Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Lecture 2:

Linear Algebra Fundamentals

Markus Hohle

University California, Berkeley

Numerical Methods for Computational Science

MSSE 273, 3 Units

Berkeley UNIVERSITY OF CALIFORNIA

Numerical Methods for Computational Science

Course Map

Outline

- Arrays in Python
- Vectors
- Matrices
- Lecture Exercise
- Solving Linear Systems

Outline

- Arrays in Python

- Vectors

- Matrices

- Lecture Exercise

- Solving Linear Systems

shapes:

```
A = np.arange(0,5,1)
```

```
In [36]: print(A)
[0 1 2 3 4]
```

```
In [37]: A.shape
Out[37]: (5,)
```

```
A = np.zeros((1,5))
```

```
In [53]: print(A)
[[0. 0. 0. 0. 0.]]
```

```
In [54]: A.shape
Out[54]: (1, 5)
```

Make sure that shapes make sense!

```
A = range(0,5,1)
```

```
In [40]: print(A)
range(0, 5)
```

```
In [41]: A.shape
Traceback (most recent call last):

  Cell In[41], line 1
    A.shape

AttributeError: 'range' object has no attribute 'shape'
```

```
In [60]: A = np.arange(0,5,1)

In [61]: Anew = A.reshape((1,5))

In [62]: Anew.shape
Out[62]: (1, 5)
```

Linear Algebra Fundamentals

shapes:

```
A = np.arange(0,5,1)
```

```
In [36]: print(A)
[0 1 2 3 4]
```

```
In [65]: print(type(A))
<class 'numpy.ndarray'>
```

```
A = np.zeros((1,5))
```

```
In [53]: print(A)
[[0. 0. 0. 0. 0.]]
```

```
In [68]: print(type(A))
<class 'numpy.ndarray'>
```

```
A = range(0,5,1)
```

```
In [71]: print(type(A))
<class 'range'>
```

Outline

- Arrays in Python
- **Vectors**
- Matrices
- Lecture Exercise
- Solving Linear Systems

recap:

quantities with **magnitude** and **direction**

→ vectors $\vec{\boldsymbol{v}}$

- velocity
- current
- acceleration
- force
- electric and magnetic field
- …

quantities with **magnitude**, **no direction**

→ scalars **s**

- density **(and all related quantities like concentration, pressure etc)**
- mass
- energy
- …

Why do we need vectors:

- “vectorized” code is : faster
  shorter
  maintainable → see lecture exercise
  readable
  scalable

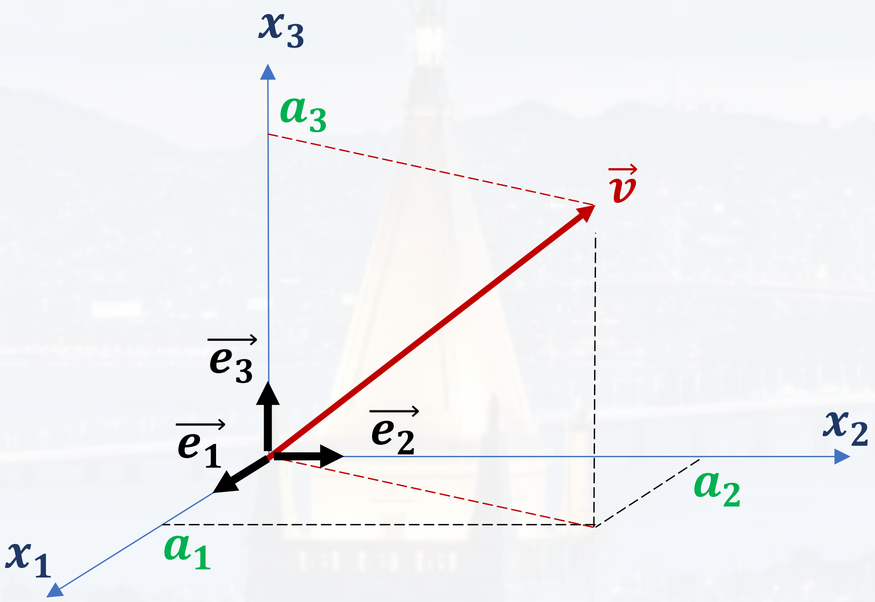
- data analysis and data acquisition of physical problems requires basic understanding

- many ML algorithms and tools are inspired by physics

- dimension reduction, PCA (eigenvectors, eigenvalues)

recap:

Cartesian coordinates



unit vectors $\vec{e_1}$, $\vec{e_2}$ and $\vec{e_3}$

$$\vec{e_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \vec{e_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \vec{e_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

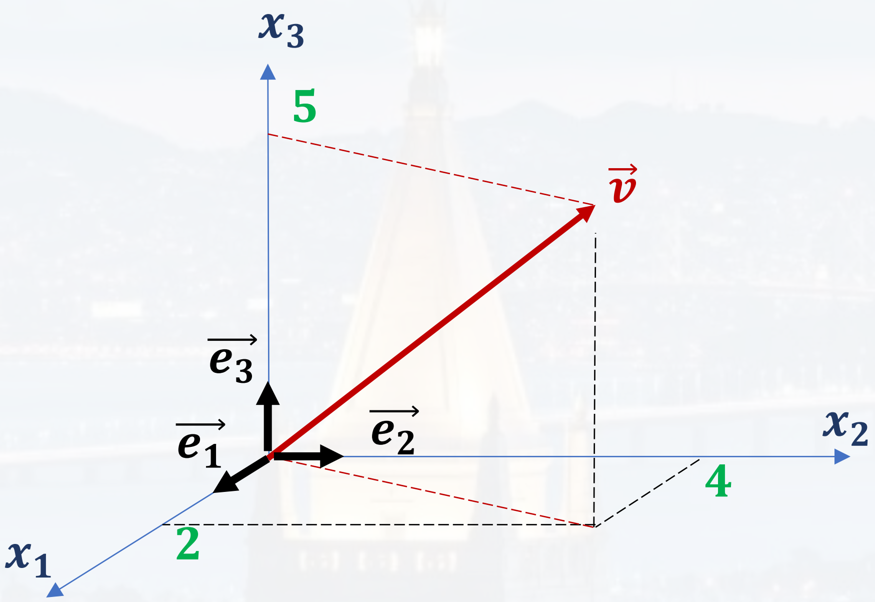
$\left.\begin{array}{l}\text{length 1 (normalized)} \\ \text{mutually orthogonal}\end{array}\right\}$ ortho normal

$$\vec{v} = a_1\vec{e_1} + a_2\vec{e_2} + a_3\vec{e_3}$$

$$\vec{v} = a_1\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + a_2\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + a_3\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

recap:

Cartesian coordinates



unit vectors $\vec{e_1}$, $\vec{e_2}$ and $\vec{e_3}$

$$\vec{e_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \vec{e_2} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \vec{e_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\left.\begin{array}{l} \text{length 1 (normalized)} \\ \text{mutually orthogonal} \end{array}\right\} \text{ ortho normal}$$

$$\vec{v} = a_1\vec{e_1} + a_2\vec{e_2} + a_3\vec{e_3}$$

$$\vec{v} = 2\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 4\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 5\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 5 \end{pmatrix}$$

recap:

FYI:

- non orthogonal coordinate systems (e. g. crystal lattices)
  → covariant and contravariant vectors

- curved ortho normal coordinates (polar coordinates, spherical coordinates etc.)

- general coordinates: curved, not normalized and not orthogonal **(not important for this course)**

recap:

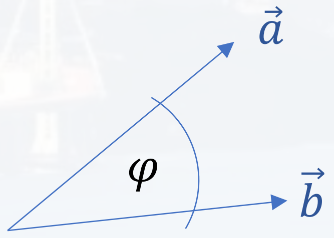
dot product of two vectors

$$\vec{a} \circ \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \circ \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$\vec{a} \circ \vec{b} = \sum_{i=1}^{I} a_i b_i$$

$$\cos(\varphi) = \frac{\vec{a} \circ \vec{b}}{|\vec{a}| \left|\vec{b}\right|}$$

angle between $\vec{a}$ and $\vec{b}$



$|\vec{a}|$ length of $\vec{a}$

recap:

dot product of two vectors

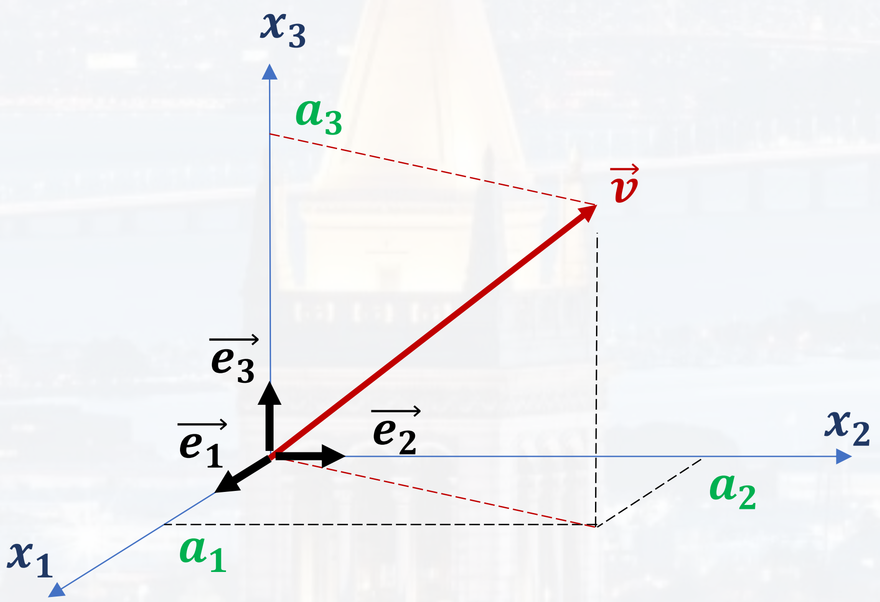
$$\vec{a} \circ \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \circ \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$\vec{a} \circ \vec{b} = \sum_{i=1}^{I} a_i b_i$$

"physical length" ***L*** of a vector

$$L = |\vec{a}| = \sqrt{a_1^2 + a_2^2 + a_3^2}$$

$$= \sqrt{\vec{a} \circ \vec{a}}$$



recap:

dot product of two vectors

$$\cos(\varphi) = \frac{\vec{a} \circ \vec{b}}{|\vec{a}|\,|\vec{b}|}$$

angle between $\vec{a}$ and $\vec{b}$

$$\vec{\boldsymbol{e_1}} = \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \qquad \vec{\boldsymbol{e_2}} = \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{pmatrix} \qquad \vec{\boldsymbol{e_3}} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{pmatrix}$$

$\vec{\boldsymbol{e_i}} \circ \vec{\boldsymbol{e_j}}$ = zero for i ≠ j

= 1 for i = j

$$\vec{\boldsymbol{e_i}} \circ \vec{\boldsymbol{e_j}} = \boldsymbol{\delta_{ij}}$$

Kronecker symbol $\delta_{ij} = \begin{cases} 0 \; for \; i \neq j \\ 1 \; for \; i = j \end{cases}$

Outline

- Arrays in Python
- Vectors
- **Matrices**
- Lecture Exercise
- Solving Linear Systems

Why do we need matrices:

- “vectorized” code is : faster
  shorter
  maintainable → see lecture exercise
  readable
  scalable

- AI/ML: essentially matrix operations

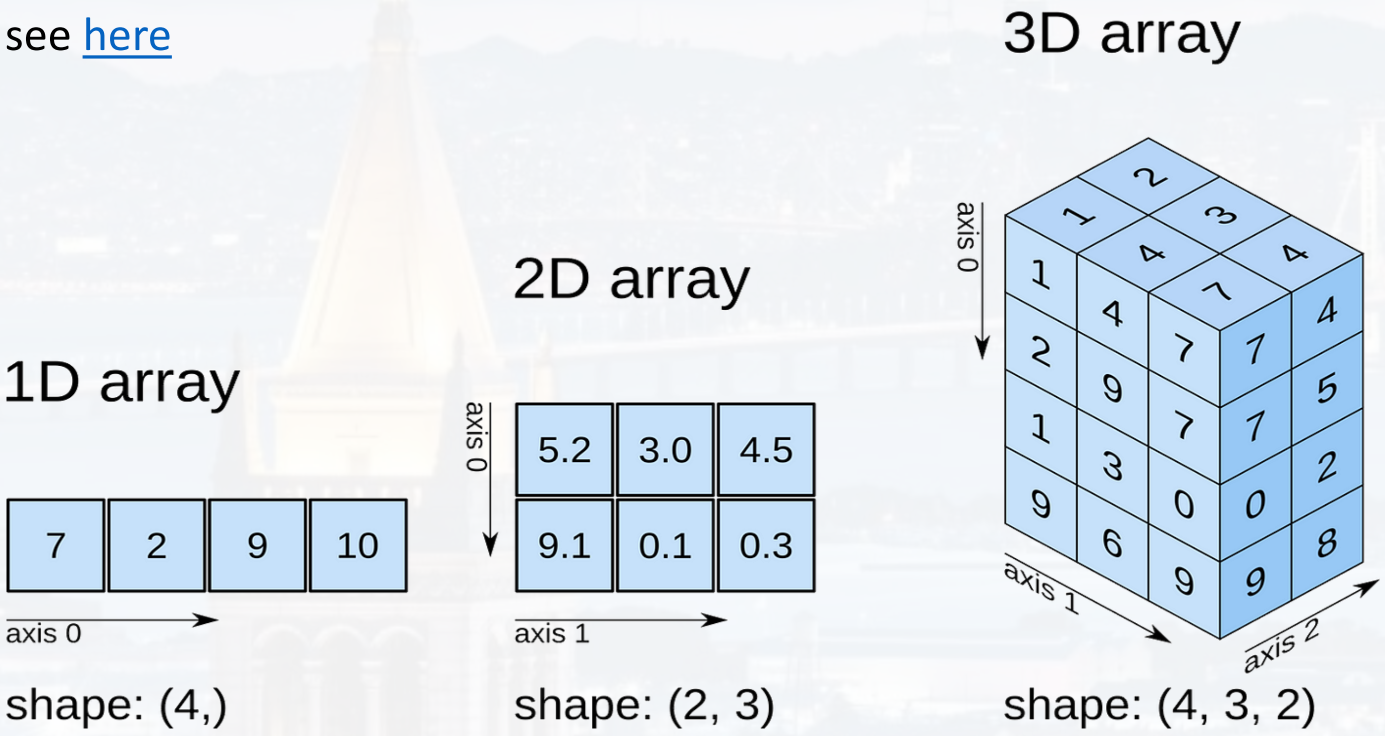
- regression, linear models etc

- easy to parallelize

in python:

```
[[ 3 -1  2  2]
 [15 -5 10 10]
 [ 0  0  0  0]
 [-9  3 -6 -6]]
```

see here



… and higher

note:

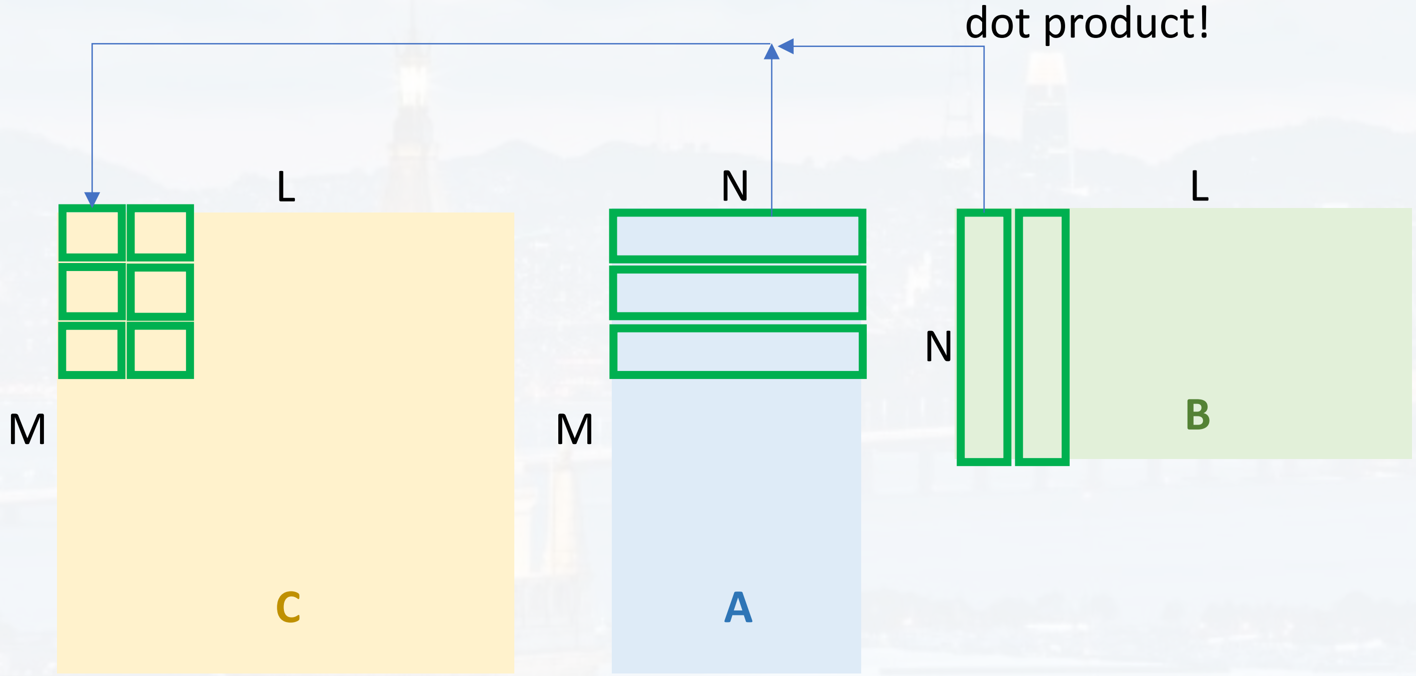
higher dimensional arrays are called **tensor** in the CS community, but they are not the same tensors as in math & physics

Linear Algebra Fundamentals

in math:

actual matrix multiplication

C = A*B



$$c_{m,l} = \sum_{n=0}^{N} a_{m,n} \; b_{n,l}$$

- only works if $n_{column}(A) = n_{row}(B)$
- result: $C(n_{row}(A), n_{column}(B))$

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])
```

```
1) np.dot(v1,v2)
2) np.outer(v1,v2)
```

$$\begin{pmatrix} a & b & c \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = a\alpha + b\beta + c\gamma$$

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \begin{pmatrix} a & b & c \end{pmatrix} = \begin{pmatrix} a\alpha & \alpha b & \alpha c \\ a\beta & \beta b & \beta c \\ a\gamma & \gamma b & \gamma c \end{pmatrix}$$

```
In [75]: print(np.dot(v1,v2))
-8

In [76]: print(np.dot(v2,v1))
-8
```

```
In [78]: print(np.outer(v1,v2))
[[ 3 -1  2  2]
 [15 -5 10 10]
 [ 0  0  0  0]
 [-9  3 -6 -6]]

In [79]: print(np.outer(v2,v1))
[[ 3 15  0 -9]
 [-1 -5  0  3]
 [ 2 10  0 -6]
 [ 2 10  0 -6]]
```

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])
```

```
In [81]: print(v1*v2)
[ 3 -5  0 -6]

In [82]: print(v2*v1)
[ 3 -5  0 -6]
```

default: element wise multiplication

```
In [84]: v1v2 = v1*v2

In [85]: v1v2.shape
Out[85]: (4,)

In [86]: v2v1 = v2*v1

In [87]: v2v1.shape
Out[87]: (4,)
```

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])

v1 = v1.reshape((len(v1),1))
v2 = v2.reshape((1,len(v2)))
```

column vector

row vector

v1 * v2[0]

```
In [94]: print(v1*v2)
[[ 3 -1  2  2]
 [15 -5 10 10]
 [ 0  0  0  0]
 [-9  3 -6 -6]]
```

v1 * v2[1]

```
In [95]: print(v2*v1)
[[ 3 -1  2  2]
 [15 -5 10 10]
 [ 0  0  0  0]
 [-9  3 -6 -6]]
```

note:
operations in Python are not exactly like those in math!

more info [here]

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])
```

dot product:

```
np.dot(v1,v1)**0.5
```

"physical length" ***L*** of a vector

$$\cos(\varphi) = \frac{\vec{v_1} \cdot \vec{v_2}}{|\vec{v_1}||\vec{v_2}|}$$

angle between v1 and v2

```
np.arccos(np.dot(v1,v2)/ np.sum(v1**2)**0.5 / np.sum(v2**2)**0.5)
```

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])
```

```
np.multiply(v1,v2)
```

```
In [119]: np.multiply(v1,v2)
Out[119]:
array([[ 3, -1,  2,  2],
       [15, -5, 10, 10],
       [ 0,  0,  0,  0],
       [-9,  3, -6, -6]])

In [120]: np.multiply(v2,v1)
Out[120]:
array([[ 3, -1,  2,  2],
       [15, -5, 10, 10],
       [ 0,  0,  0,  0],
       [-9,  3, -6, -6]])
```

v2 * v1[0]

v2 * v1[1]

```
v1 = np.array([1,5,0,-3])
v2 = np.array([3,-1,2,2])
```

```
v1 = v1.reshape((len(v1),1))
v2 = v2.reshape((1,len(v2)))
```

```
np.multiply(v1,v2)
```

```
In [119]: np.multiply(v1,v2)
Out[119]:
array([[ 3, -1,  2,  2],
       [15, -5, 10, 10],
       [ 0,  0,  0,  0],
       [-9,  3, -6, -6]])

In [120]: np.multiply(v2,v1)
Out[120]:
array([[ 3, -1,  2,  2],
       [15, -5, 10, 10],
       [ 0,  0,  0,  0],
       [-9,  3, -6, -6]])
```

```
In [122]: np.multiply(v1,v2)
Out[122]: array([ 3, -5,  0, -6])

In [123]: np.multiply(v2,v1)
Out[123]: array([ 3, -5,  0, -6])
```

Linear Algebra Fundamentals


```
M = np.array([[1,2,5,6], [0,8,5,-4], [-4,4,6,1],\
              [2,3,-1,0]])
```

```
array([[ 1,  2,  5,  6],
       [ 0,  8,  5, -4],
       [-4,  4,  6,  1],
       [ 2,  3, -1,  0]])
```

element wise multiplication:

```
M*M
M**2
```

```
np.outer(M,M)
```

1x	1	2	5	6	0	8	5	-4	-4	4	6	1	2	3	-1	0
2x	2	4	10	12	0	16	10	-8	-8	8	12	2	4	6	-2	0
5x	5	10	25	30	0	40	25	-20	-20	20	30	5	10	15	-5	0
	6	12	30	36	0	48	30	-24	-24	24	36	6	12	18	-6	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	16	40	48	0	64	40	-32	-32	32	48	8	16	24	-8	0
	5	10	25	30	0	40	25	-20	-20	20	30	5	10	15	-5	0
	-4	-8	-20	-24	0	-32	-20	16	16	-16	-24	-4	-8	-12	4	0
	-4	-8	-20	-24	0	-32	-20	16	16	-16	-24	-4	-8	-12	4	0
	4	8	20	24	0	32	20	-16	-16	16	24	4	8	12	-4	0
	6	12	30	36	0	48	30	-24	-24	24	36	6	12	18	-6	0
	1	2	5	6	0	8	5	-4	-4	4	6	1	2	3	-1	0
	2	4	10	12	0	16	10	-8	-8	8	12	2	4	6	-2	0
	3	6	15	18	0	24	15	-12	-12	12	18	3	6	9	-3	0
	-1	-2	-5	-6	0	-8	-5	4	4	-4	-6	-1	-2	-3	1	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
M = np.array([[1,2,5,6], [0,8,5,-4], [-4,4,6,1],\
              [2,3,-1,0]])
```

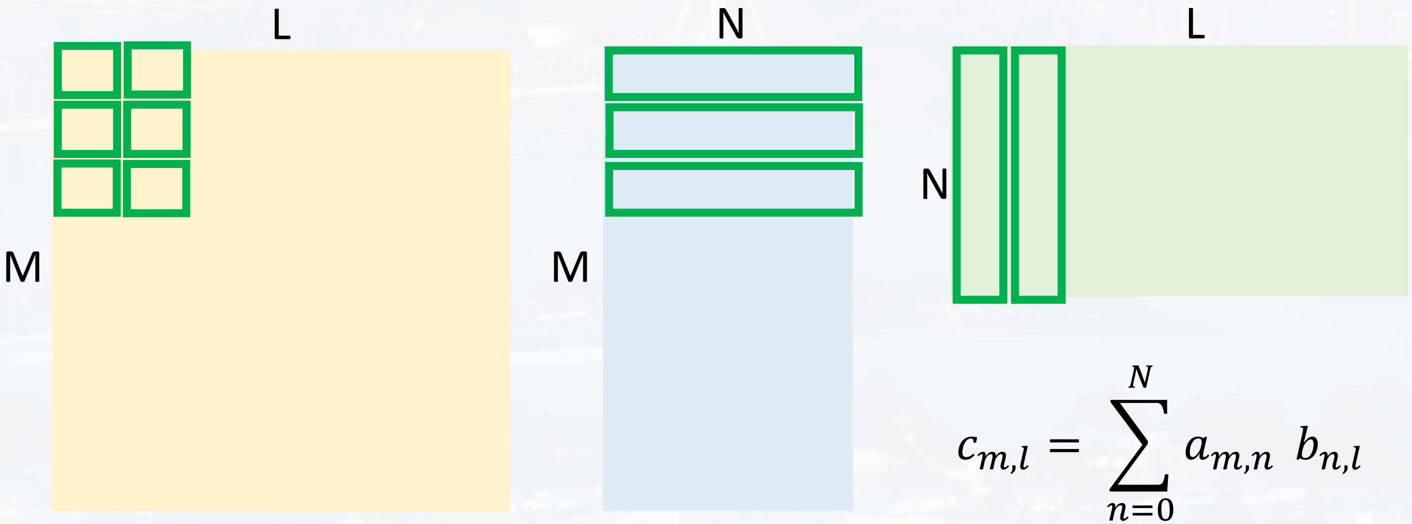
```
array([[ 1,  2,  5,  6],
       [ 0,  8,  5, -4],
       [-4,  4,  6,  1],
       [ 2,  3, -1,  0]])
```

```
np.dot(M,M)
```

```
[[ -7,  56,  39,   3],
 [-28,  72,  74, -27],
 [-26,  51,  35, -34],
 [  6,  24,  19,  -1]]
```

actual matrix multiplication

C = A*B



$$c_{m,l} = \sum_{n=0}^{N} a_{m,n}\ b_{n,l}$$

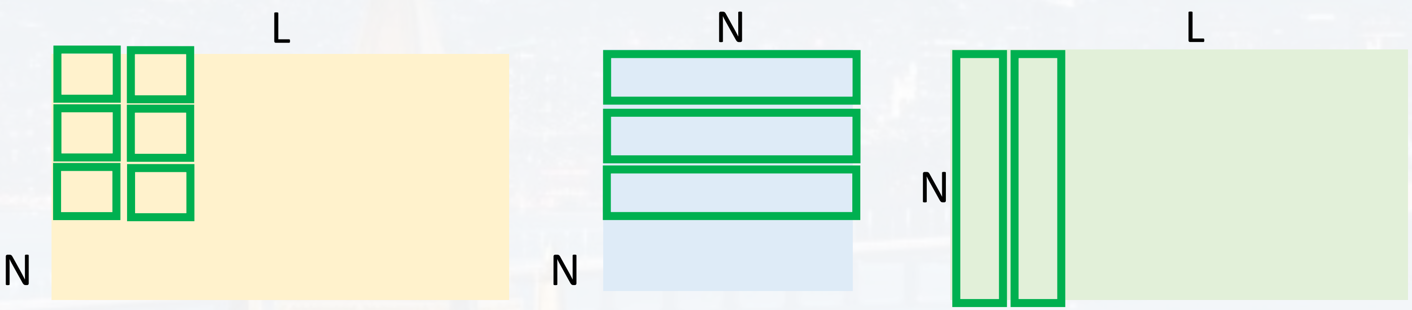
identity matrix

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathrm{I} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & 1 & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

identity: $\boldsymbol{I\,M} = \boldsymbol{M}$
inverse: $M^{-1}M = \mathrm{I}$
transpose: $[m_{ij}]^T = [m_{ji}]$
symmetry: $[m_{ij}] = [m_{ji}]$

actual matrix multiplication

B = I*B



$$c_{m,l} = \sum_{n=0}^{N} a_{m,n}\ b_{n,l}$$

```
np.eye(5)
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

$$MM^{-1} = I$$

```
M_1 = np.linalg.inv(M)

print(np.dot(M,M_1))
```

identity:	$I\,M = M$
inverse:	$\boldsymbol{M^{-1}M} = \mathbf{I}$
transpose:	$[m_{ij}]^T = [m_{ji}]$
symmetry:	$[m_{ij}] = [m_{ji}]$

```
[[ 1.00000000e+00  5.55111512e-17  5.55111512e-17 -1.11022302e-16]
 [ 0.00000000e+00  1.00000000e+00  5.55111512e-17 -1.11022302e-16]
 [-4.16333634e-17  2.77555756e-17  1.00000000e+00  5.55111512e-17]
 [ 5.55111512e-17 -5.55111512e-17  8.32667268e-17  1.00000000e+00]]
```

$$\mathrm{I} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & 1 & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

½ **machine epsilon**
¼ **machine epsilon**

numerical accuracy of the operation **(machine epsilon)**

identity:	$I\ M = M$
inverse:	$M^{-1}M = \mathrm{I}$
transpose:	$[\boldsymbol{m_{ij}}]^{\boldsymbol{T}} = [\boldsymbol{m_{ji}}]$
symmetry:	$[m_{ij}] = [m_{ji}]$

```
M = np.array([[1,2,3,4], [5,6,7,8],\
              [9,10,11,12], [13,14,15,16]])
```

```
In [9]: M
Out[9]:
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

```
In [10]: M.transpose()
Out[10]:
array([[ 1,  5,  9, 13],
       [ 2,  6, 10, 14],
       [ 3,  7, 11, 15],
       [ 4,  8, 12, 16]])
```

Linear Algebra Fundamentals

identity:	$I\ M = M$
inverse:	$M^{-1}M = \mathrm{I}$
transpose:	$[m_{ij}]^T = [m_{ji}]$
symmetry:	$[\boldsymbol{m_{ij}}] = [\boldsymbol{m_{ji}}]$

usually (but not always!) **distance** matrices

- see lecture exercise
- some ML algorithms (trees)

	0	1	2	3	4	5	6	7
0	0	0.118625	0.235409	0.000235242	0.0545817	0.0351447	0.0947077	0.148681
1	0.118625	0	0.0198161	0.129425	0.0122749	0.282906	0.42532	0.00169516
2	0.235409	0.0198161	0	0.250527	0.0632834	0.452469	0.628746	0.00991963
3	0.000235242	0.129425	0.250527	0	0.0619835	0.0296293	0.0855028	0.160744
4	0.0545817	0.0122749	0.0632834	0.0619835	0	0.177322	0.293085	0.0230932
5	0.0351447	0.282906	0.452469	0.0296293	0.177322	0	0.0144665	0.328399
6	0.0947077	0.42532	0.628746	0.0855028	0.293085	0.0144665	0	0.480718
7	0.148681	0.00169516	0.00991963	0.160744	0.0230932	0.328399	0.480718	0

<u>Outline</u>

- Arrays in Python
- Vectors
- Matrices
- **Lecture Exercise**
- Solving Linear Systems

Let's combine some of those commands we just have learned

```
np.zeros()
np.arange()
np.tile()
np.transpose()
```

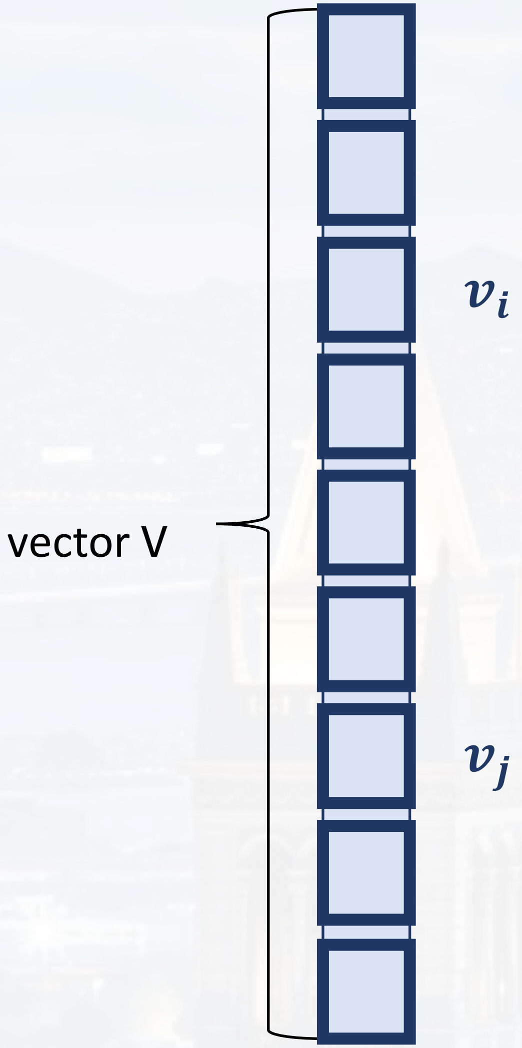
for a particular example about **avoiding loops**!

Linear Algebra Fundamentals

Lecture Exercise

for loops are simple, but slow → use lin algebra or numpy commands

calculating distances d



calculating the distance between each element

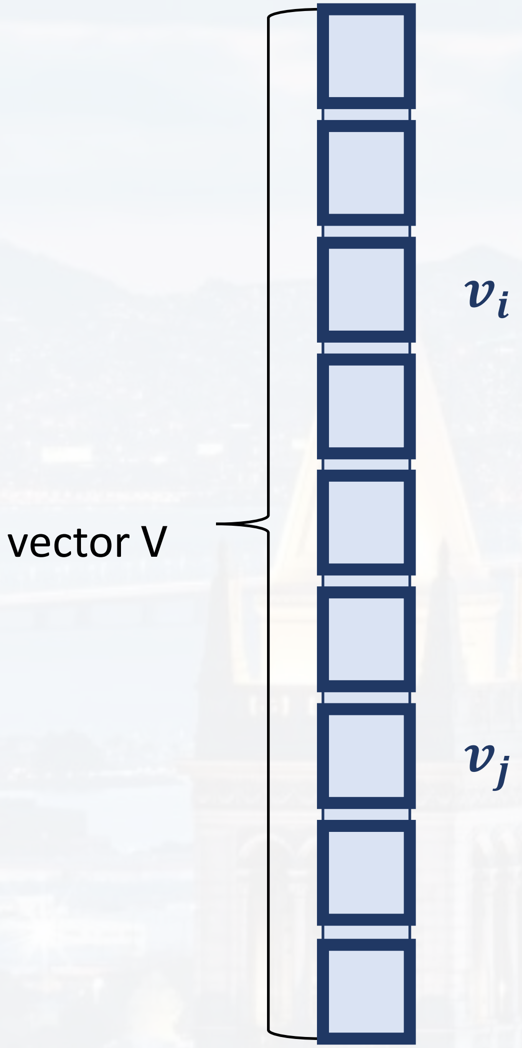
$$d(v_i, v_j) = (v_i - v_j)^2$$

Linear Algebra Fundamentals

Lecture Exercise

`for` loops are simple, but slow → use lin algebra or numpy commands

calculating distances d



calculating the distance between each element

$$d(v_i, v_j) = (v_i - v_j)^2$$

efficiency:

vector of length N → N*N operations

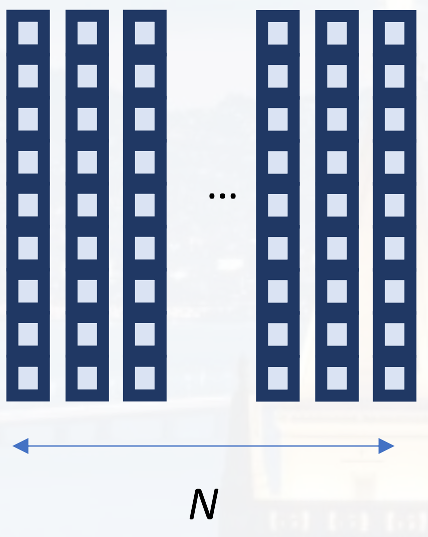
we know that the diagonal $d(v_i, v_i) = 0$ → N fewer operations

only half the operations are necessary: $d(v_i, v_j) = d(v_j, v_i)$

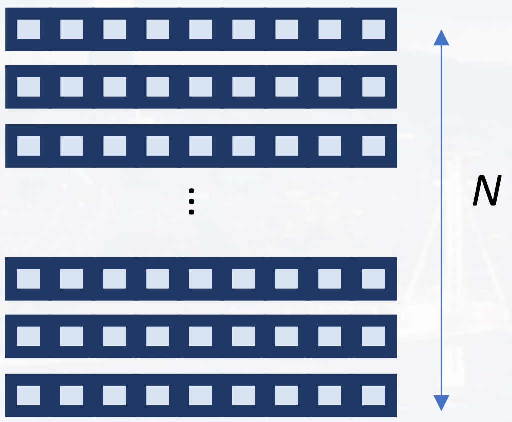
→ instead of N*N operations: only (N-1)*(N-1)/2 needed `for` loops are simple, but slow → use lin algebra or numpy commands

calculating distances d

```
Rep = np.tile(V, (N,1))
```



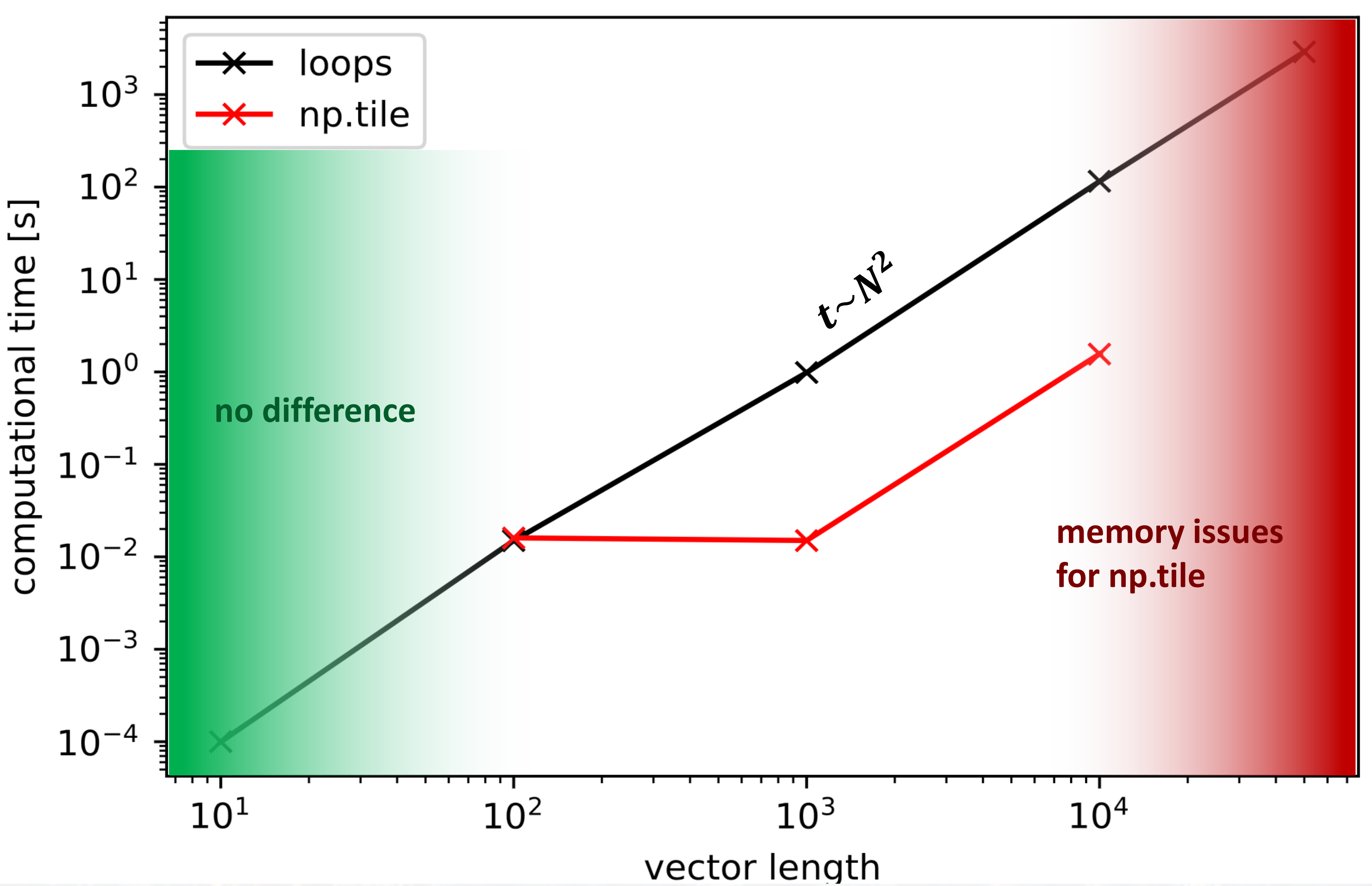

```
D   = Rep.transpose()
```




```
D   = (Rep - Rep.transpose())**2
```

	`loop`	`efficient loop`	`np.tile`
N = 500	**0.11 sec**	**0.05 sec**	**0.0003 sec**
N = 10,000	**35.0 sec**	**17.8 sec**	**1.25 sec**
N = 50,000			**180 sec**

see `02_Lecture_Exercise.ipynb`



<u>Outline</u>

- Arrays in Python
- Vectors
- Matrices
- Lecture Exercise
- **Solving Linear Systems**

finding the intersection of **two lines**:

$$y_1 = a_1 x_1 + c_1$$

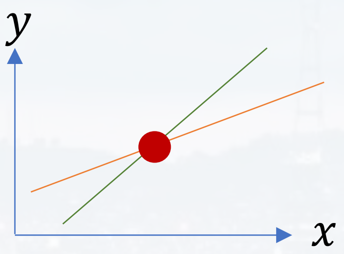
$$y_2 = a_2 x_2 + c_2$$

$$x_1 = x_2$$
$$y_1 = y_2$$

$$a_2 x + c_2 = a_1 x + c_1$$

$$x = \frac{c_2 - c_1}{a_1 - a_2}$$

$$y = a_1 \frac{c_2 - c_1}{a_1 - a_2} + c_1$$



finding the intersection of **three planes**:

$$y_1 = a_{11} x_{11} + a_{12} x_{12} + c_1$$

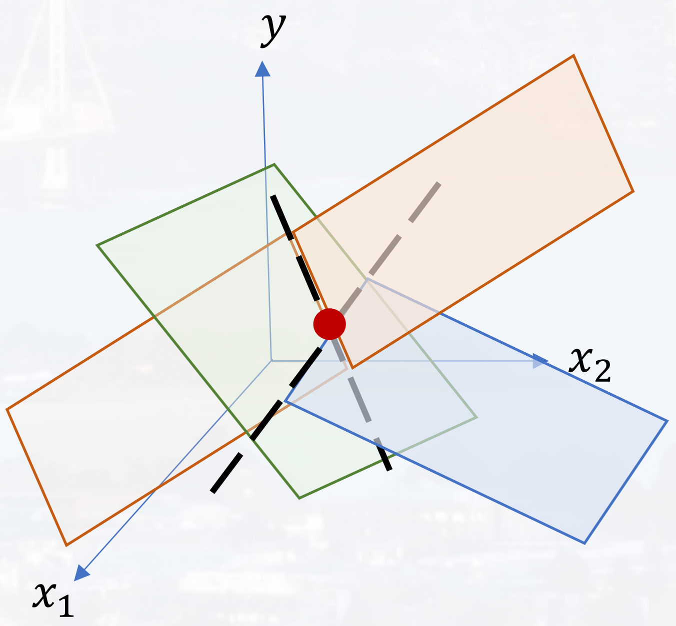
$$y_2 = a_{21} x_{21} + a_{22} x_{22} + c_2$$

$$y_2 = a_{31} x_{31} + a_{32} x_{32} + c_2$$

$$x_{11} = x_{21} = x_{31} = x_1$$

$$x_{12} = x_{22} = x_{32} = x_2$$

$$y_1 = y_2 = y_3 = y$$



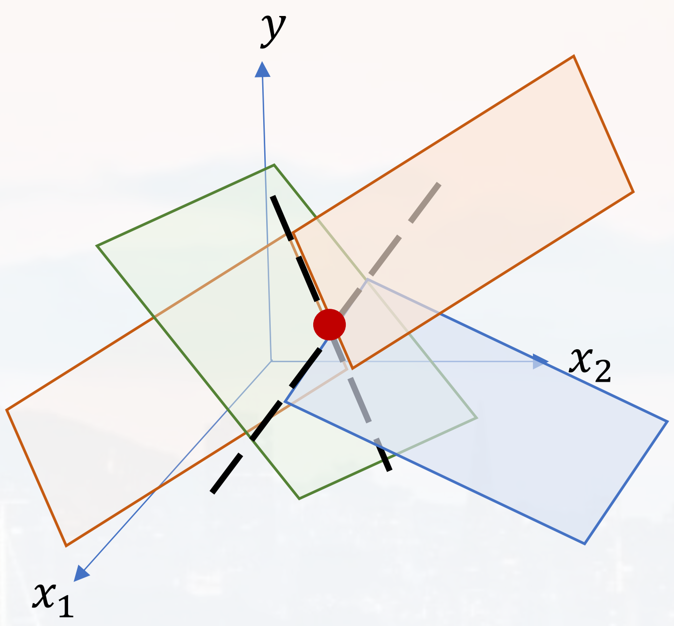
<u>more general:</u>

$$x_{11} = x_{21} = x_{31} = x_1$$

$$x_{12} = x_{22} = x_{32} = x_2$$

$$y_1 = y_2 = y_3 = y$$

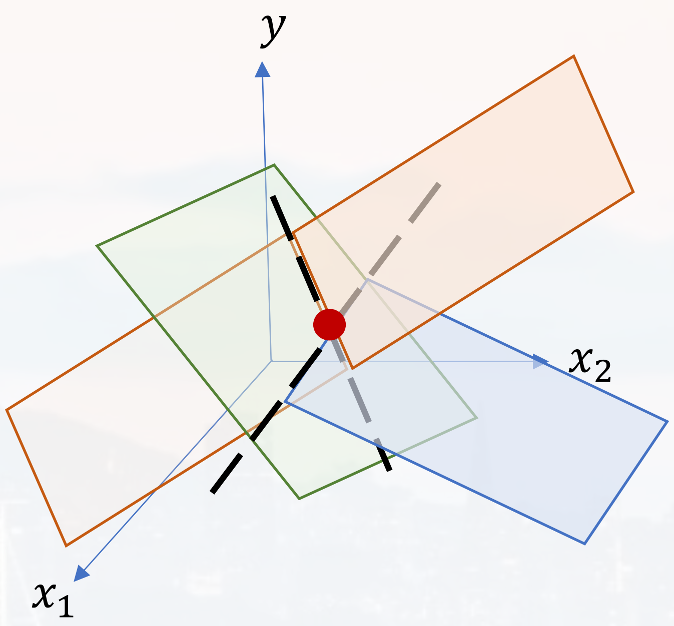
$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ \dots + a_{1n}x_n = c_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ \dots + a_{2n}x_n = c_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ \dots + a_{3n}x_n = c_3$$
$$\dots$$
$$a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ \dots + a_{mn}x_n = c_m$$



$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}$$

more general:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

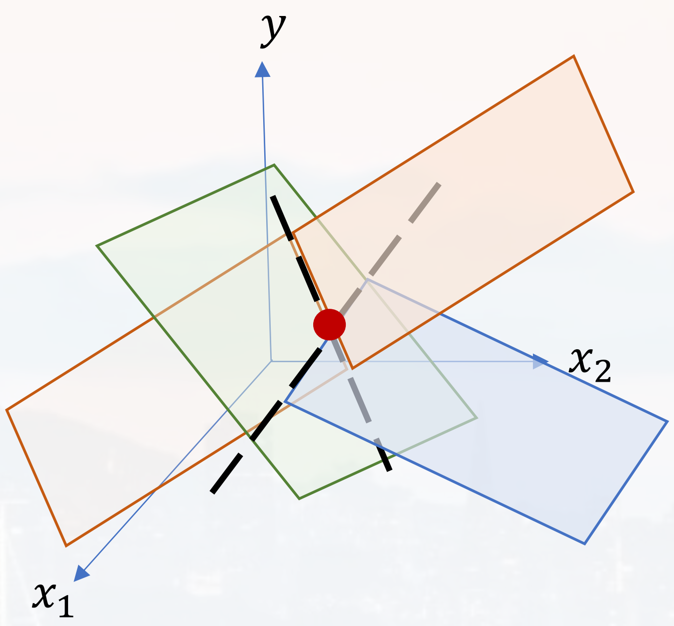


$$\boxed{A\vec{x} = \vec{c}}$$

more general:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

$$\boxed{A\vec{x} = \vec{c}}$$



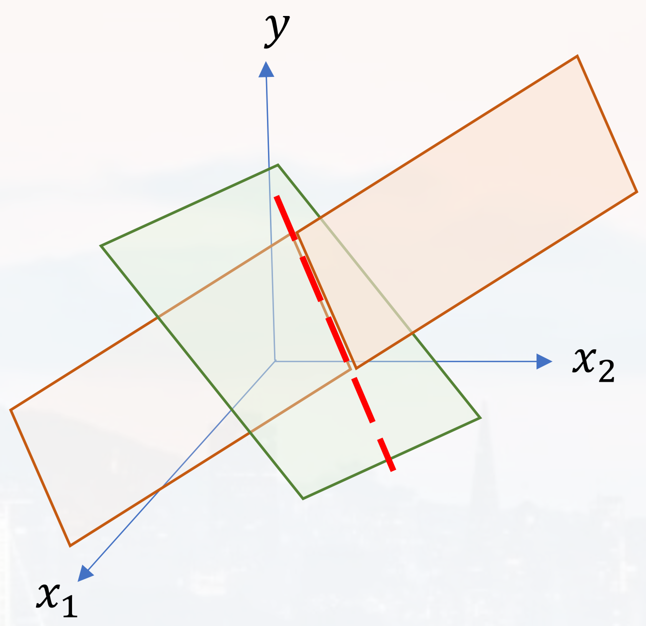
general set of solutions

for n = m → solution is unique: a point

<u>more general:</u>

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

$$\boxed{A\vec{x} = \vec{c}}$$



<u>**general** set of solutions</u>

for n = m → solution is unique: a point

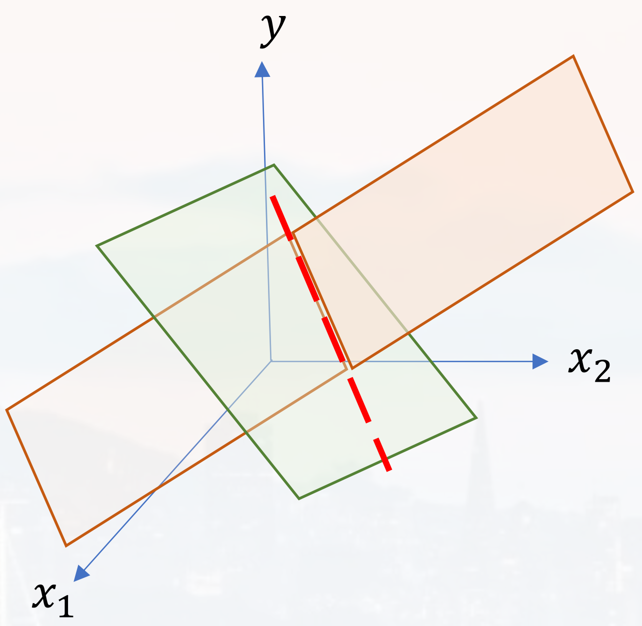
for n > m (more variables than equations)
→ solution is not unique: line, hyperplane

for n < m (more equations than variables)
→ no solution

more general:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

$$\boxed{A\vec{x} = \vec{c}}$$



general set of solutions

for n = m → solution is unique: a point

for n > m (more variables than equations)
→ solution is not unique: line, hyperplane

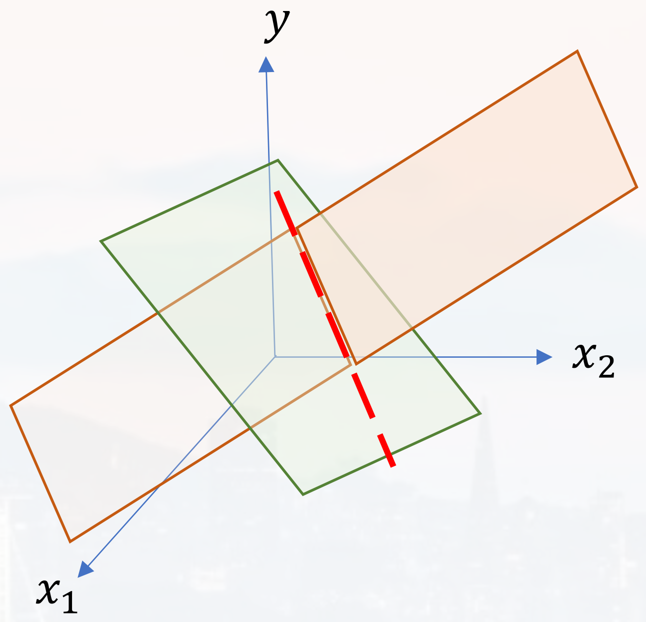
for n < m (more equations than variables)
→ no solution

exceptions!

more general:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

$$\boxed{A\vec{x} = \vec{c}}$$



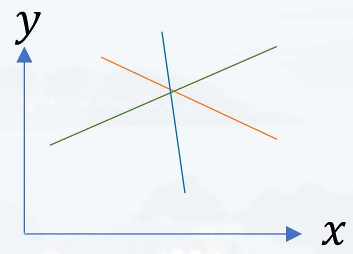
general set of solutions

for n = m → solution is unique: a point

for n > m (more variables than equations)
→ solution is not unique: line, hyperplane

for n < m (more equations than variables)
→ no solution

exceptions!



more general:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}}_{\vec{c}}$$

$$\boxed{A\vec{x} = \vec{c}}$$

$\vec{x} = ?$

for n = m

$$A^{-1}A\vec{x} = A^{-1}\vec{c}$$

$$\vec{x} = A^{-1}\vec{c}$$

solving for $\vec{x}$:

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix}$$

one possibility: **Gaussian elimination**

idea: it is a system of linear equations, hence we can

- swap rows
- subtract rows from each other
- multiply rows with any non-zero number

goal: turning this into the **identity matrix *I*** by performing the above operations

→ $\vec{c}$ must be the solution $\vec{x}$

$$\underbrace{\begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{\vec{x}} = \underbrace{\begin{pmatrix} 8 \\ 11 \\ -3 \end{pmatrix}}_{\vec{c}} \longrightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

solving for $\vec{x}$:

$$\begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 11 \\ -3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\begin{pmatrix} 2x_1 & x_2 & -x_3 \\ \boxed{-3x_1 \quad -x_2 \quad 2x_3} \\ -2x_1 & x_2 & 2x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 11 \\ -3 \end{pmatrix}$$

adding 3/2 of row 1 to row 2

$$\begin{pmatrix} 2x_1 & x_2 & -x_3 \\ 0 & 0.5x_2 & 0.5x_3 \\ \boxed{-2x_1 \quad x_2 \quad 2x_3} \end{pmatrix} = \begin{pmatrix} 8 \\ 1 \\ -3 \end{pmatrix}$$

adding row 1 to row 3

$$\begin{pmatrix} 2x_1 & x_2 & -x_3 \\ 0 & 0.5x_2 & 0.5x_3 \\ \boxed{0 \quad 2x_2 \quad x_3} \end{pmatrix} = \begin{pmatrix} 8 \\ 1 \\ 5 \end{pmatrix}$$

subtracting 4x row 2 from row 3

$$\begin{pmatrix} 2x_1 & x_2 & -x_3 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 1 \\ 1 \end{pmatrix}$$

lower left looks already like from an identity matrix
→ doing the same for the upper right now

solving for $\vec{x}$:

$$\begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 11 \\ -3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\begin{pmatrix} 2x_1 & x_2 & -x_3 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 1 \\ 1 \end{pmatrix}$$

subtracting row 3 from row 1

$$\begin{pmatrix} 2x_1 & x_2 & 0 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 1 \\ 1 \end{pmatrix}$$

subtracting 2 x row 2 from row 1

$$\begin{pmatrix} 2x_1 & 0 & -x_3 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 1 \end{pmatrix}$$

subtracting row 3 from row 1

$$\begin{pmatrix} 2x_1 & 0 & 0 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}$$

solving for $\vec{x}$:

$$\begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 11 \\ -3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$\begin{pmatrix} 2x_1 & 0 & 0 \\ 0 & 0.5x_2 & 0.5x_3 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}$$

adding 0.5 row 3 to row 2

$$\begin{pmatrix} 2x_1 & 0 & 0 \\ 0 & 0.5x_2 & 0 \\ 0 & 0 & -x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 1.5 \\ 1 \end{pmatrix}$$

dividing row 1 by 2
multiplying row 2 by 2
multiplying row 3 by -1

$$\boxed{\begin{pmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix}}$$

With the same trick, we can calculate the inverse of A, A^{-1}

$$AA^{-1} = I$$

therefore

$$A\,I = \begin{pmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

multiplying with A^{-1}

goal: turning this into the structure below, by performing the operations from a few slides ago

→ will return A^{-1}

$$AA^{-1}IA^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \bar{a}_{11} & \bar{a}_{12} & \bar{a}_{13} \\ \bar{a}_{21} & \bar{a}_{22} & \bar{a}_{23} \\ \bar{a}_{31} & \bar{a}_{23} & \bar{a}_{33} \end{pmatrix}$$

$\boxed{A\vec{x} = \vec{c}}$

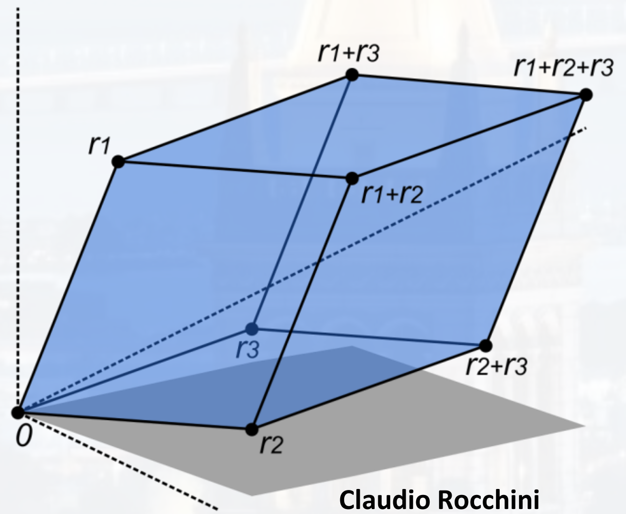
solving for x:

→ need to calculate A^{-1}
→ need to calculate a quantity called

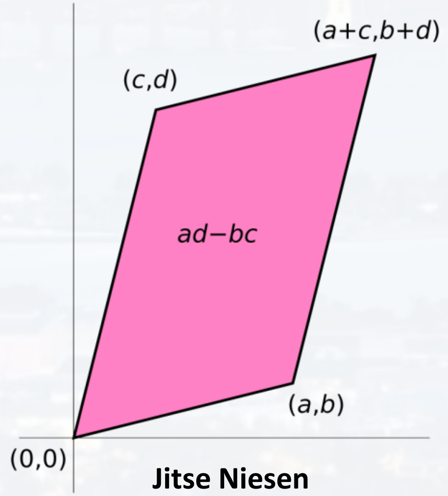
determinant of A, *det(A)*

$$A^{-1} \sim \frac{1}{\det(A)}$$

- if *det(A) = 0* → no solution
- *|det(A)|*: volume spanned by the vectors in A



Claudio Rocchini



Jitse Niesen

$$\boxed{A\vec{x} = \vec{c}}$$

<u>solving for x:</u>

→ need to calculate A^{-1}
→ need to calculate a quantity called

determinant of A, *det(A)*

$$A^{-1} \sim \frac{1}{\det(A)}$$

$$\begin{array}{ccccc} + & + & + & & \\ a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \\ - & - & - & & \end{array}$$

Kmhkmh

$$\det(A) = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32}$$
$$-a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12}$$

$$A\vec{x} = \vec{c}$$

solving for x:

→ need to calculate A^{-1}
→ need to calculate a quantity called

determinant of A, *det(A)*

$$A^{-1} \sim \frac{1}{\det(A)}$$

N x N matrix:

$$\det(A) = \sum_{i_1, i_2, \ldots, i_n} \varepsilon_{i_1 \ldots i_n} a_{1,i_1} \ldots a_{n\, i_n}$$

where

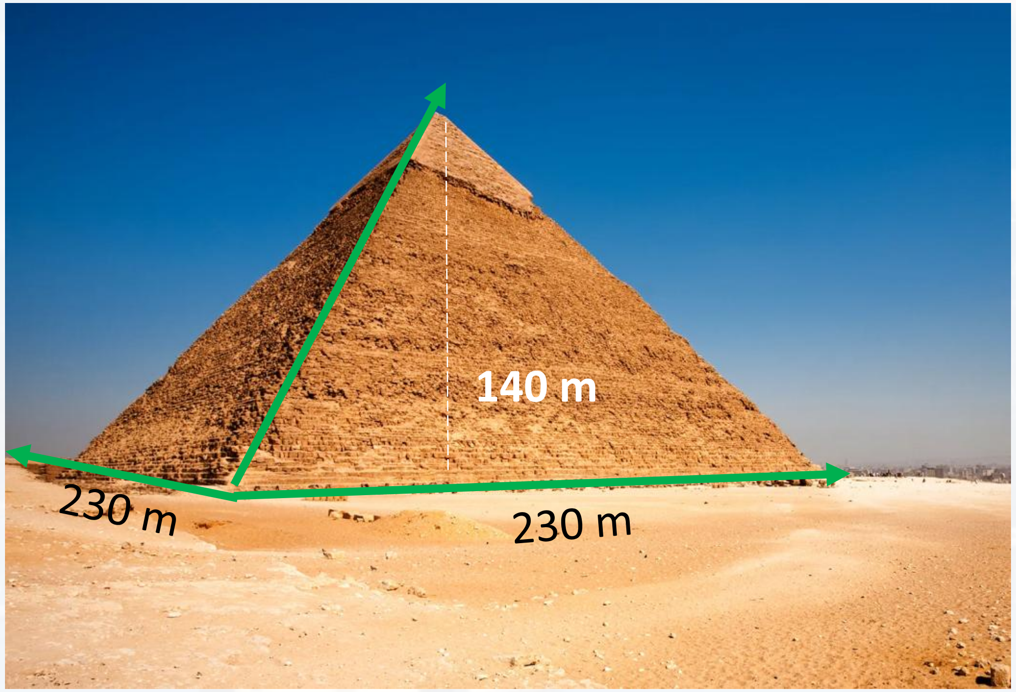
$$\varepsilon_{i_1 \ldots i_n} = \boxed{\prod_{1 \le \mu < \vartheta \le n} \operatorname{sgn}(i_\vartheta - i_\mu)}$$

(Levi-Civita symbol)

changing indices does not change |det(A)|

determinant of A, *det(A)*

$$A\vec{x} = \vec{c}$$



$$\varepsilon_{i_1 \ldots i_n} = \prod_{1 \le \mu < \vartheta \le n} \operatorname{sgn}(i_\vartheta - i_\mu)$$

$$V = \left| \det \begin{pmatrix} 230 & 0 & 115 \\ 0 & 230 & 115 \\ 0 & 0 & 140 \end{pmatrix} \right| \frac{1}{3} = \frac{230 * 230 * 140 + 0 + 0 - 0 - 0 - 0}{3} = 2{,}468{,}666\ m^3$$

determinant of A, *det(A)*

$$\varepsilon_{i_1 \ldots i_n} = \prod_{1 \le \mu < \vartheta \le n} \operatorname{sgn}(i_\vartheta - i_\mu)$$

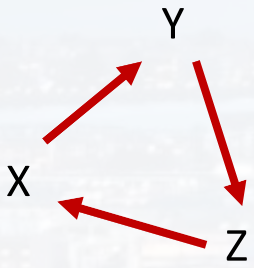
$$V = \left| \det \begin{pmatrix} 230 & 0 & 115 \\ 0 & 230 & 115 \\ 0 & 0 & 140 \end{pmatrix} \right| \frac{1}{3} = \frac{230 * 230 * 140 + 0 + 0 - \ 0 \ - 0 - 0}{3} = 2{,}468{,}666\ m^3$$

volume does not depend on **where** I put my **coord origin**…

$$V = \left| \det \begin{pmatrix} 0 & 230 & 115 \\ 230 & 0 & 115 \\ 0 & 0 & 140 \end{pmatrix} \right| \frac{1}{3} = \left| \frac{0 + 0 + 0 - \ 140 * 230 * 230 \ - 0 - 0}{3} \right| = 2{,}468{,}666\ m^3$$

…or how I **turn** the object!

$$V = \left| \det \begin{pmatrix} 115 & 230 & 0 \\ 115 & 0 & 230 \\ 140 & 0 & 0 \end{pmatrix} \right| \frac{1}{3} = \left| \frac{0 + 230 * 230 * 140 + 0 - \ 0 - 0 - 0}{3} \right| = 2{,}468{,}666\ m^3$$



$$V = \left| \det \begin{pmatrix} 140 & 0 & 0 \\ 115 & 230 & 0 \\ 115 & 0 & 230 \end{pmatrix} \right| \frac{1}{3} = \left| \frac{140 * 230 * 230 + 0 + 0 - \ 0 - 0 - 0}{3} \right| = 2{,}468{,}666\ m^3$$

$$\boxed{A\vec{x} = \vec{c}}$$

<u>note:</u> - more columns than rows in A

→ too many variables, not solvable
→ A^{-1} doesn't exist

- fewer columns than rows in A

→ too many equations, solution not unique
→ A^{-1} doesn't exist

- A^{-1} only **exists in principle if *A* is a square matrix**!

- even then the system of equations might be **singular** or **degenerate**
→ A^{-1} doesn't exist, *det(A) = 0*

Thank you for your attention!