SBT 645 Introduction to Scientific Computing in Sports Science

#8

SERDAR ARITAN

serdar.aritan@hacettepe.edu.tr

Biyomekanik Araştırma Grubu
www.biomech.hacettepe.edu.tr
Spor Bilimleri Fakültesi
www.sbt.hacettepe.edu.tr
Hacettepe Universitesi, Ankara, Türkiye
www.hacettepe.edu.tr

De Motu Animalium G.Borelli (1680)

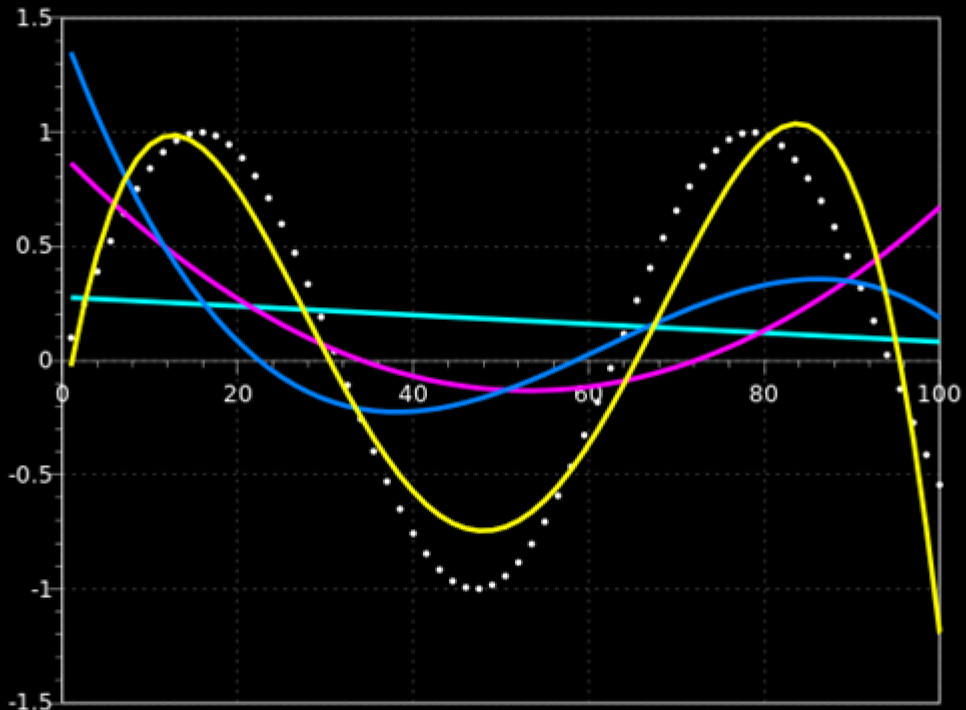
Polynomials

Most commonly, one fits a function of the form *y=f(x)*.

First degree polynomial equation $y = ax + b$

Second degree polynomial equation $y = ax^2 + bx + c$

Third degree polynomial equation $y = ax^3 + bx^2 + cx + d$



Polynomial curves fitting points generated with a sine function.

Cyan line is a first degree polynomial, purple line is second degree, blue line is third degree and yellow is fourth degree.

Polynomials

NumPy

An n[th] order polynomial in variable x is written as

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

It is natural to associate a row vector A_p with p, namely

$$A_p = \begin{bmatrix} a_1 & a_2 & \cdots & a_n & a_{n+1} \end{bmatrix}$$

A few examples: Row vector representations of

$$p(x) = 6x^3 + 5x^2 - 3x + 7 \qquad q(x) = 9x^2 - 2x + 4$$

are

```
>>> p = P.Polynomial([6, 5, -3, 7])
>>> q = P.Polynomial([9, -2, 4])
```

"leading zeros"

But [0 0 0 9 -2 4] also represents *q*…

Polynomials

- Polynomial addition
- Polynomial subtraction
- Polynomial multiplication
- Polynomial evaluation (use **polyval**)
- Plotting the graph of a polynomial
- Roots of a polynomial (use **roots**)

The inputs to the functions will be row vectors, representing polynomials.

Polynomials

Adding two polynomials requires adding their coefficients. The sum of

$$a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}$$

$$b_1x^n + b_2x^{n-1} + \cdots + b_nx + b_{n+1}$$

is simply

$$(a_1 + b_1)x^n + (a_2 + b_2)x^{n-1} + \cdots + (a_n + b_n)x + (a_{n+1} + b_{n+1})$$

In terms of the row-vector representation of the polynomials, we simply add them, element-by-element.

But the row vectors may be different lengths, and we need to "align" them.

Polynomials

NumPy

The row vector representations of

$$a(x) = 6x^3 + 5x^2 - 3x + 7$$

$$b(x) = 9x^2 + 2x - 4$$

Are

```
import numpy as np

P = np.polynomial
a = P.Polynomial([6, 5, -3, 7])
b = P.Polynomial([0, 9, 2, -4])

print(np.polyadd(a, b)
[Polynomial([ 6., 14., -1., 3.], domain=[-1.,  1.], window=[-1.,  1.])]
```

Polynomials

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$

$b(x) = 3x^2 + 2x - 4$

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$

$+ (6x^3 + 5x^2 - 3x + 7)*2x$

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [  0  0  0  0  0  0 ]
```

Polynomials

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$

$+ (6x^3 + 5x^2 - 3x + 7)*2x$

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [   0    0    0    0    0    0  ]
```

Polynomials

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add **A*B(1)** to **C(1:4)**

$+ (6x^3 + 5x^2 - 3x + 7)*2x$

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [   0    0    0    0    0    0  ]
```

Polynomials

NumPy

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add A*B(1) to C(1:4)

$+ (6x^3 + 5x^2 - 3x + 7)*2x$

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [   18   15   -9   21    0    0 ]
```

Polynomials

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add A*B(1) to C(1:4)

$+ (6x^3 + 5x^2 - 3x + 7)*2x$ **add A*B(2) to C(2:5)**

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [  18  15  -9  21   0   0 ]
```

Polynomials

NumPy

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add A*B(1) to C(1:4)

$+ (6x^3 + 5x^2 - 3x + 7)*2x$ add A*B(2) to C(2:5)

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$

The result is 5th order, so we need a 1-by-6 array for the result

`C = [ 18 27 1 15 14 0 ]`

Polynomials

NumPy

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add A*B(1) to C(1:4)

$+ (6x^3 + 5x^2 - 3x + 7)*2x$ add A*B(2) to C(2:5)

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$ **add A*B(3) to C(3:6)**

The result is 5th order, so we need a 1-by-6 array for the result

```
C = [  18  27   1  15  14   0 ]
```

Polynomials

NumPy

Multiplying

$a(x) = 6x^3 + 5x^2 - 3x + 7$ `A = [6 5 -3 7]`

$b(x) = 3x^2 + 2x - 4$ `B = [3 2 -4]`

Express the product as

$(6x^3 + 5x^2 - 3x + 7)(3x^2 + 2x - 4) =$

$(6x^3 + 5x^2 - 3x + 7)*3x^2$ add A*B(1) to C(1:4)

$+ (6x^3 + 5x^2 - 3x + 7)*2x$ add A*B(2) to C(2:5)

$+ (6x^3 + 5x^2 - 3x + 7)*(-4)$ add A*B(3) to C(3:6)

The result is 5^{th} order, so we need a 1-by-6 array for the result

```
C = [  18  27  -23  -5  26  -28 ]
```

Polynomials

```
import numpy as np

P = np.polynomial

a = P.Polynomial([6, 5, -3, 7])
b = P.Polynomial([0, 3, 2, -4])

print(np.polymul(a, b))
```

```
[Polynomial([ 0., 18., 27., -23., -5., 26., -28.], domain=[-1.,1.],window=[-1., 1.])]
```

Polynomials

NumPy

Use the numpy function **polyval**

```
import numpy as np

P = np.polynomial
x = 2.6
y = P.polynomial.polyval(x, [6, 5, -3, 7])
```

polyval works for vectors too

```
import numpy as np

P = np.polynomial
x = np.linspace(-3,3,200)
y = P.polynomial.polyval(x, [6, 5, -3, 7])
```

Polynomials

NumPy

An n^{th} order polynomial (with nonzero leading coefficient)

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_n x + a_{n+1}$$

It is a fundamental theorem of algebra that the equation

$$p(x) = 0$$

has **n** solutions, called the <u>roots of *p*</u>. Label these roots r_1, r_2, … , r_n. Another way to say this is that **p** can be factored into the for

$$p(x) = a_1 (x - r_1)(x - r_2) \cdots (x - r_n)$$

Even if the coefficients of **p** are real numbers, the roots may be complex.

Polynomials

The command **roots** computes the roots of a polynomial, and returns them as a column vector.

Compute the roots of

$$p(x) = x^2 + x - 2$$

$$q(x) = x^2 + 3x + 5$$

$$v(x) = 2x^4 - x^3 + 4x^2 + x - 3$$

```
import numpy as np

P = np.polynomial

print(P.polynomial.polyroots([1, 1, -2]))
print(P.polynomial.polyroots([1, 3, 5]))
print(P.polynomial.polyroots([2, -1, 4, 1, -3]))
```

Solving Linear Equations

Consider *n* equations in *m* unknowns.

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1m}x_m = b_1$$
$$A_{21}x_1 + A_{22}x_2 + \cdots + A_{2m}x_m = b_2$$
$$\vdots$$
$$A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nm}x_m = b_n$$

Think of the A_{ij} as <u>known</u> coefficients, and the b_i as <u>known</u> numbers. The goal is to <u>solve</u> for all of the unknowns x_j

Solving Linear Equations

Example of Linear Equations

Intersection of two lines

Simple truss structures

- Consist of beams
- Frictionless "pin" joints

Heat Transfer through conductive material

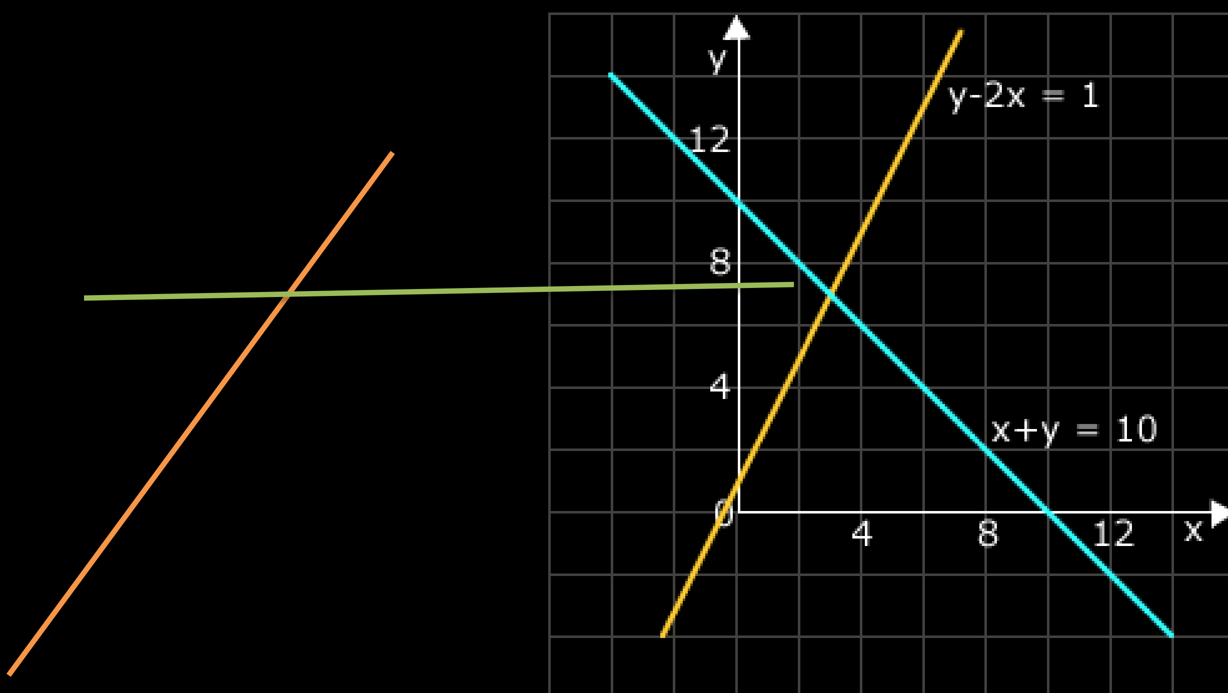
Electrical current flow through resistive network

Getting proper balance of nutrients from selection of foods

Solving Linear Equations

Example of Linear Equations

Intersection of two lines



$$y - 2x = 1$$

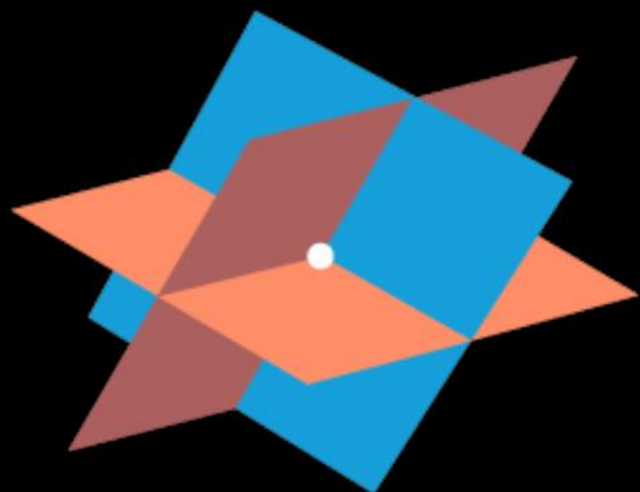
$$y + x = 10$$

Solving Linear Equations

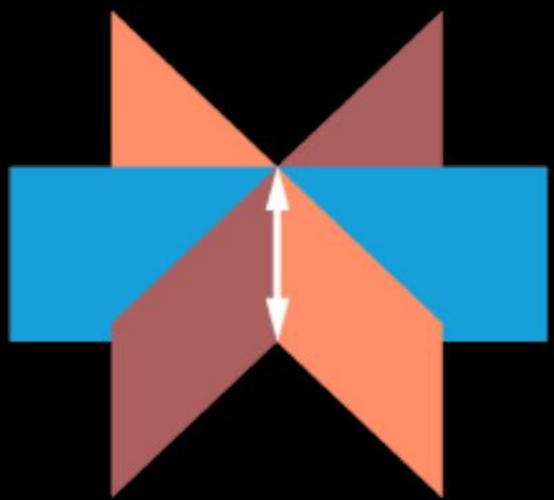
Example of Linear Equations

$$Ax + By + Cz = D$$
$$\mathrm{E}y + Fz = G$$
$$\mathrm{H}z = K$$

Intersection of three planes



Three planes intersect at a single point, representing a three-by-three system with a single solution.



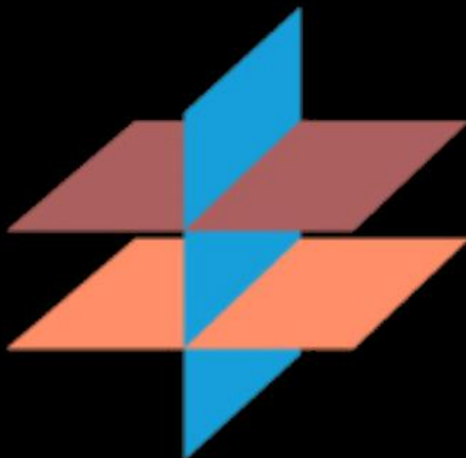
Three planes intersect in a line, representing a three-by-three system with infinite solutions

Solving Linear Equations

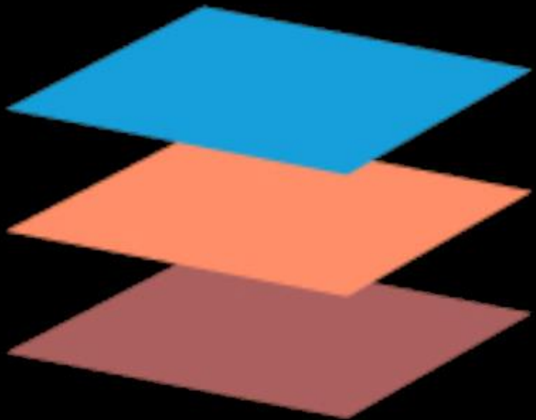
Example of Linear Equations

Intersection of three planes

The three planes intersect with each other, but not at a common point.



Two of the planes are parallel and intersect with the third plane, but not with each other.

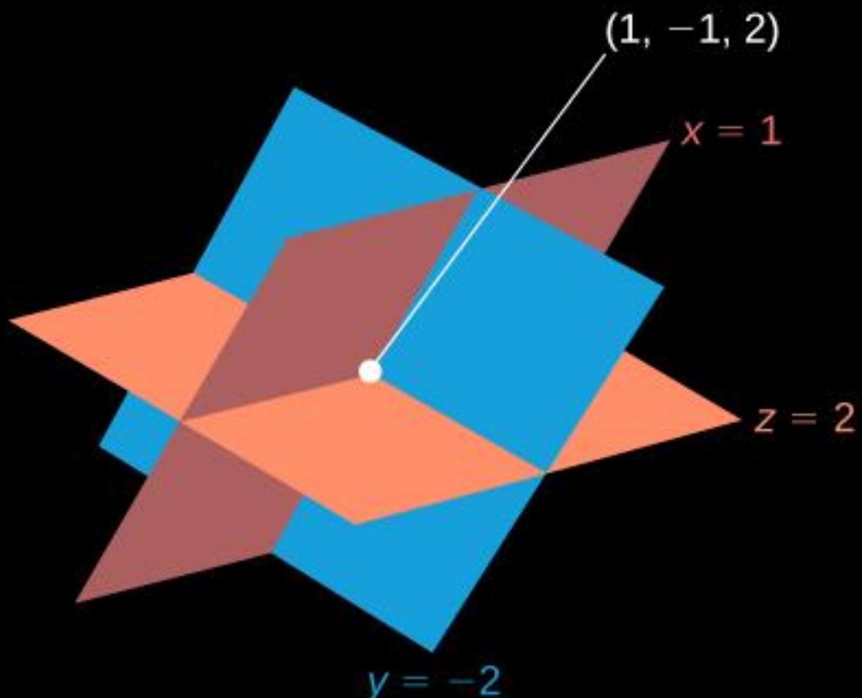


All three planes are parallel, so there is no point of intersection.

Solving Linear Equations

Example of Linear Equations

Intersection of three planes



$$x - 2y + 3z = 9$$
$$-x + 3y - z = -6$$
$$2x - 5y + 5z = 17$$

Solving Linear Equations

```
import numpy as np

my_matrix = np.array([[1 , -2 , 3] ,
                       [-1 , 3 , -1] ,
                       [2 , -5 , 5]])

my_vector = np.array([9 , -6 , 17])

solution = np.linalg.solve(my_matrix, my_vector)
print('Solution is x , y , z = ' , solution )

# Alternative method
solution1 = np.matmul(np.linalg.inv(my_matrix ), my_vector)
print('Solution is x1 , x2 , x3 by inv [ A ]* y = ', solution1)

Solution is x , y , z =  [ 1. -1.  2.]
Solution is x1 , x2 , x3 by inv [ A ]* y =  [ 1. -1.  2.]
```

Solving Linear Equations

```
import numpy as np

A = np.array([[2 , -3 , 1] ,
              [1 , -1 , 2] ,
              [3 ,  1 ,-1]])

b = np.array([-1 , -3 , 9])

solution1 = np.linalg.solve(A, b)
print('Solution1 is x , y , z = ' , solution1 )
# Alternative method
solution2 = np.dot(np.linalg.inv(A ), b)
print('Solution2 is x1 , x2 , x3 by inv [ A ]* y = ', solution2)
print('Determinant of A ', np.linalg.det(A))
Solution1 is x , y , z =  [ 2. 1.  -2.]
Solution2 is x1 , x2 , x3 by inv [ A ]* y =  [ 2. 1.  -2.]
Determinant of A  -19.000000000000004
```

$$\begin{cases} 2x - 3y + z = -1 \\ x - y + 2z = -3 \\ 3x + y - z = 9 \end{cases}$$

Solving Linear Equations With No Solution

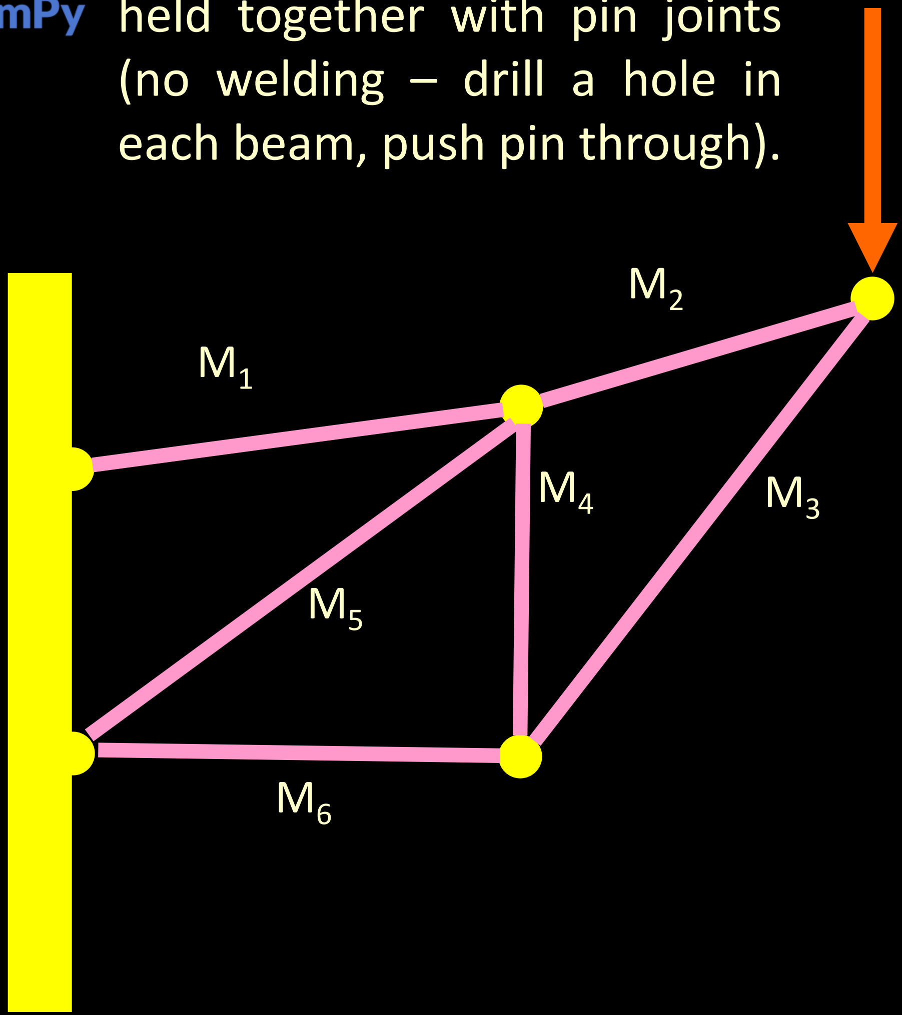
```
import numpy as np

A = np.array([[1 , -1 , 4] ,
              [0 ,  0 , 1] ,
              [-1,  1 ,-4]])

b = np.array([-5 , 0 , 20])

solution1 = np.linalg.solve(A, b)
print('Determinant of A ', np.linalg.det(A))

print('Solution1 is x , y , z = ' , solution1 )
# Alternative method
solution2 = np.dot(np.linalg.inv(A ), b)
print('Solution2 is x1 , x2 , x3 by inv [ A ]* y = ', solution2)
Determinant of A  0.0
LinAlgError: Singular matrix
```

$$\begin{cases} x - y + 4z = -5 \\ 3x + z = 0 \\ -x + y - 4z = 20 \end{cases}$$

Solving Linear Equations

Truss members are beams held together with pin joints (no welding – drill a hole in each beam, push pin through).

Pins transfer force between beams. If the truss is in equilibrium, all forces acting on a pin must sum to zero.



Simple truss analysis: Basic Concepts

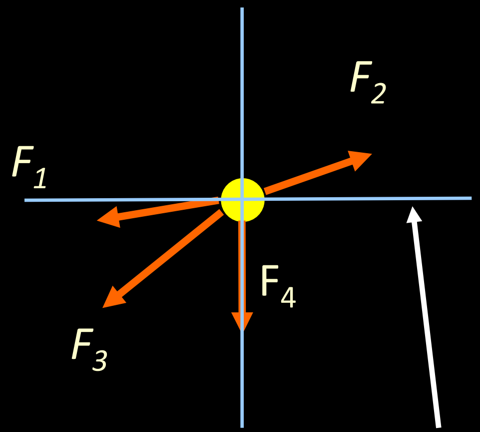
Beams only have a force acting at each end, no moments. These are called 2-force members. If the truss is in eqilibrium, total force on beam must be 0, and there cannot be a torque on the beam.

Solving Linear Equations

NumPy

Force balance on a pin

Draw a free-body diagram of a given pin. The forces acting on it are the forces from the members (Newton's 3rd law)



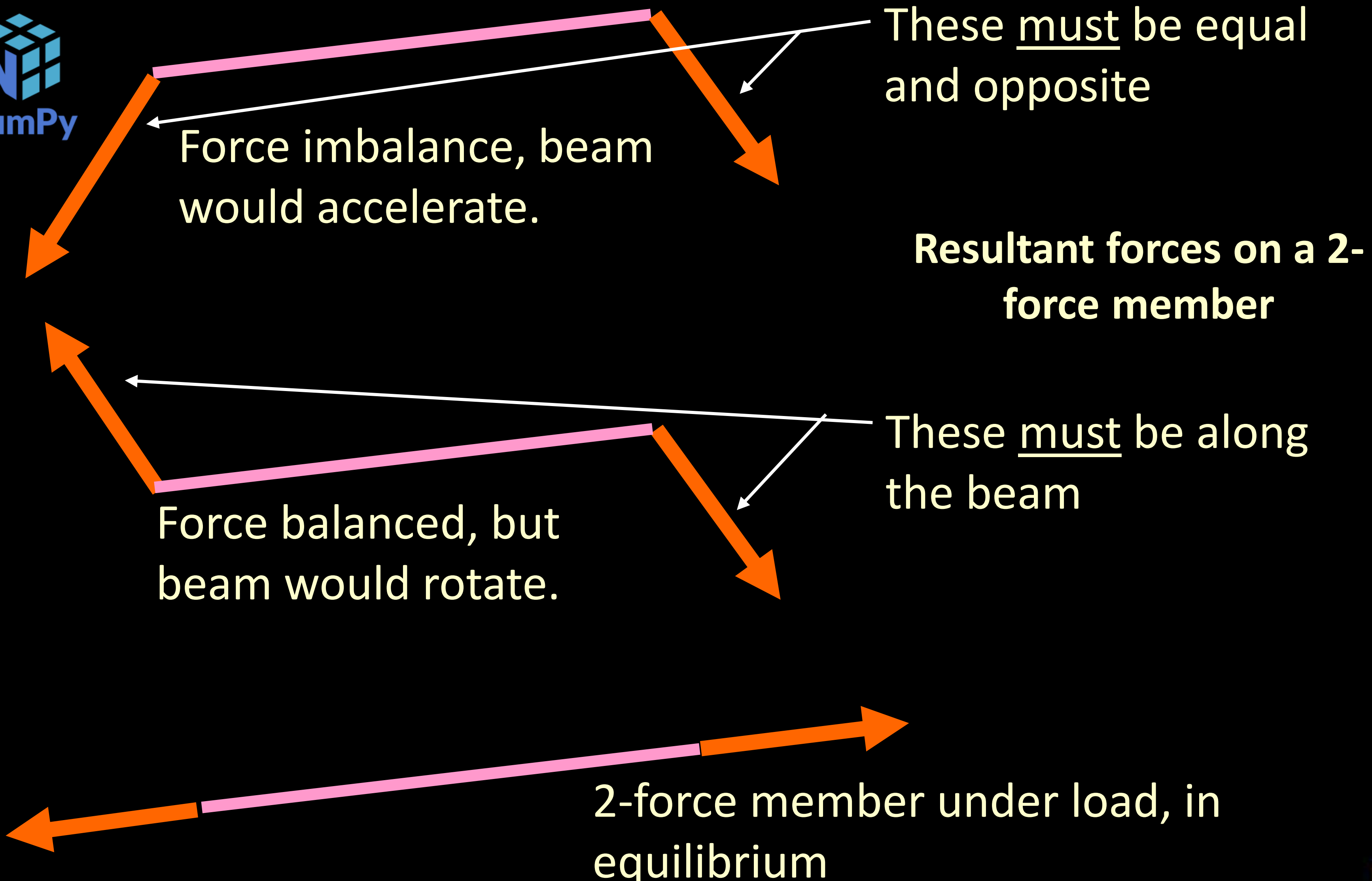
Measure θ from here (eg.)

Sum forces on pin in horizontal and vertical directions. For equilibrium, forces must sum to zero.

$$F_1 \cos\theta_1 + F_2 \cos\theta_2 + F_3 \cos\theta_3 + F_4 \cos\theta_4 = 0$$

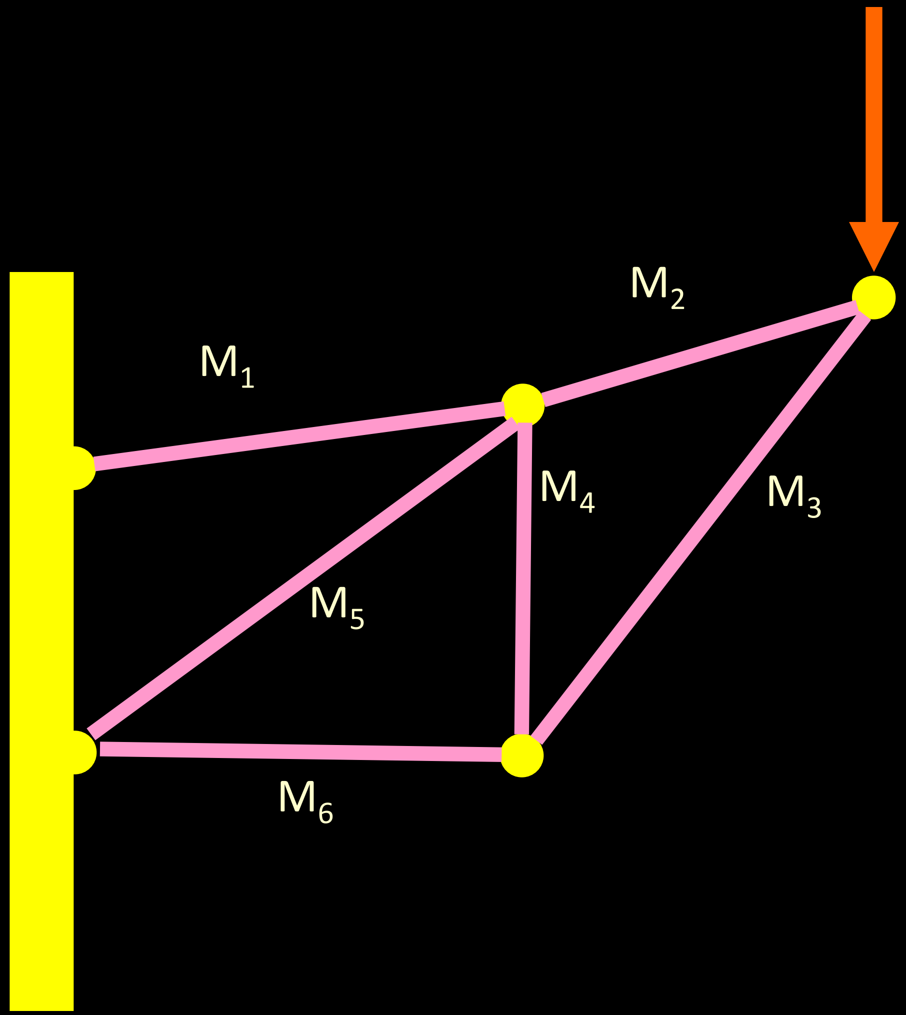
$$F_1 \sin\theta_1 + F_2 \sin\theta_2 + F_3 \sin\theta_3 + F_4 \sin\theta_4 = 0$$

Solving Linear Equations



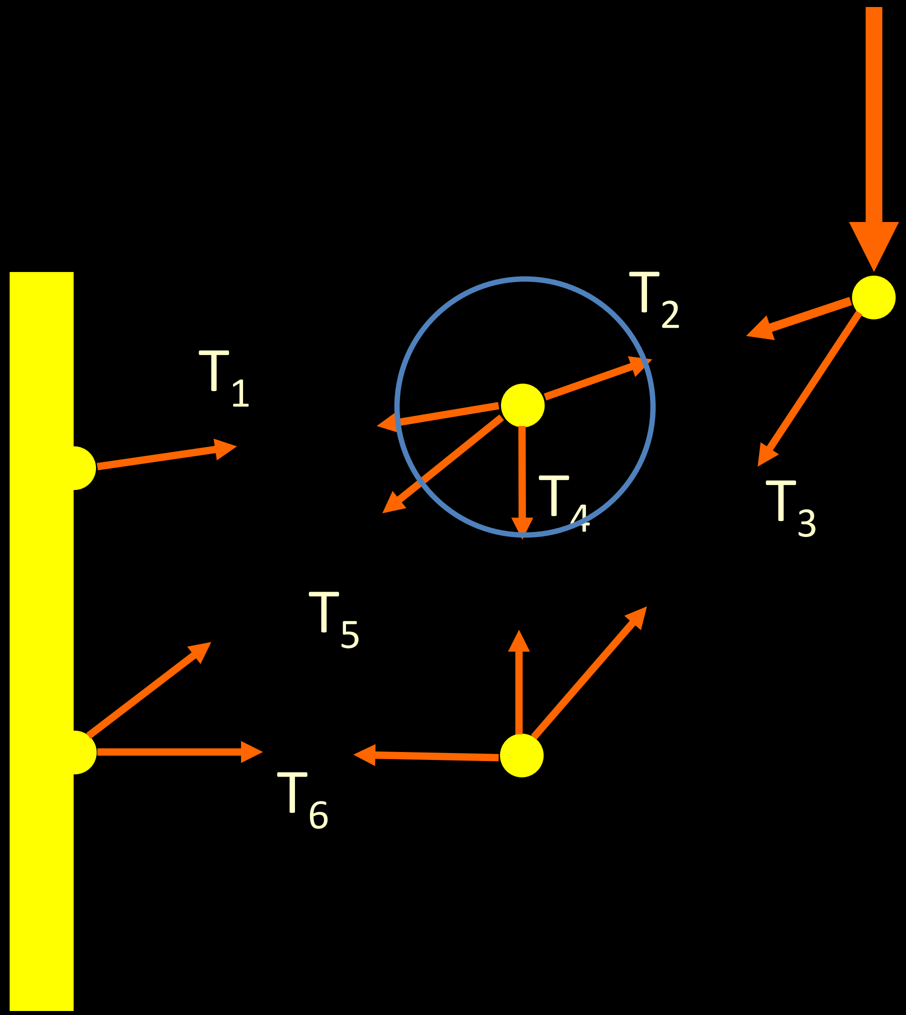
Solving Linear Equations

NumPy



Let $\mathbf{T}_i$ be the force in member $\mathbf{M}_i$.

Solving Linear Equations

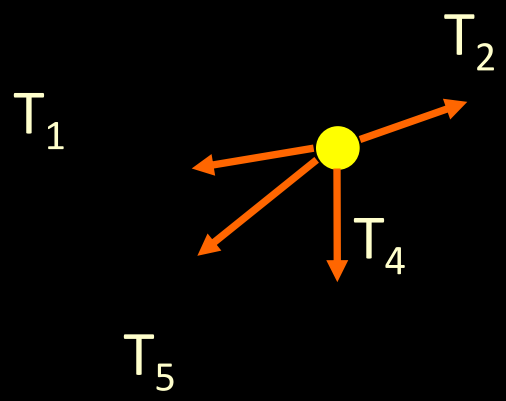


Free-body diagrams on each pin

Solving Linear Equations

NumPy

Force balance on a pin



Sum forces on pin in horizontal and vertical directions

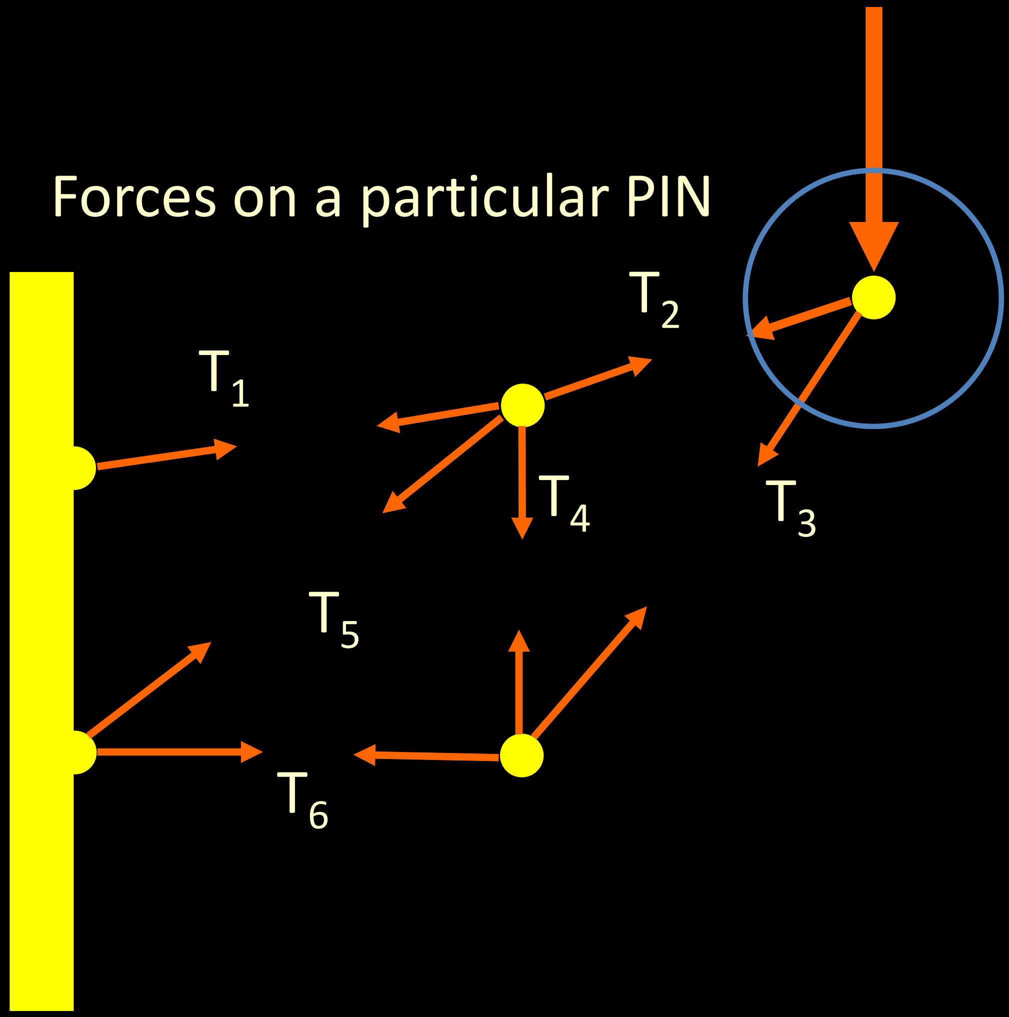
$$T_1 \cos\theta_1 + T_2 \cos\theta_2 + T_4 \cos\theta_4 + T_5 \cos\theta_5 = 0$$

$$T_1 \sin\theta_1 + T_2 \sin\theta_2 + T_4 \sin\theta_4 + T_5 \sin\theta_5 = 0$$

Solving Linear Equations

NumPy

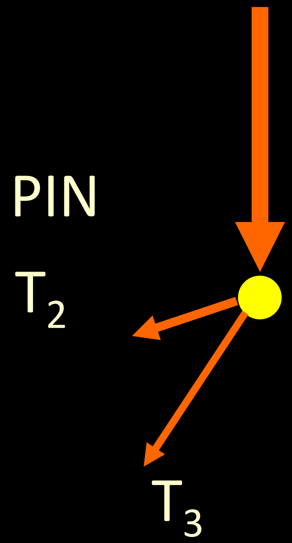
Forces on a particular PIN



Solving Linear Equations

NumPy

Forces on a particular PIN

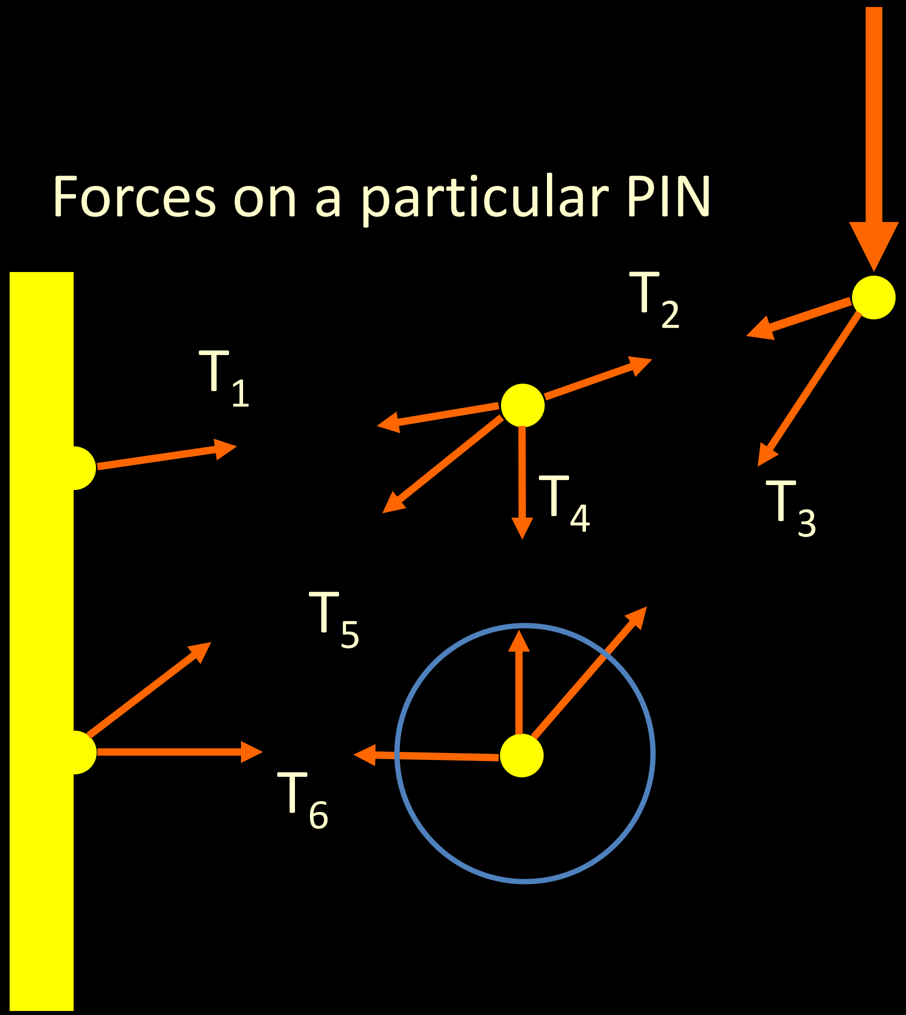


$$T_2 \cos\phi_2 + T_3 \cos\phi_3 = 0$$

$$-P + T_2 \sin\phi_2 + T_3 \sin\phi_3 = 0$$

Solving Linear Equations

Forces on a particular PIN

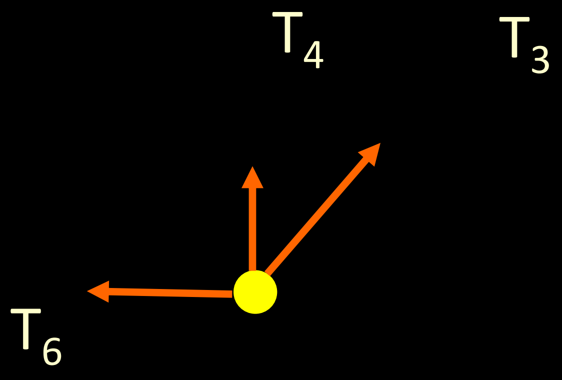


Solving Linear Equations

$$T_3 \cos\psi_3 + T_4 \cos\psi_4 + T_6 \cos\psi_6 = 0$$

$$T_3 \sin\psi_3 + T_4 \sin\psi_4 + T_6 \sin\psi_6 = 0$$

Forces on a particular PIN



Solving Linear Equations

NumPy

$$T_3 \cos\psi_3 + T_4 \cos\psi_4 + T_6 \cos\psi_6 = 0$$

$$T_3 \sin\psi_3 + T_4 \sin\psi_4 + T_6 \sin\psi_6 = 0$$

$$T_2 \cos\phi_2 + T_3 \cos\phi_3 = 0$$

$$-P + T_2 \sin\phi_2 + T_3 \sin\phi_3 = 0$$

$$T_1 \cos\theta_1 + T_2 \cos\theta_2 + T_4 \cos\theta_4 + T_5 \cos\theta_5 = 0$$

$$T_1 \sin\theta_1 + T_2 \sin\theta_2 + T_4 \sin\theta_4 + T_5 \sin\theta_5 = 0$$

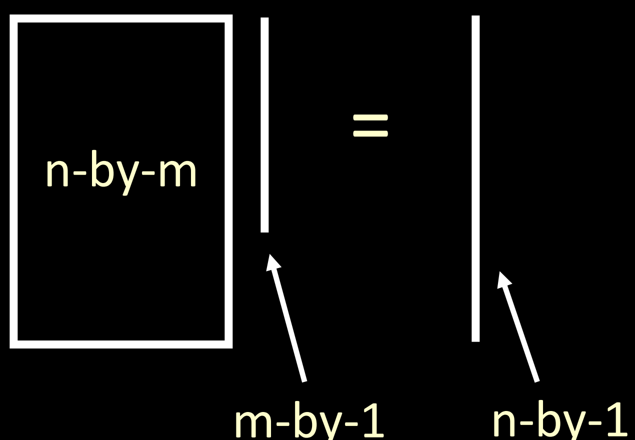
If geometry is fixed, and external load is known, then this is 6 equations, 6 unknowns. We need some “good notation” for linear equations….

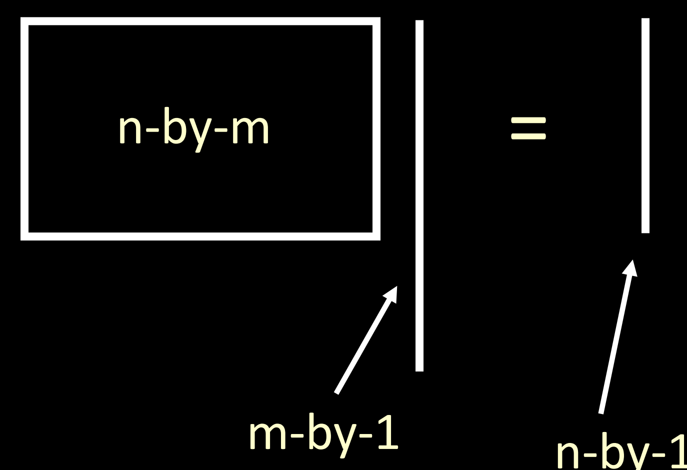
Solving Linear Equations

If A is an *n-by-m* array, and x is an *m-by-1* vector, then the "product Ax" is a *n-by-1* vector, whose i'th component is

Array-Vector multiplication

$$(Ax)_i = \sum_{j=1}^{m} A_{ij} x_j$$





Solving Linear Equations

NumPy If *A* is an *n-by-m* array, and *x* is an *m-by-1* vector, then the "product *Ax*" is a *n-by-1* vector, whose i'th component is

$$Ax = \begin{bmatrix} (Ax)_1 \\ (Ax)_2 \\ \vdots \\ (Ax)_n \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1m}x_m \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2m}x_m \\ \vdots \\ A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nm}x_m \end{bmatrix}$$

$$(Ax)_i = \sum_{j=1}^{m} A_{ij}x_j$$

Solving Linear Equations

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1m} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1m}x_m \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2m}x_m \\ \vdots \\ A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nm}x_m \end{bmatrix}$$

add, to give

$$\begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{n1} \end{bmatrix} [x_1] + \begin{bmatrix} A_{12} \\ A_{22} \\ \vdots \\ A_{n2} \end{bmatrix} [x_2] + \begin{bmatrix} A_{13} \\ A_{23} \\ \vdots \\ A_{n3} \end{bmatrix} [x_3] + \cdots + \begin{bmatrix} A_{1m} \\ A_{2m} \\ \vdots \\ A_{nm} \end{bmatrix} [x_m]$$

Solving Linear Equations

Consider *n* equations in *m* unknowns.

$$A_{11}x_1 + A_{12}x_2 + \cdots + A_{1m}x_m = b_1$$
$$A_{21}x_1 + A_{22}x_2 + \cdots + A_{2m}x_m = b_2$$
$$\vdots$$
$$A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nm}x_m = b_n$$

Collect

- The A_{ij} into an *n-by-m* array called *A*
- The b_i into a *n-by-1* vector called *b*, and
- The x_j into an *m-by-1* vector called *x*

Then the equations above can be written concisely as

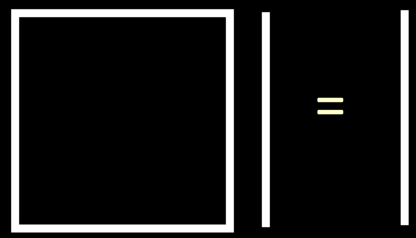
$$Ax = b$$

matrix/vector multiply

vector equality

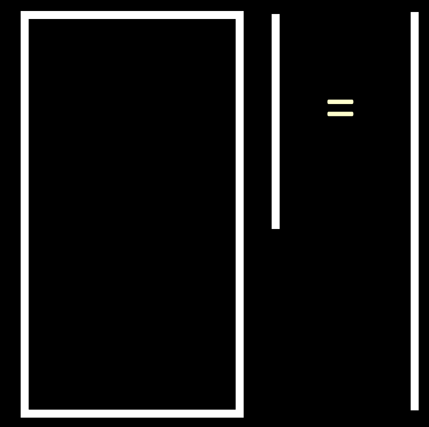
Solving Linear Equations

For the equation *Ax=b*, there are 3 distinct cases



Square, equal number of unknowns and equations

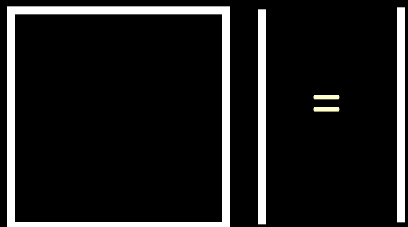
Underdetermined: more unknowns than equations



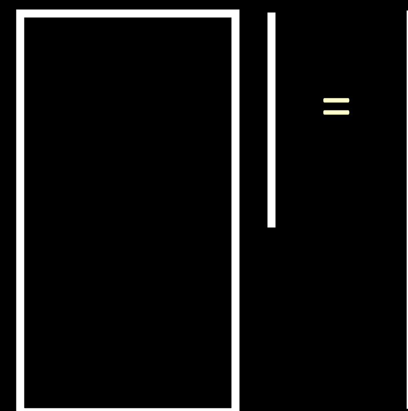
Overdetermined: fewer unknowns than equations

Solving Linear Equations

Types of solutions



One solution (eg., 2 lines intersect at one point)



Infinite solutions (eg., 2 planes intersect at many points)

No solutions (eg., 3 lines don’t intersect at a point)

Solving Linear Equations

$$T_3 \cos\psi_3 + T_4 \cos\psi_4 + T_6 \cos\psi_6 = 0$$

$$T_3 \sin\psi_3 + T_4 \sin\psi_4 + T_6 \sin\psi_6 = 0$$

$$T_2 \cos\phi_2 + T_3 \cos\phi_3 = 0$$

$$-P + T_2 \sin\phi_2 + T_3 \sin\phi_3 = 0$$

$$T_1 \cos\theta_1 + T_2 \cos\theta_2 + T_4 \cos\theta_4 + T_5 \cos\theta_5 = 0$$

$$T_1 \sin\theta_1 + T_2 \sin\theta_2 + T_4 \sin\theta_4 + T_5 \sin\theta_5 = 0$$

If geometry is fixed, and external load is known, then this is 6 equations, 6 unknowns.

Solving Linear Equations

NumPy

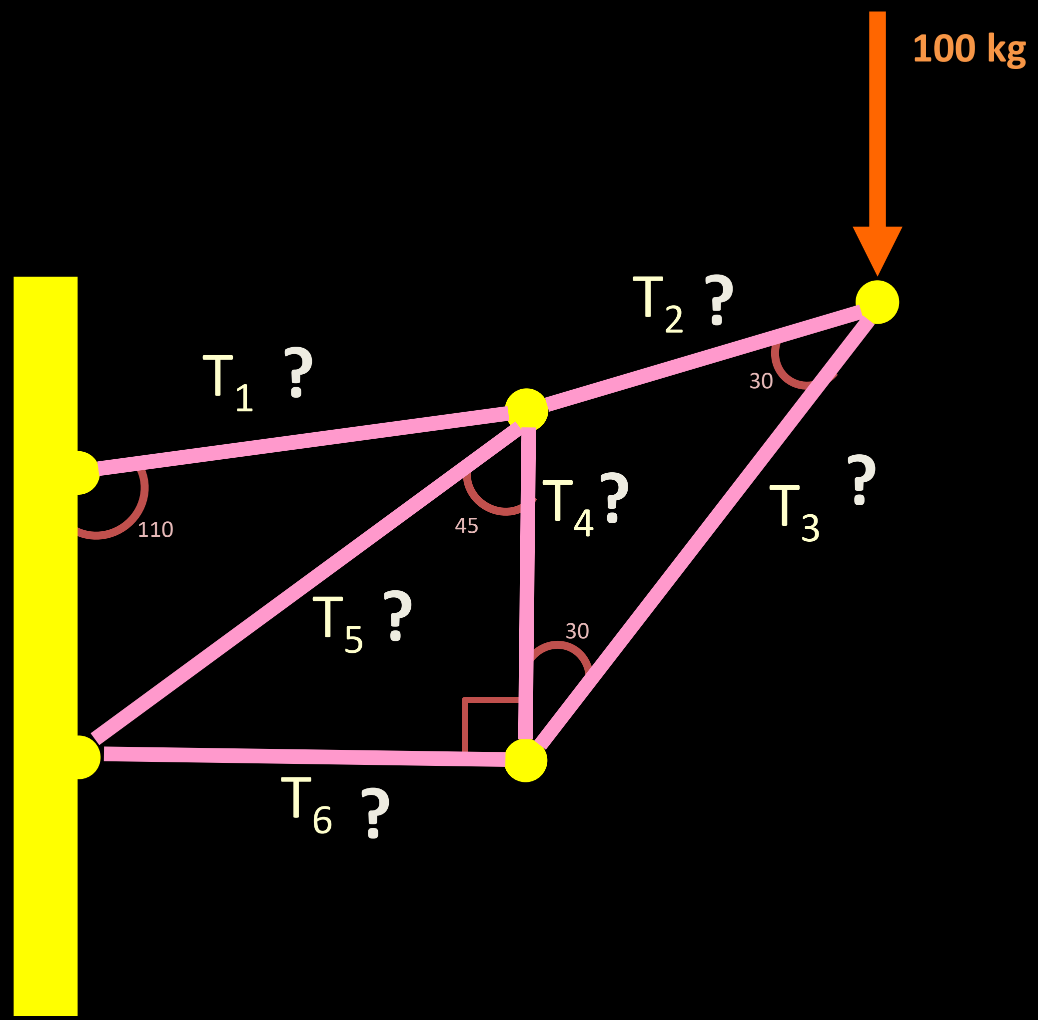
$$T_3 \cos\psi_3 + T_4 \cos\psi_4 + T_6 \cos\psi_6 = 0$$
$$T_3 \sin\psi_3 + T_4 \sin\psi_4 + T_6 \sin\psi_6 = 0$$
$$T_2 \cos\phi_2 + T_3 \cos\phi_3 = 0$$
$$-P + T_2 \sin\phi_2 + T_3 \sin\phi_3 = 0$$
$$T_1 \cos\theta_1 + T_2 \cos\theta_2 + T_4 \cos\theta_4 + T_5 \cos\theta_5 = 0$$
$$T_1 \sin\theta_1 + T_2 \sin\theta_2 + T_4 \sin\theta_4 + T_5 \sin\theta_5 = 0$$

In matrix/vector form

$$\begin{bmatrix} c_{\theta_1} & c_{\theta_2} & 0 & c_{\theta_4} & c_{\theta_5} & 0 \\ s_{\theta_1} & s_{\theta_2} & 0 & s_{\theta_4} & s_{\theta_5} & 0 \\ 0 & c_{\phi_2} & c_{\phi_3} & 0 & 0 & 0 \\ 0 & s_{\phi_2} & s_{\phi_3} & 0 & 0 & 0 \\ 0 & 0 & c_{\psi_3} & c_{\psi_4} & 0 & c_{\psi_6} \\ 0 & 0 & s_{\psi_3} & s_{\psi_4} & 0 & s_{\psi_6} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ P \\ 0 \\ 0 \end{bmatrix}$$

Solving Linear Equations

NumPy



Solving Linear Equations