

# Linear Algebra for Machine Learning in Python

---

Dr. Moritz Wolter

August 12, 2022

High Performance Computing and Analytics Lab

Introduction

Essential operations

Linear curve fitting

Regularization

# Introduction

---

*Même le feu est régi par les nombres.*

Fourier<sup>1</sup> studied the transmission of heat using tools that would later be called an eigenvector-basis. Why would he say something like this?

---

<sup>1</sup>Jean Baptiste Joseph Fourier (1768-1830)

$\mathbf{A} \in \mathbb{R}^{m,n}$  is a real-valued Matrix with  $m$  rows and  $n$  columns.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, a_{ij} \in \mathbb{R}. \quad (1)$$

# Essential operations

---

# Addition

To matrices  $\mathbf{A} \in \mathbf{R}^{m,n}$  and  $\mathbf{B} \in \mathbf{R}^{m,n}$  can be added by adding their elements.

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \dots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \dots & a_{mn} + b_{mn} \end{pmatrix} \quad (2)$$

# Multiplication

Multiply  $\mathbf{A} \in \mathbb{R}^{m,n}$  by  $\mathbf{B} \in \mathbb{R}^{n,p}$  produces  $\mathbf{C} \in \mathbb{R}^{m,p}$ ,

$$\mathbf{AB} = \mathbf{C}. \quad (3)$$

To compute  $\mathbf{C}$  the elements in the rows of  $\mathbf{A}$  are multiplied with the column elements of  $\mathbf{B}$  and the products added,

$$c_{ik} = \sum_{j=1}^m a_{ij} \cdot b_{jk}. \quad (4)$$



## Linear Algebra for Machine Learning in Python

## └ Essential operations

## └ Multiplication

Multiply  $\mathbf{A} \in \mathbb{R}^{m,n}$  by  $\mathbf{B} \in \mathbb{R}^{n,p}$  produces  $\mathbf{C} \in \mathbb{R}^{m,p}$ ,

$$\mathbf{AB} = \mathbf{C}. \quad (3)$$

To compute  $\mathbf{C}$  the elements in the rows of  $\mathbf{A}$  are multiplied with the column elements of  $\mathbf{B}$  and the products added,

$$c_{ik} = \sum_{j=1}^n a_{ij} \cdot b_{jk}. \quad (4)$$

Define on the board:

- Dot product  $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$  for two vectors  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ .
- Row times column view [Str+09]:

# The identity matrix

$$\mathbf{I} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \quad (5)$$

## Linear Algebra for Machine Learning in Python

└ Essential operations

└ The identity matrix

$$\mathbf{I} = \begin{pmatrix} 1 & & \\ & 1 & \\ & & \ddots \\ & & & 1 \end{pmatrix} \quad (5)$$

Demonstrate multiplication with the inverse by hand. TODO

# Matrix inverse

The inverse Matrix  $\mathbf{A}^{-1}$  undoes the effects of  $\mathbf{A}$ , or in mathematical notation,

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}. \quad (6)$$

The process of computing the inverse is called gaussian elimination.

## Linear Algebra for Machine Learning in Python

## └ Essential operations

## └ Matrix inverse

The inverse Matrix  $\mathbf{A}^{-1}$  undoes the effects of  $\mathbf{A}$ , or in mathematical notation,

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

(6)

The process of computing the inverse is called gaussian elimination.

Example on the board:

$$\mathbf{A} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \rightsquigarrow \left( \begin{array}{cc|cc} 2 & 0 & 1 & 0 \\ 1 & 3 & 0 & 1 \end{array} \right) \rightsquigarrow \left( \begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 1 & 3 & 0 & 1 \end{array} \right) \quad (7)$$

$$\rightsquigarrow \left( \begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 0 & 3 & -\frac{1}{2} & 1 \end{array} \right) \rightsquigarrow \left( \begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 0 & 1 & -\frac{1}{6} & \frac{1}{3} \end{array} \right) \quad (8)$$

Test the result:

$$\begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{3} \end{pmatrix} = \begin{pmatrix} 2 \cdot \frac{1}{2} + 0 \cdot -\frac{1}{6} & 2 \cdot 0 + 0 \cdot \frac{1}{3} \\ 1 \cdot \frac{1}{2} + 3 \cdot -\frac{1}{6} & 0 \cdot 0 + 3 \cdot \frac{1}{3} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (9)$$

# The Transpose

The transpose operation flips matrices along the diagonal, for example in  $\mathbb{R}^2$ ,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \quad (10)$$

# Motivation of the determinant

TODO

## Computing determinants in two or three dimensions

The two dimensional case:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} \quad (11)$$

(12)

Computing the determinant of a three dimensional matrix.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{22} & a_{23} \end{vmatrix} \quad (13)$$



# Linear Algebra for Machine Learning in Python

## └ Essential operations

## └ Computing determinants in two or three dimensions

The two dimensional case:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} \quad (11)$$

(12)

Computing the determinant of a three dimensional matrix:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \quad (13)$$

Draw the sign pattern on the board:

$$\begin{vmatrix} + & - & + & \dots \\ - & + & - & \dots \\ + & - & + & \dots \\ \vdots & \vdots & \vdots & \ddots \end{vmatrix} \quad (14)$$

The determinant can be expanded along any column as long as the sign pattern is respected.

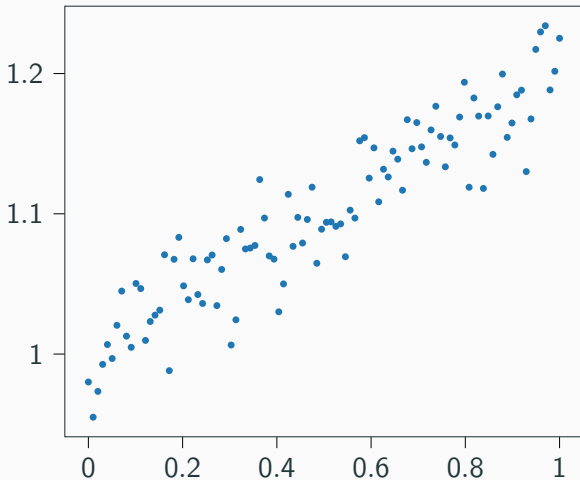
# Determinants in n-dimensions

$$\begin{vmatrix} a_{11} & a_{21} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & \dots & a_{2n} \\ \vdots & & \vdots \\ a_{m2} & \dots & a_{mn} \end{vmatrix} + a_{21} \begin{vmatrix} a_{21} & \dots & a_{2n} \\ \vdots & & \vdots \\ a_{m2} & \dots & a_{mn} \end{vmatrix} \\ - a_{m1} \begin{vmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & & \vdots \end{vmatrix}$$

# Linear curve fitting

---

## What is the best line connecting measurements?



## Problem Formulation

A line has the form  $cx + d$ , with  $c, x, d \in \mathbb{R}$ . In matrix language we could ask for every point to be on the line,

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}. \quad (15)$$

We can treat polynomials as vectors, too! The coordinates populate the matrix rows in  $\mathbf{A} \in \mathbb{R}^{n_p \times 2}$ , and the coefficients appear in  $\mathbf{x} \in \mathbb{R}^2$ , with the points we would like to model in  $\mathbf{b} \in \mathbb{R}^{n_p}$ . The problem now appears in matrix form and can be solved using linear algebra!

## The Pseudoinverse [Str+09; DFO20]

The inverse we saw earlier only exists for square that is  $n$  by  $n$  matrices. Nonsquare  $\mathbf{A}$  such as the one we just saw, require the pseudoinverse,

$$\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T. \quad (16)$$

Sometimes solving  $\mathbf{Ax} + \mathbf{b} = 0$  is impossible, the pseudoinverse considers,

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (17)$$

$$(18)$$

instead.  $\mathbf{A}^\dagger \mathbf{b} = \mathbf{x}$  yields the solution.

## Linear Algebra for Machine Learning in Python

## └ Linear curve fitting

## └ The Pseudoinverse [Str+09; DFO20]

The inverse we saw earlier only exists for square that is  $n$  by  $n$  matrices. Nonsquare  $\mathbf{A}$  such as the one we just saw, require the pseudoinverse,

$$\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T. \quad (16)$$

Sometimes solving  $\mathbf{Ax} + \mathbf{b} = 0$  is impossible, the pseudoinverse considers,

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (17)$$

(18)

instead  $\mathbf{A}^\dagger \mathbf{b} = \mathbf{x}$  yields the solution.

Sometimes solving  $\mathbf{Ax} + \mathbf{b} = 0$  is impossible. One the board, derive:

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (19)$$

$$(20)$$

At the optimum we expect,

$$0 = \nabla_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (21)$$

$$= \nabla_x \frac{1}{2} (\mathbf{Ax} - \mathbf{b})^T (\mathbf{Ax} - \mathbf{b}) \quad (22)$$

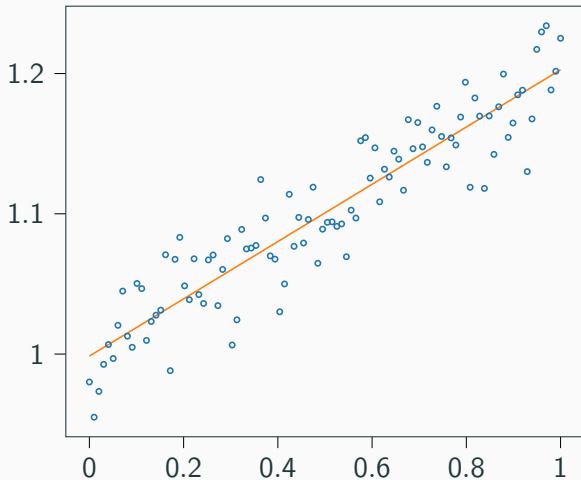
$$= (\mathbf{Ax} - \mathbf{b}) \mathbf{A}^T \quad (23)$$

$$= \mathbf{A}^T \mathbf{Ax} - \mathbf{A}^T \mathbf{b} \quad (24)$$

$$\mathbf{A}^T \mathbf{b} = \mathbf{A}^T \mathbf{Ax} \quad (25)$$

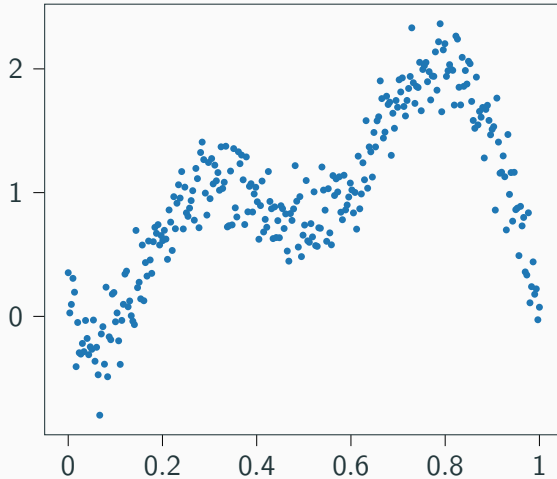
$$(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} = \mathbf{x} \quad (26)$$

# Linear regression





## What about harder problems?



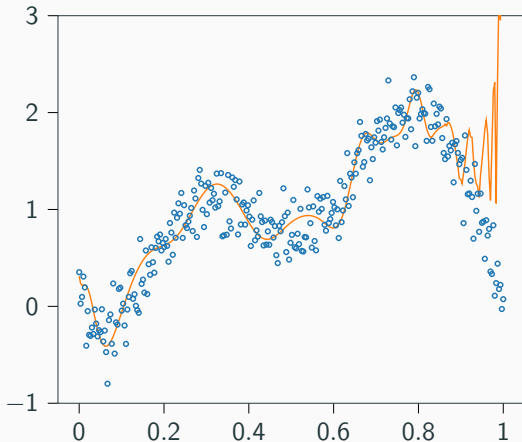
## Fitting higher order polynomials

$$\underbrace{\begin{pmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^m \\ 1 & x_2^1 & x_2^2 & \dots & x_2^m \\ 1 & x_3^1 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^1 & x_n^2 & \dots & x_n^m \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}}_{\mathbf{b}}. \quad (27)$$

As we saw for the linear regression  $\mathbf{A}^\dagger \mathbf{b} = \mathbf{x}$  gives us the coefficients.

# Overfitting

Below the solution for a polynomial of 7th degree, that is  $m = 7$ .



The noise took over! What now?

# Regularization

---

- Is there a way to fix the previous example?
- To do so we start from a rather peculiar observation.

## Eigenvalues and Eigen-Vectors

Multiply matrix **A** with vectors **x**<sub>1</sub> and **x**<sub>2</sub>,

$$\mathbf{A} = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}, \mathbf{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad (28)$$

we observe

$$\mathbf{Ax}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{Ax}_2 = \begin{pmatrix} 8 \\ 2 \end{pmatrix} \quad (29)$$

Vector **x**<sub>1</sub> has not changed! Vector **x**<sub>2</sub> was multiplied by two. In other words,

$$\mathbf{Ax}_1 = 1\mathbf{x}_1, \mathbf{Ax}_2 = 2\mathbf{x}_2 \quad (30)$$

# Eigenvalues and Eigen-vectors

Eigenvectors turn multiplication with a matrix into multiplication with a number,

$$\mathbf{Ax} = \lambda \mathbf{x}. \quad (31)$$

# Eigenvalue-Decomposition [Str+09]

Eigenvalues let us look into the heart of a square system-matrix

$\mathbf{A} \in \mathbb{R}^{n,n}$ .

$$\mathbf{A} = \mathbf{S} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \mathbf{S}^{-1} = \mathbf{S} \mathbf{\Lambda} \mathbf{S}^{-1} \quad (32)$$

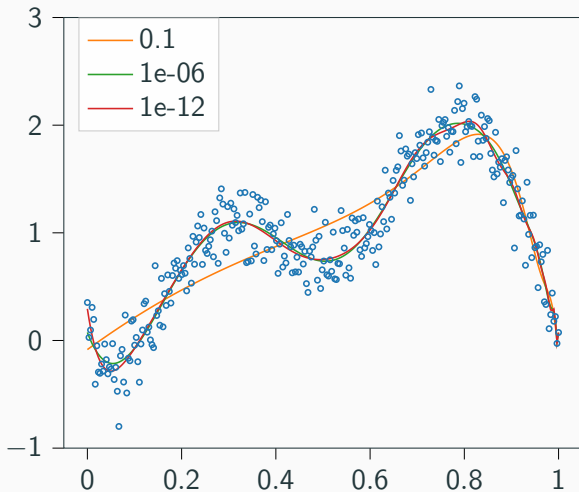


# Computing the Decomposition

Dealing with matrices which aren't square:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (33)$$

## Regularized solution



# Conclusion

- True scientists know what linear can do for them!

## References

---

- [DFO20] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.
- [Str+09] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*. Vol. 4. Wellesley-Cambridge Press Wellesley, MA, 2009.