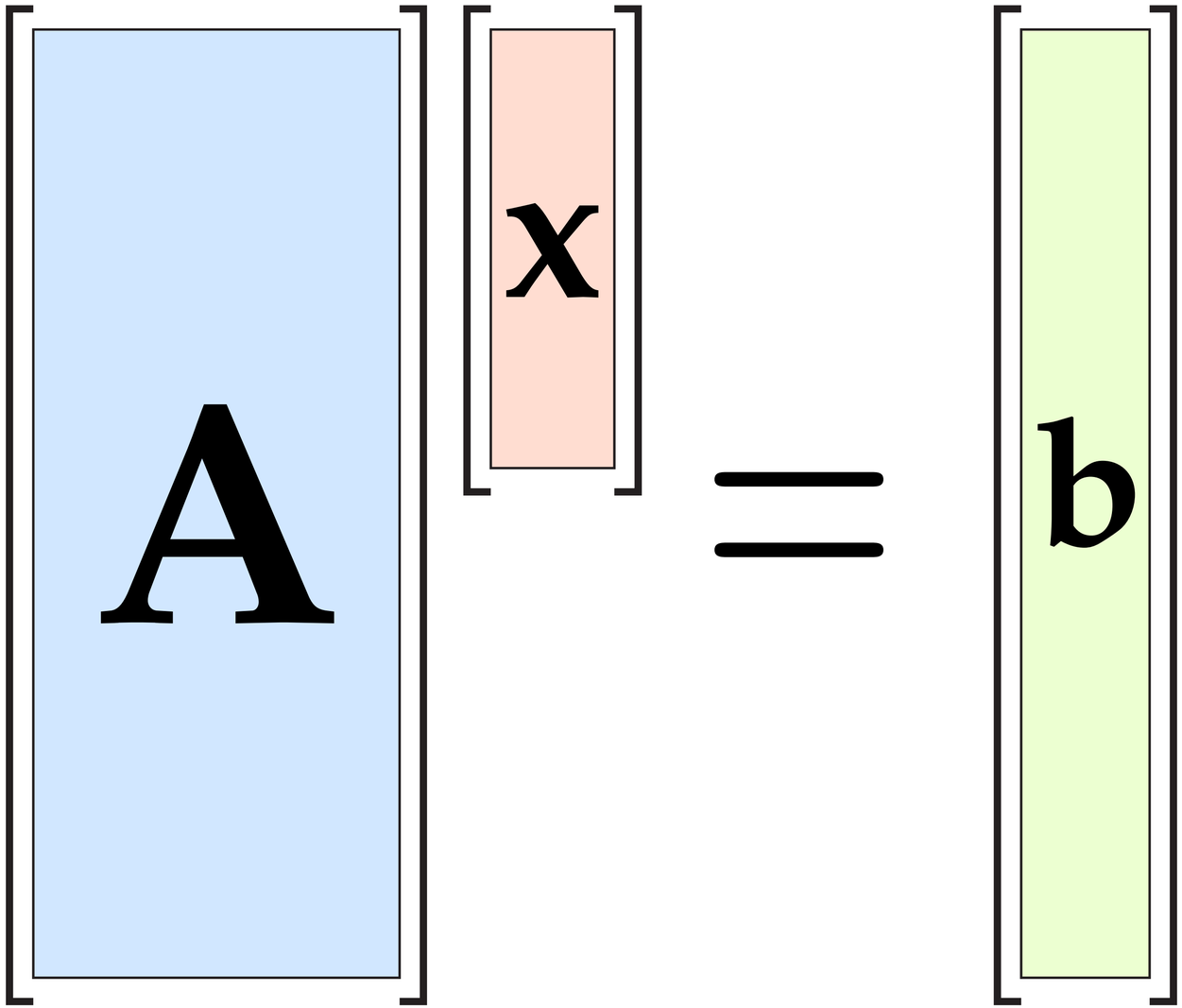
Carl Friedrich Gauss

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

Normal equations

Tall linear system

Generally ill-defined



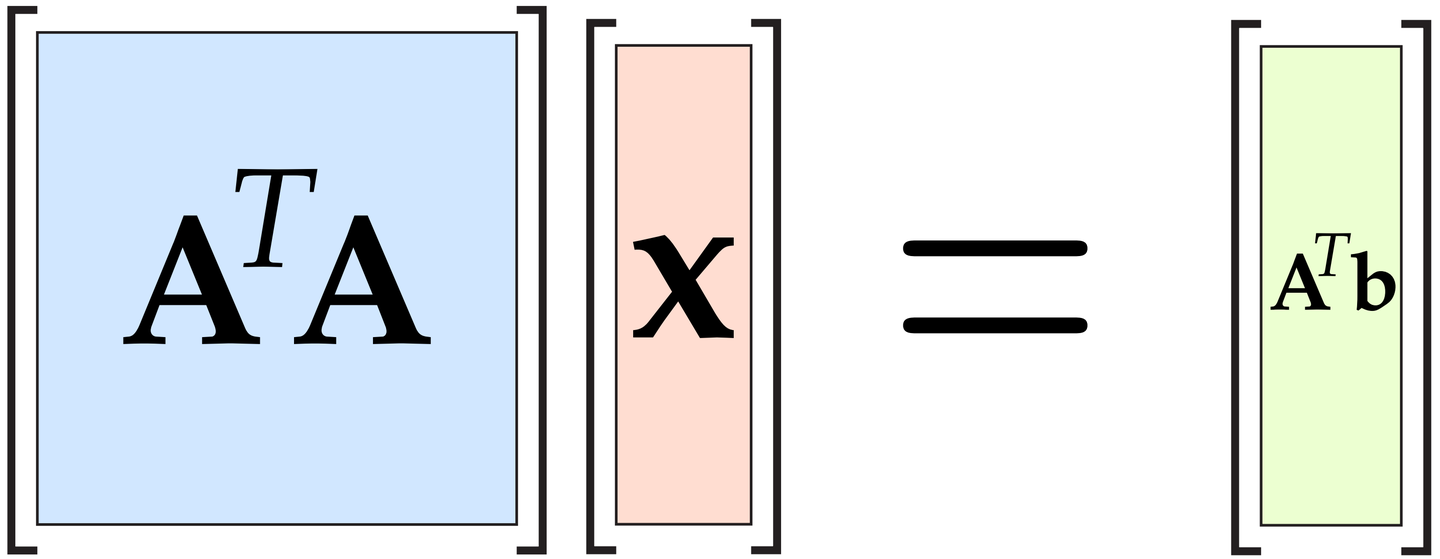
Linear least squares via the normal equations

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

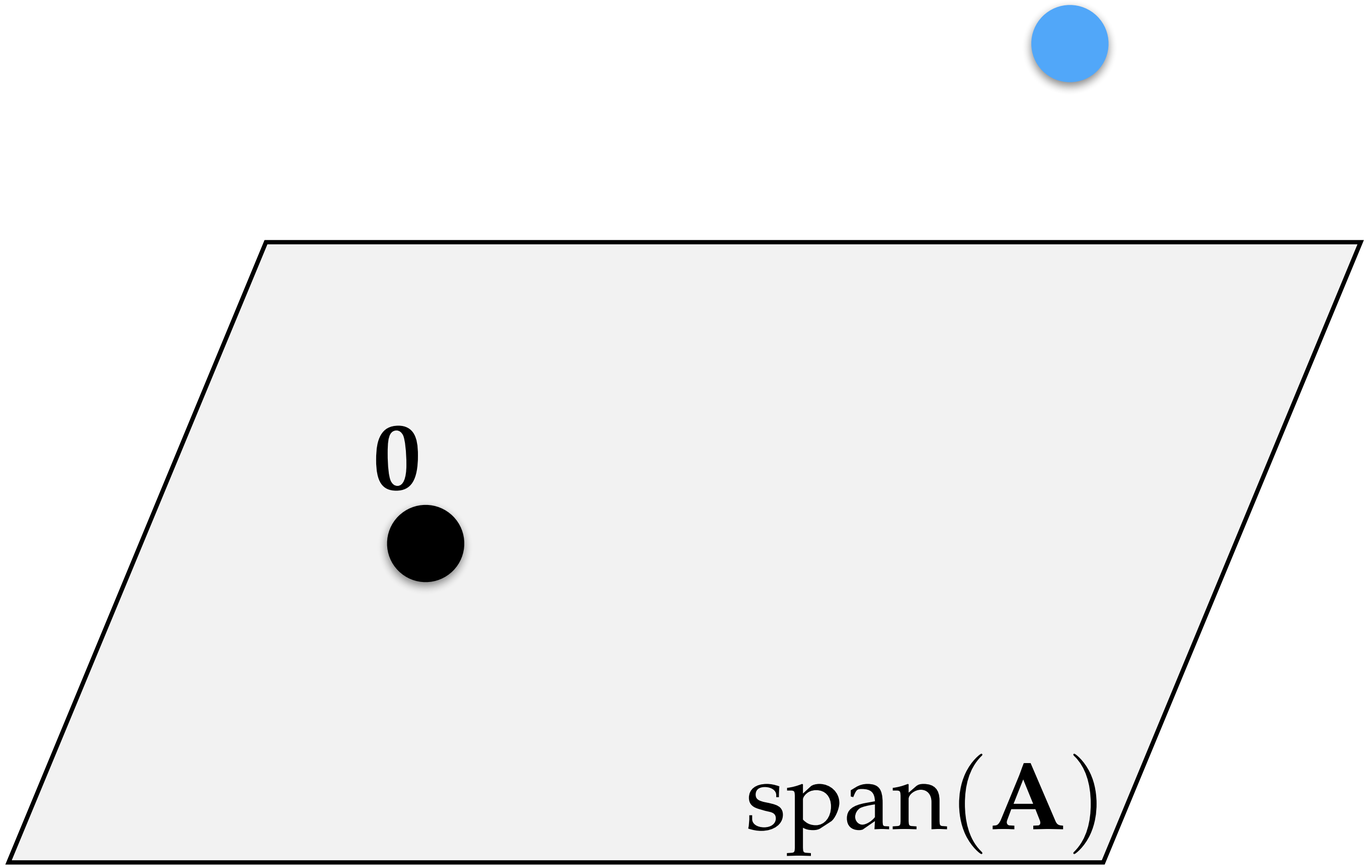
$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$$

Linear least squares via the normal equations

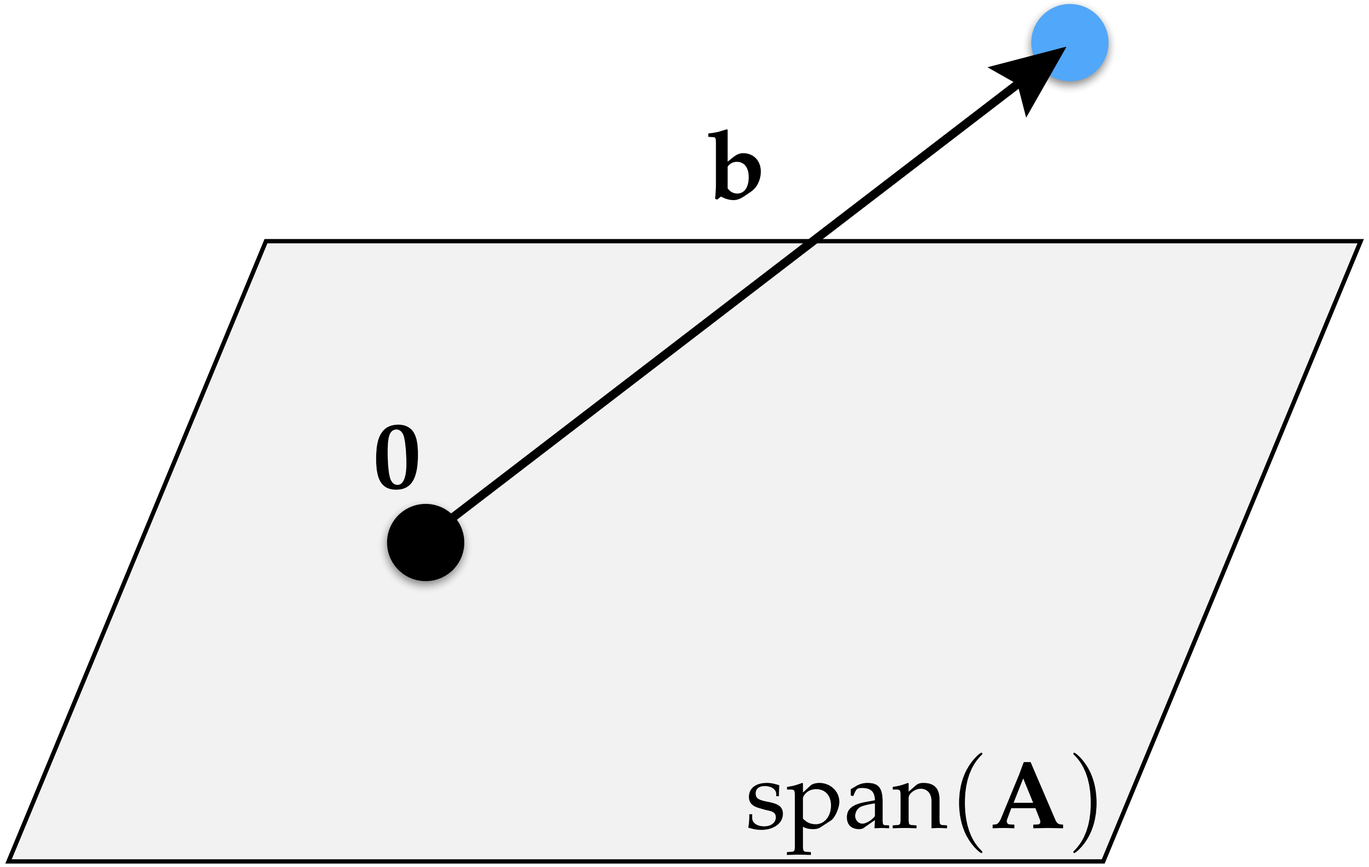
.. which is equivalent to a smaller square linear system



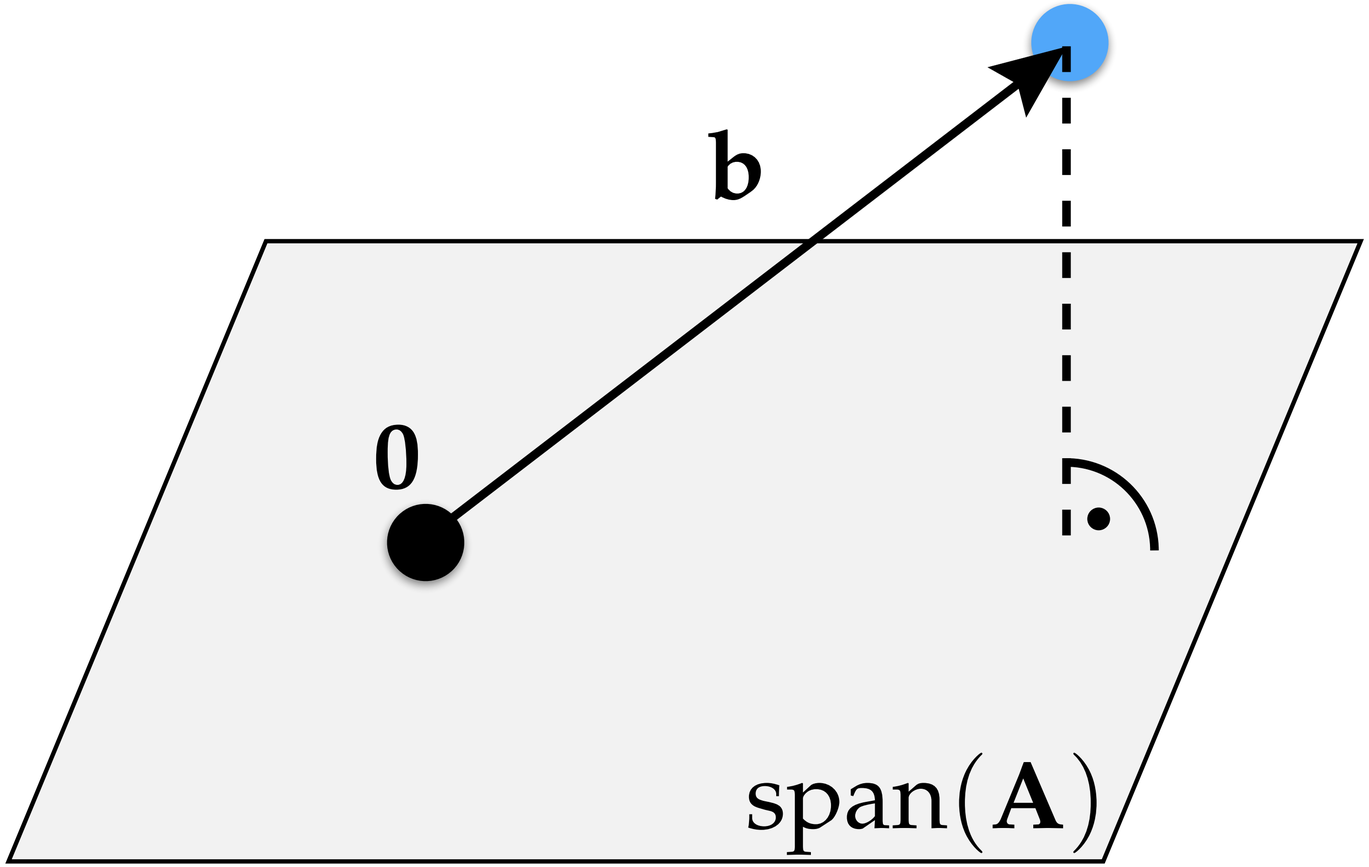
Why are they called normal equations?



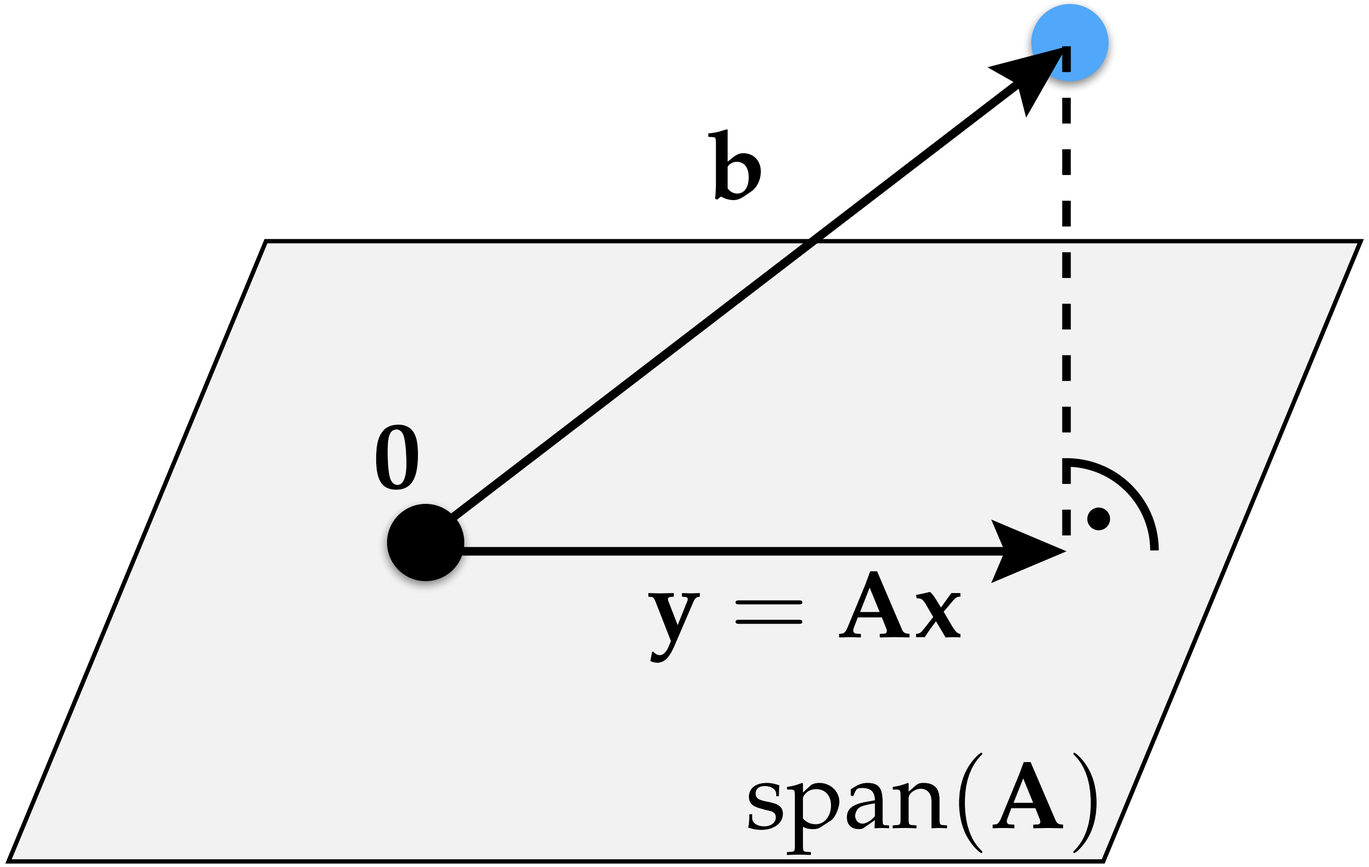
Why are they called normal equations?



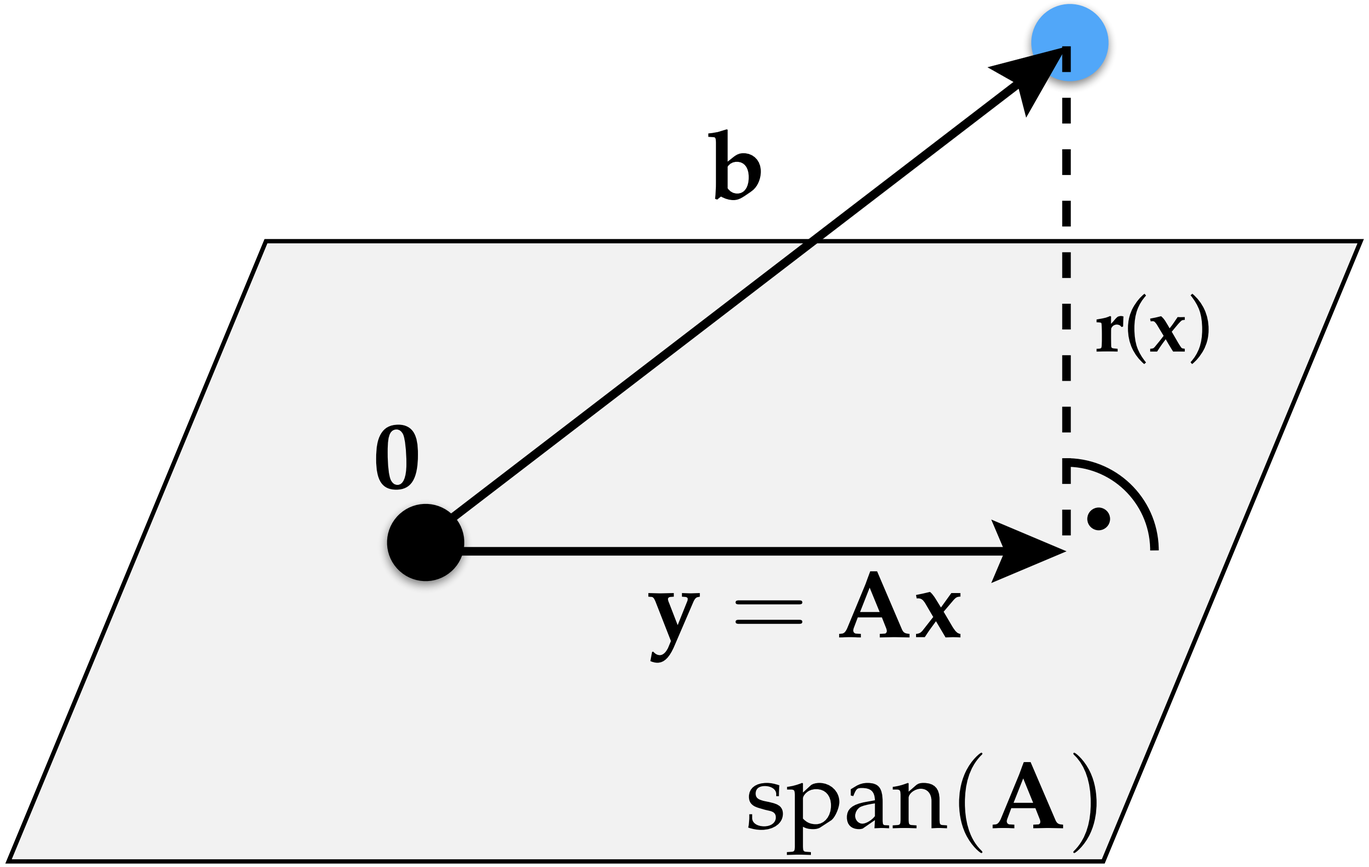
Why are they called normal equations?



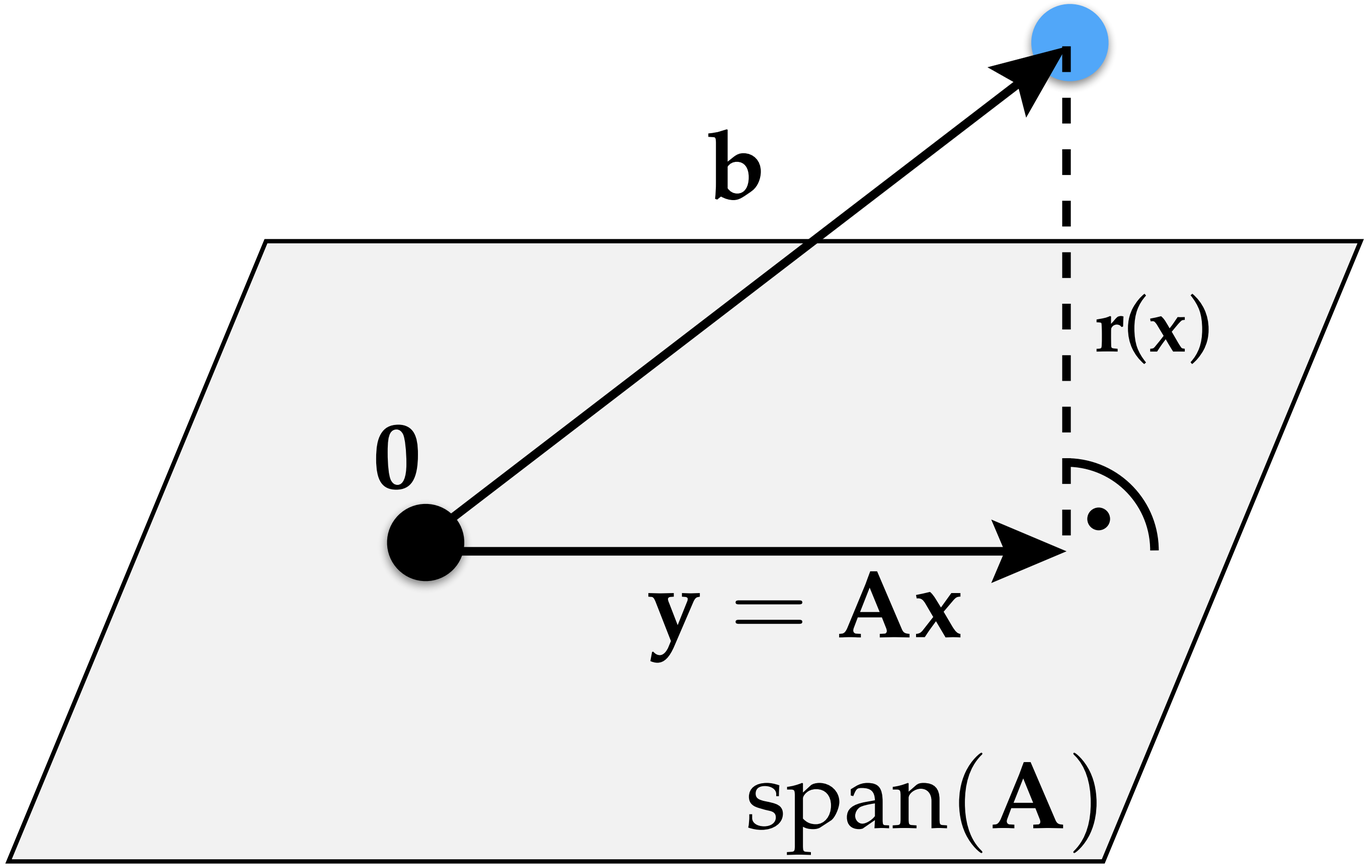
Why are they called normal equations?



Why are they called normal equations?

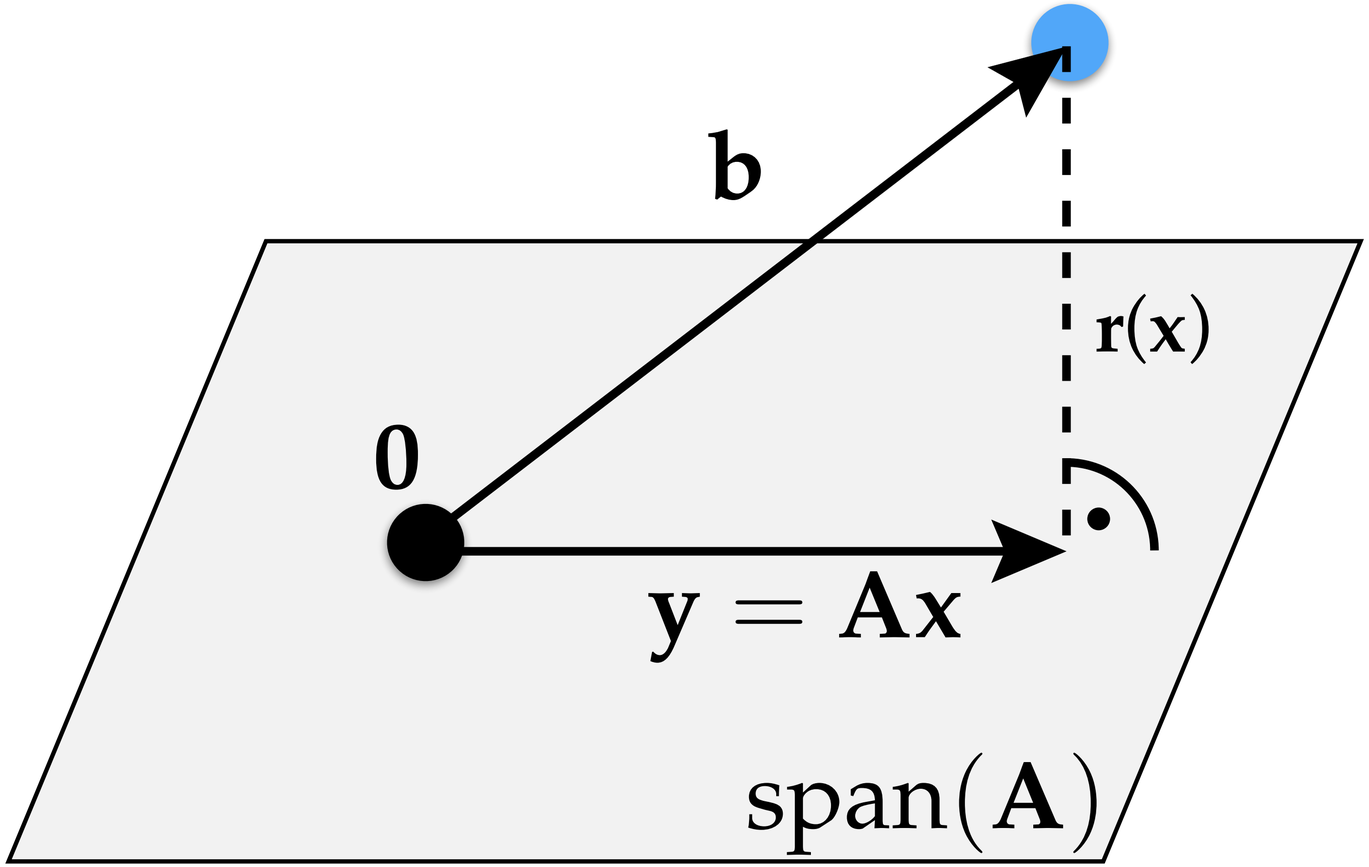


Why are they called normal equations?



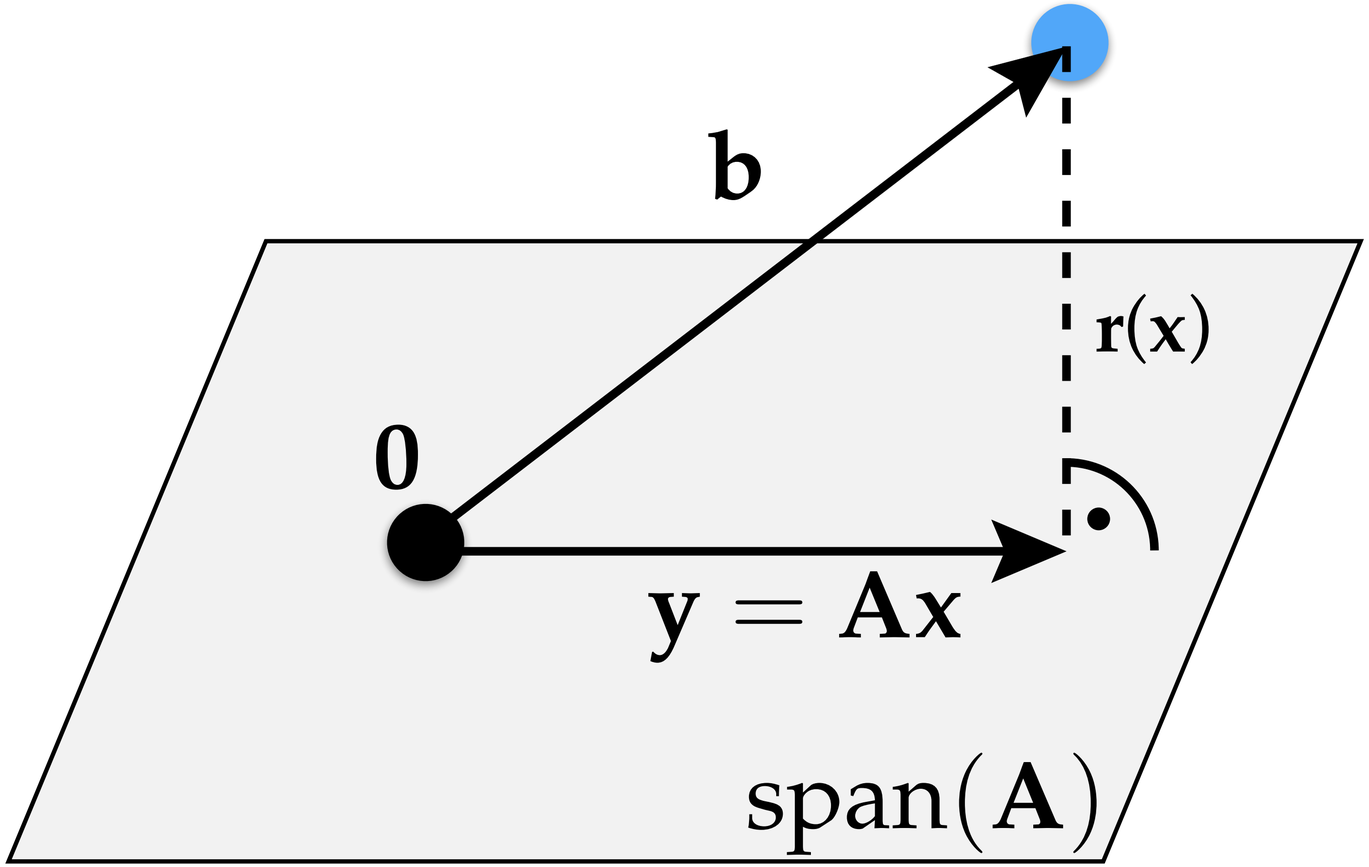
$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$$

Why are they called normal equations?



$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b} \;\Leftrightarrow\; \mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) =$$

Why are they called normal equations?



$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b} \;\Leftrightarrow\; \mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{0} \;\Leftrightarrow\; \mathbf{A}^T\mathbf{r}(\mathbf{x}) = \mathbf{0}.$$