UPPSALA
UNIVERSITET

Department of Information Technology

# Scientific Computing for Data Analysis

Davoud Mirzaei

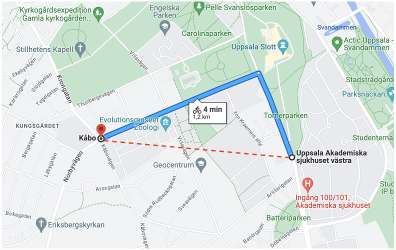
# Block 2
# Numerical Linear Algebra- I

# Lecture 5: Review of norms + regression analysis and least squares

## Agenda

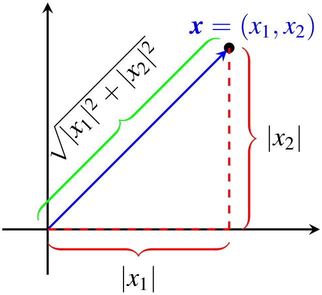
- Vector and matrix norm
- Stability and condition numbers
- Polynomial fitting
- Least squares problem and normal equation

- What is a norm? Norm means distance:




- Norms for vectors and matrices - Why?
  Problem: Measure size of vectors or matrices. What is “small” and what is “large”?
  Problem: Measure distance between vectors or matrices. When are they “close together” or “far apart”?
  Answers are given by norms
- Also: Tool to analyze convergence and stability of algorithms

# Vector norms




Some frequently used vector norms: Assume that $\boldsymbol{x} = (x_1, \ldots, x_n)$ is an $n$-vector:

- $\|\boldsymbol{x}\|_2 = \sqrt{|x_1|^2 + \cdots + |x_n|^2}$ (norm 2 or Euclidean norm)
- $\|\boldsymbol{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n|$ (norm 1 or sum norm)
- $\|\boldsymbol{x}\|_\infty = \max\limits_{1 \le k \le n} \{|x_k|\}$ (norm infinity or maximum norm)

In 1D all the norms are identical with $|x|$ (the absolute value of $x$)

Assume that $\boldsymbol{x}, \boldsymbol{y}$ are two vectors and $\alpha$ is a real number. Any norm must possess the following properties:

- $\|\boldsymbol{x}\| \geq 0$ and $\|\boldsymbol{x}\| = 0 \Longleftrightarrow \boldsymbol{x} = \boldsymbol{0}$ (positivity)
- $\|\alpha \boldsymbol{x}\| = |\alpha| \|\boldsymbol{x}\|$ (homogeneity)
- $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ (triangle inequality)

We can simply show that norm 1 and norm infinity satisfy these conditions. For norm 2 the only challenging part to prove might be the triangle inequality.

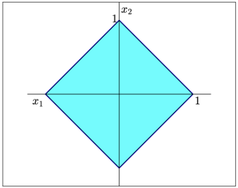
Note: Norm 2 is induced form the inner product

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \boldsymbol{x}^T \boldsymbol{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

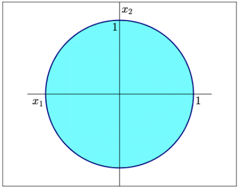
It is clear that

$$\|\boldsymbol{x}\|_2 = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$$

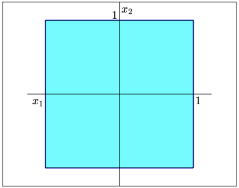
The unit ball $B = \{\boldsymbol{x} : \|\boldsymbol{x}\| \leq 1\}$ in different norms:




$\|x\|_1 \leq 1$
or
$|x_1| + |x_2| \leq 1$




$\|x\|_2 \leq 1$
or
$\left(|x_1|^2 + |x_2|^2\right)^{1/2} \leq 1$




$\|x\|_\infty \leq 1$
or
$\max\{|x_1|, |x_2|\} \leq 1$

# Matrix norms

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

For matrices we again look for some norm definitions with the same properties:

- $\|A\| \geq 0$ and $\|A\| = 0 \iff A = \mathbf{0}$ (positivity)
- $\|\alpha A\| = |\alpha| \|A\|$ (homogeneity)
- $\|A + B\| \leq \|A\| + \|B\|$ (triangle inequality)

and usually with two additional properties:

- $\|A\boldsymbol{x}\|_* \leq \|A\| \|\boldsymbol{x}\|_*$ for some vector norm $\|\cdot\|_*$ (consistency)
- $\|AB\| \leq \|A\| \|B\|$ (submultiplicativity)

These additional properties are sometimes necessary because when dealing with products, matrices differ from $mn$-vectors!

# Matrix norms

Example: Let $\|A\|_{max} := \max_{i,j} |a_{ij}|$. It satisfies the first third properties but (at least) not the last property. For example

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad then \quad AB = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$$

We see that $\|A\|_{max} = 4,\ \|B\|_{max} = 1,\ \|AB\|_{max} = 7$ thus

$$\|AB\| > \|A\|\|B\|$$

Example: But if we define $\|A\|_F := \sqrt{\sum_{i,j} |a_{ij}|^2}$ then it satisfies all 5 properties. (The fourth property with $\|\boldsymbol{x}\|_* = \|\boldsymbol{x}\|_2$). This norm is called the Frobenius norm (It is not norm 2 of a matrix).

$$A = \begin{bmatrix} 5 & -4 & 2 \\ -1 & 2 & 3 \\ -2 & 1 & 0 \end{bmatrix}, \quad \|A\|_F = \sqrt{25+16+4+1+4+9+4+1+0} = 8$$

# Matrix norms

Example (continued): Norm Frobenius is consistent with the Euclidean vector norm (norm 2): $\|A\boldsymbol{x}\|_2 \leqslant \|A\|_F \|\boldsymbol{x}\|_2$ for all vectors $\boldsymbol{x}$ of size $n$.

$$A = \begin{bmatrix} 5 & -4 & 2 \\ -1 & 2 & 3 \\ -2 & 1 & 0 \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} -3 \\ 1 \\ -1 \end{bmatrix}$$

$$A\boldsymbol{x} = \begin{bmatrix} 3 \\ 12 \\ 0 \end{bmatrix} \quad A\boldsymbol{y} = \begin{bmatrix} -21 \\ 2 \\ 7 \end{bmatrix}$$

$$\|A\boldsymbol{x}\|_2 = \sqrt{153}, \quad \|A\boldsymbol{y}\|_2 = \sqrt{494}, \quad \|\boldsymbol{x}\|_2 = \sqrt{14}, \quad \|\boldsymbol{y}\|_2 = \sqrt{11}, \quad \|A\|_F = 8$$

- $\sqrt{153} = \|A\boldsymbol{x}\|_2 \leqslant \|A\|_F \|\boldsymbol{x}\|_2 = 8\sqrt{14}$
- $\sqrt{494} = \|A\boldsymbol{y}\|_2 \leqslant \|A\|_F \|\boldsymbol{y}\|_2 = 8\sqrt{11}$

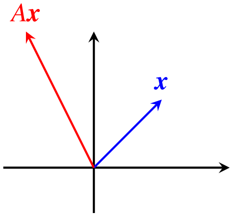
Inequality holds true not only for these two vectors but also for all vectors of size $3 \times 1$. (we skip the math. proof!)

## Natural norms

We can define a class of matrix norms via vector norms. If we apply a matrix $A$ on a vector $\boldsymbol{x}$ we get a new vector $A\boldsymbol{x}$ perhaps with a different length and a different direction:

$$A = \begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

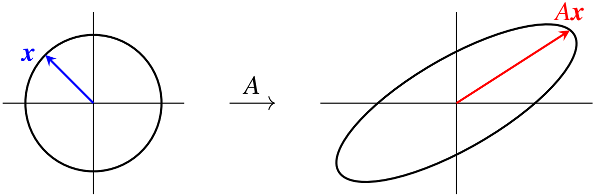
$$A\boldsymbol{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$




- Matrix $A$ scales (stretches or shrinks) and rotates $\boldsymbol{x}$

# Matrix norms

Consider a set of all vectors $\boldsymbol{x}$ with $\|\boldsymbol{x}\| = 1$ (e.g. in norm 2: all vector with heads on the unit circle)




The norm of $A$ is identified by measuring “how much $A$ stretches the unit vectors?”

The definition:

$$\|A\|_p := \max_{\boldsymbol{x}:\|\boldsymbol{x}\|=1} \|A\boldsymbol{x}\|_p, \quad p = 1, 2, \ldots, \infty$$

We can prove that such definition is indeed a matrix norm which satisfies all 5 properties above.

This norm is called $p$-norm or natural norm of matrix $A$

## Matrix norms

It is impossible to compute the $p$-norm of a matrix by directly using the definition above. However, form the definition we can prove that

- $\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|$
- $\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| = \|A^T\|_1$
- $\|A\|_2 = \sqrt{\lambda_{max}(A^TA)}$, ($\lambda_{max}$ maximum eigenvalue)

Example:

$$A = \begin{bmatrix} 0 & -1 \\ 5 & -3 \\ -2 & 1 \end{bmatrix}, \qquad \begin{array}{l} \|A\|_1 = \max\{7,5\} = 7 \\ \|A\|_\infty = \max\{1,8,3\} = 8 \\ \|A\|_2 = ? \end{array}$$

$$A^TA = \begin{bmatrix} 29 & -17 \\ -17 & 11 \end{bmatrix} \Longrightarrow \lambda_1 \doteq 0.7646,\ \lambda_2 \doteq 39.2354$$

$$\Longrightarrow \|A\|_2 \doteq \sqrt{39.2354} \doteq 6.2638$$

# Matrix norms

The matrix $p$ norm is consistent with the vector $p$ norm (why?):

$$\|A\boldsymbol{x}\|_p \leqslant \|A\|_p \|\boldsymbol{x}\|_p, \quad p = 1, 2, \ldots, \infty$$

Also we can show (how?)

$$\|AB\|_p \leqslant \|A\|_p \|B\|_p$$

Some Exercises:

- Show that $\|\boldsymbol{x}\|_\infty \leqslant \|\boldsymbol{x}\|_2$ for all vectors $\boldsymbol{x}$.
- Show that $\|\boldsymbol{x}\|_\infty \leqslant \|\boldsymbol{x}\|_1$ for all vectors $\boldsymbol{x}$.
- If $A$ is symmetric then $\|A\|_1 = \|A\|_\infty$
- If $A$ is symmetric then $\|A\|_2 = \max\{|\lambda_k(A)|\}$
- Show that $\|A\|_F = \sqrt{\mathrm{trace}(A^T A)}$ (The trace of a matrix is the sum of its diagonals)

# In Python ...

The code for norms of vectors and matrices are provided in `numpy.linalg.norm` library:

```
import numpy as np

x = np.array([5,-2,1])                    # x of size 3x1
A = np.array([[1,2,3],[4,-1,-2],[0,3,2]])  # A of size 3x3

Norm1_x = np.linalg.norm(x,1)          # norm 1 of x
Norm2_x = np.linalg.norm(x,2)          # norm 2 of x
NormInf_x = np.linalg.norm(x,np.inf)  # norm infinity of x

Norm1_A = np.linalg.norm(A,1)          # norm 1 of A
Norm2_A = np.linalg.norm(A,2)          # norm 2 of A
NormInf_A = np.linalg.norm(A,np.inf)  # norm infinity of A
NormFro_A = np.linalg.norm(A,'fro')   # norm Frobenius of A

Norm_x = np.linalg.norm(x)  # (default) norm 2 of x
Norm_A = np.linalg.norm(A)  # (default) norm Frobenius of x
```

$$Input + \delta \longrightarrow \boxed{Model} \longrightarrow Output + \epsilon$$

- The input data of a model (problem) is always subjected to some perturbations (like measurement errors, roundoff errors)
- The fundamental question then is:

  **How much the solution of the perturbed problem is close to the solution of the original problem?**

- If the sensitivity is low, the model is referred to as well-conditioned; otherwise, it is termed ill-conditioned
- If we replace the term “model" with “method" then then we say the method is either stable or unstable

Example: Consider $A\boldsymbol{x} = \boldsymbol{b}$ for $n \times n$ nonsingular matrix $A$ and $n$-vector $\boldsymbol{b}$. Let $A$ is Hilbert matrix:

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \cdots & \frac{1}{2n-1} \end{bmatrix} \tag{1}$$

and $\boldsymbol{x} = [1, 1, \ldots, 1]^T$ and $\boldsymbol{b} = A\boldsymbol{x}$. Now given that $\boldsymbol{b}$ find $\boldsymbol{x}$ in Python:

```
import numpy as np
import scipy as sp
n = 11
A = sp.linalg.hilbert(n)
b, x = np.sum(A, axis = 0), np.ones(n)
x_hat = np.linalg.solve(A,b)
print("x_hat = ", x_hat)
err = np.linalg.norm(x-x_hat,np.inf)/np.linalg.norm(x,np.inf)
print('RelativeError = ', err)
```

## Example: conditioning of linear systems

We expect a solution $\widehat{\boldsymbol{x}}$ close to exact solution $\boldsymbol{x} = [1, 1, \ldots, 1]^T$ but the output is:

```
x_hat = [0.99999999 1.00000071 0.99998152 1.0002066
         0.99876902 1.00432858 0.99057385 1.01285236
         0.98932278 1.00494066 0.99902392]

RelativeError =  0.01285235836299159
```

- Compare the relative error to the machine epsilon $\texttt{eps} \approx 10^{-16}$, fifteen significant digits are lost!
- Here $n = 11$ is small. Much more terrible results for larger $n$
- What is the source of this disaster? The problem $A\boldsymbol{x} = \boldsymbol{b}$ itself or the algorithm of `solve` function in Python?
- The algorithm is stable but the system $A\boldsymbol{x} = \boldsymbol{b}$ with $A$ Hilbert matrix is ill-conditioned, small perturbation in inputs (here $A$ and $\boldsymbol{b}$) results in a huge error in the solution (here $\boldsymbol{x}$). The error was amplified by a factor of $10^{15}$

## Condition number of a matrix

- Consider the system $A\boldsymbol{x} = \boldsymbol{b}$ and the perturbed system $A(\boldsymbol{x} + \boldsymbol{\varepsilon}) = \boldsymbol{b} + \boldsymbol{\delta}$ we get $\boldsymbol{\varepsilon} = A^{-1}\boldsymbol{\delta}$. (Assuming perturbation only in $\boldsymbol{b}$ for simplicity)
- Take norm from both sides:

$$\|\boldsymbol{\varepsilon}\| = \|A^{-1}\boldsymbol{\delta}\| \leqslant \|A^{-1}\|\|\boldsymbol{\delta}\|$$

- From original system $A\boldsymbol{x} = \boldsymbol{b}$ we have $\|\boldsymbol{b}\| \leqslant \|A\|\|\boldsymbol{x}\|$, or

$$\frac{1}{\|\boldsymbol{x}\|} \leqslant \frac{\|A\|}{\|\boldsymbol{b}\|}$$

- Multiplying both sides of recent equations we get

$$\frac{\|\boldsymbol{\varepsilon}\|}{\|\boldsymbol{x}\|} \leqslant \|A\| \cdot \|A^{-1}\| \frac{\|\boldsymbol{\delta}\|}{\|\boldsymbol{b}\|}$$

- At the worst case the error in the output is amplified by $\|A\| \cdot \|A^{-1}\|$. So the **condition number** of $A$ is defined as

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

Example:

$$A = \begin{bmatrix} 2 & 4 & -1 \\ 2 & 5 & 2 \\ -1 & -1 & 1 \end{bmatrix} \quad \xrightarrow{\textit{Verify!}} \quad A^{-1} = \frac{1}{5}\begin{bmatrix} -7 & 3 & -13 \\ 4 & -1 & 6 \\ -3 & 2 & -2 \end{bmatrix}$$

Thus

$$\mathrm{cond}_1(A) = \|A\|_1 \cdot \|A^{-1}\|_1 = 10 \cdot \frac{21}{5} = 42$$

$$\mathrm{cond}_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty = 9 \cdot \frac{23}{5} = 41.4$$

$$\mathrm{cond}_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \doteq 7.1561 \times 3.4184 \doteq 24.4621$$

- In the Hilbert example the roundoff error in $\boldsymbol{b}$ is of order machine epsilon (`eps`), i.e.

$$\frac{\|\boldsymbol{\delta}\|_\infty}{\|\boldsymbol{b}\|_\infty} = \texttt{eps} \approx 10^{-16}$$

  The condition number of a Hilbert matrix of size $11 \times 11$ is approximately $10^{+15}$. So we have

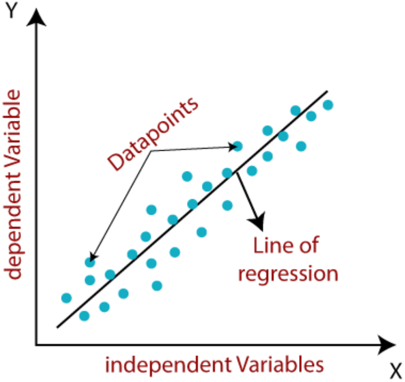
$$\frac{\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_\infty}{\|\boldsymbol{x}\|_\infty} \lesssim 10^{+15} \cdot 10^{-16} = 0.1$$

  which shows that about $15$ decimal digits will be lost. This confirms our observation from the Python code!
- Hilbert matrix is not the only ill-conditioned matrix, another example is Vandermonde matrix, and many other matrices.
- If $A$ is singular, even non-square, its condition number can be defined via SVD (later in the course)
- Lesson: avoid ill-conditioned matrices in your algorithms!

# Regression analysis and least squares

# Regression analysis

- Build regression model from (large) data sets
- Finding trend-lines in data
- Lots of regression in big data and machine learning areas
- The idea is to minimize distance between data points and the model, for example a polynomial



**Example: Linear least squares**

- Ansatz: $y = a_0 + a_1 x$
- $y$: dependent variable, predicted/measured outcome
- $x$: independent variable
- $a_0$, $a_1$: coefficients to be determined
- Approximate equations on $m$ points:

$$\begin{aligned} y_1 &\approx a_0 + a_1 x_1 \\ y_2 &\approx a_0 + a_1 x_2 \\ &\vdots \\ y_m &\approx a_0 + a_1 x_m \end{aligned} \implies \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \implies A\boldsymbol{a} \approx \boldsymbol{y}$$

## Example: Quadratic least squares

- Ansatz: $y = a_0 + a_1 x + a_2 x^2$
- Approximate equations on $m$ points:

$$\begin{aligned} y_1 &\approx a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 &\approx a_0 + a_1 x_2 + a_2 x_2^2 \\ &\vdots \\ y_m &\approx a_0 + a_1 x_m + a_2 x_m^2 \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \\ 1 & x_m & x_m^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$A\boldsymbol{a} \approx \boldsymbol{y}$$

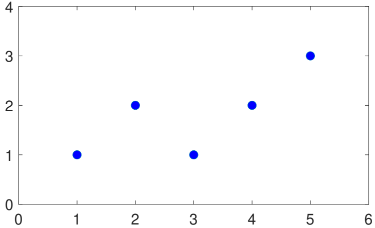
- The error is $\boldsymbol{e} = A\boldsymbol{a} - \boldsymbol{y}$
- in order to find the least squares solution (coefficient vector $\boldsymbol{a}$) we minimize the 2-norm of the error:

$$\min_{\boldsymbol{a} \in \mathbb{R}^3} \|A\boldsymbol{a} - \boldsymbol{y}\|_2$$

We will see how!

## Example: Quadratic least squares

$$
\begin{array}{c|ccccc}
 & x_1 & x_2 & x_3 & x_4 & x_5 \\
x & 1 & 2 & 3 & 4 & 5 \\
\hline
y & 1 & 2 & 1 & 2 & 3 \\
 & y_1 & y_2 & y_3 & y_4 & y_5
\end{array}
$$




$$
\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \\ 3 \end{bmatrix}
$$

Overdetermined system of equations (5 equations, 3 unknowns) $\Longrightarrow$ no exact solution

# Least squares fitting (Normal Equations)

The least squares problem leads to **normal equations** (why?):

$$\min_{\boldsymbol{a}\in\mathbb{R}^3} \|A\boldsymbol{a} - \boldsymbol{y}\|_2 \quad \Longrightarrow \quad A^T A\boldsymbol{a} = A^T\boldsymbol{y}$$

If $A$ is a full rank matrix then $A^TA$ is non-singular and the normal system has a unique solution.

## Again the Example:

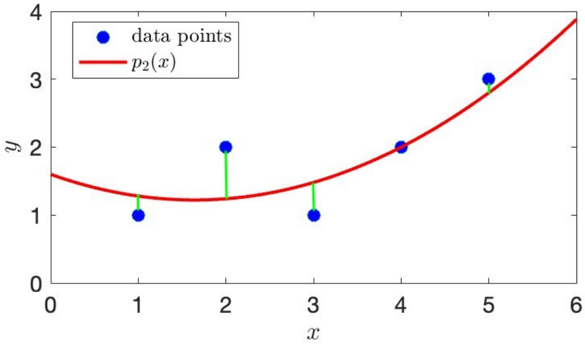
$$A^T A\boldsymbol{a} = A^T\boldsymbol{y} \Longrightarrow \begin{bmatrix} 5 & 15 & 55 \\ 15 & 55 & 225 \\ 55 & 225 & 979 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 9 \\ 31 \\ 125 \end{bmatrix}$$

Solve with Gaussian elimination yields (rounded to two decimals)

$$\boldsymbol{a} = \begin{bmatrix} 1.60 \\ -0.46 \\ 0.14 \end{bmatrix}$$

Plug solution into the ansatz:

$$
\begin{aligned}
p_2(x) &= a_0 + a_1 x + a_2 x^2 \\
&= 1.6 - 0.46x + 0.14x^2
\end{aligned}
$$




We minimize the sum of squares of vertical distances (green vertical lines), so the method is called **least squares**

Steps for a polynomial ansatz are:

- Substitute the data $(x_1, y_1)$, $(x_2, y_2)$, $\ldots$, $(x_m, y_m)$, into the ansatz and form matrix $A$ and vector value $\boldsymbol{y}$
- Solve the normal equations $A^T A\boldsymbol{a} = A^T\boldsymbol{y}$ (as $A$ overdetermined)
- Substitute the solution $\boldsymbol{a}$ into the ansatz to get the final least squares polynomial

In principle you can use any polynomial (or other function) as ansatz. But some are better than others.

## Least squares fitting via normal equations

Problems with normal system:

- Significant digits may be lost when $A^T$ is multiplied by $A$. Example:

$$A = \begin{bmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{bmatrix} \Longrightarrow A^T A = \begin{bmatrix} 1+\epsilon^2 & 1 \\ 1 & 1+\epsilon^2 \end{bmatrix}$$

  In the double precision arithmetic if $\epsilon < 10^{-8}$ then $fl(1+\epsilon^2) = 1$, i.e.

$$A^T A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

  which is singular.

- The condition number is increased:

$$\mathrm{cond}_2(A^T A) = [\mathrm{cond}_2(A)]^2$$

  In our example: $\mathrm{cond}_2(A^T A) = 0.7378 \cdot 10^4$ while $\mathrm{cond}_2(A) = 0.8589 \cdot 10^2$.
  Of course $10^4$ is a large cond. number for a $3 \times 3$ matrix!

# Shifting to improve the conditioning

Shift the ansatz by the mean of data: $p_2(x) = a_0 + a_1(x-\overline{x}) + a_2(x-\overline{x})^2$
The same example:

| $x$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $y$ | 1 | 2 | 1 | 2 | 3 |

Since $\overline{x} = 3$ we get

$$\begin{bmatrix} 1 & x_1-3 & (x_1-3)^2 \\ 1 & x_2-3 & (x_2-3)^2 \\ 1 & x_3-3 & (x_3-3)^2 \\ 1 & x_4-3 & (x_4-3)^2 \\ 1 & x_5-3 & (x_5-3)^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 11 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \\ 3 \end{bmatrix}$$

## Shifting to improve the conditioning

Normal equations become (after shift):

$$A^TA\boldsymbol{a} = A^T\boldsymbol{y} \Longrightarrow \begin{bmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 9 \\ 4 \\ 20 \end{bmatrix}$$

Solve with Gaussian elimination yields (rounded to two decimals)

$$\boldsymbol{a} = \begin{bmatrix} 1.51 \\ 0.40 \\ 0.14 \end{bmatrix}$$

Plug solution into the ansatz:

$$\begin{aligned} p_2(x) &= a_0 + a_1(x-3) + a_2(x-3)^2 \\ &= 1.51 + 0.4(x-3) + 0.14(x-3)^2 \\ &= 1.6 - 0.46x + 0.14x^2 \end{aligned}$$

Solutions are the same but $\mathrm{cond}_2(A^TA) = 19.7$ (compare $\mathrm{cond}_2(A^TA) = 7377.7$ for other ansatz)

## Shift and scale to improve the conditioning

Shift and scale the ansatz by the mean and standard deviation of data:

$$p_2(x) = a_0 + a_1 \frac{x - \bar{x}}{\sigma} + a_2 \left( \frac{x - \bar{x}}{\sigma} \right)^2$$

The same example:

| $x$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $y$ | 1 | 2 | 1 | 2 | 3 |

Since $\bar{x} = 3$ and $\sigma = 1.5811$ we get the normal equations

$$A^T A \boldsymbol{a} = A^T \boldsymbol{y} \Longrightarrow \begin{bmatrix} 5 & 0 & 4 \\ 0 & 4 & 0 \\ 4 & 0 & 5.44 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 9 \\ 2.5298 \\ 12.6491 \end{bmatrix}$$

- $\mathrm{cond}_2(A^T A) = 7.6$
- Same polynomial, but even lower cond. number
- More stable and accurate solution for many problems
- There might be still some stability issues for large scale problems even with shifted and scaled ansatz.

# In Python ...

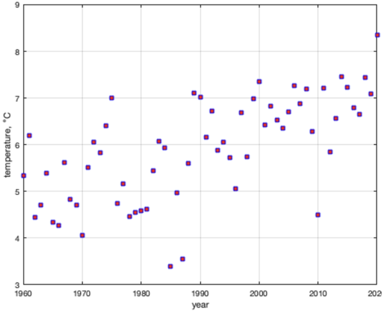
Previous example via normal equation:

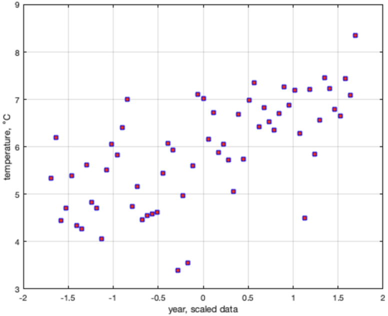
```
# Quadratic fitting with shift and scale
import numpy as np

x = np.array([1,2,3,4,5])
y = np.array([1,2,1,2,3])
mu, s = np.mean(x), np.std(x)
x = (x-mu)/s
A = np.ones((len(x),3))
A[:,1] = x; A[:,2] = x**2
a = np.linalg.solve(A.T@A, A.T@y)
print('a = ', a)
print("The model: y = %f+%f(x-%f)/%f+%f((x-%f)/%f)^2"
      % (a[0],a[1],mu,s,a[2],mu,s))
print('cond(ATA) = ', np.linalg.cond(A.T@A))
```

Output:

```
a =  [1.51428571 0.56568542 0.28571429]
The model: y = 1.51428+0.565685(x-3)/1.414214+0.285714((x-3)/1.414214)^2
cond(ATA) =  8.293712447937585
```

Data from Lab exercise 2 (temperature data):




**Original data**

Polynomial degree 1:

$\mathrm{cond}_2(A^TA) = 5 \times 10^{10}$

Polynomial degree 2:

$\mathrm{cond}_2(A^TA) = 5 \times 10^{21}$

**Centered and scaled data**

Polynomial degree 1:

$\mathrm{cond}_2(A^TA) = 1.02$

Polynomial degree 2:

$\mathrm{cond}_2(A^TA) = 7.67$

# Least squares fitting

Data from Lab exercise 2 (temperature data): What happens in Python when running the case Polynomial degree 2 when $\text{cond}_2(A^TA) = 5 \times 10^{21}$?

- If `numpy.linalg.solve` is used as equation solver, nothing happens.
- If `scipy.linalg.solve` is used, get a warning:

```
v = scipy.linalg.solve(ATA,ATy)

LinAlgWarning: Ill-conditioned matrix (rcond=3.1176e-22):
        result may not be accurate.
```

- The difference between the `Polynomial.fit` solution and normal equation solution is approximately $10^{-7}$ (the norm of the difference). Interpretation: Loose accuracy, roughly 9 decimal places (remember the best we can expect is $10^{-16}$)