

Linear Algebra for Machine Learning in Python

Moritz Wolter

September 14, 2022

High-Performance Computing and Analytics Lab, University of Bonn

Introduction

Essential operations

Linear curve fitting

Regularization

Introduction

Même le feu est régi par les nombres.

Fourier¹ studied the transmission of heat using tools that would later be called an eigenvector-basis. Why would he say something like this?

¹Jean 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 $\mathbf{C} \in \mathbb{R}^{m,p}$.

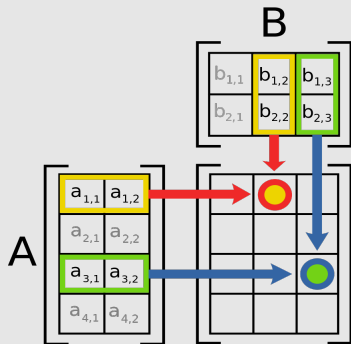
$$\mathbf{AB} = \mathbf{C}. \quad (3)$$

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 $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$ 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

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

Demonstrate multiplication with the inverse by hand.

$$\begin{pmatrix} -1 & 0 & 0 \\ 1 & -1 & 1 \\ 1 & 0 & -1 \end{pmatrix} \begin{pmatrix} -1 & -0 & -0 \\ -2 & -1 & -1 \\ -1 & -0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (6)$$

Matrix inverse

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

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}. \quad (7)$$

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,

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

(7)

The process of computing the inverse is called Gaussian elimination.

Example on the board:

$$\mathbf{A} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \rightsquigarrow \left(\begin{array}{cc|cc} 2 & 0 & 1 & 0 \\ 1 & 3 & 0 & 1 \end{array} \right) \rightsquigarrow \left(\begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 1 & 3 & 0 & 1 \end{array} \right) \quad (8)$$

$$\rightsquigarrow \left(\begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 0 & 3 & -\frac{1}{2} & 1 \end{array} \right) \rightsquigarrow \left(\begin{array}{cc|cc} 1 & 0 & \frac{1}{2} & 0 \\ 0 & 1 & -\frac{1}{6} & \frac{1}{3} \end{array} \right) \quad (9)$$

Test the result:

$$\begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} \frac{1}{2} & 0 \\ -\frac{1}{6} & \frac{1}{3} \end{pmatrix} = \begin{pmatrix} 2 \cdot \frac{1}{2} + 0 \cdot -\frac{1}{6} & 2 \cdot 0 + 0 \cdot \frac{1}{3} \\ 1 \cdot \frac{1}{2} + 3 \cdot -\frac{1}{6} & 0 \cdot 0 + 3 \cdot \frac{1}{3} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad (10)$$

The Transpose

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

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \quad (11)$$

Motivation of the determinant

- The determinant contains lots of information about a matrix in a single number.
- When a Matrix has a zero determinant, its inverse does not exist.
- We require determinants to find eigenvalues by hand.

Computing determinants in two or three dimensions

The two-dimensional case:

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

(13)

Computing the determinant of a three-dimensional matrix.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \quad (14)$$

Linear Algebra for Machine Learning in Python

Essential operations

Computing determinants in two or three dimensions

The two-dimensional case:

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

(13)

Computing the determinant of a three-dimensional matrix.

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \cdot \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \cdot \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \quad (14)$$

Example computation on the board:

$$\begin{vmatrix} -1 & 0 & 0 \\ 1 & -1 & 1 \\ 1 & 0 & -1 \end{vmatrix} = -1 \cdot \begin{vmatrix} -1 & 1 \\ 0 & -1 \end{vmatrix} - 1 \cdot \begin{vmatrix} 0 & 0 \\ 0 & -1 \end{vmatrix} + 1 \cdot \begin{vmatrix} 0 & 0 \\ -1 & 1 \end{vmatrix} \quad (15)$$

$$= (-1) \cdot ((-1) \cdot (-1) - 0 \cdot 1) - \quad (16)$$

$$(0 \cdot (-1) - 0 \cdot 0) + 0 \cdot 1 - (-1) \cdot 0 \quad (17)$$

$$= -1 \quad (18)$$

Determinants in n-dimensions

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

Linear Algebra for Machine Learning in Python

└ Essential operations

└ Determinants in n-dimensions

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n2} & \dots & a_{nn} \end{vmatrix} + a_{12} \begin{vmatrix} a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} + \dots + a_{1n} \begin{vmatrix} a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{vmatrix}$$

Draw the sign pattern on the board:

$$\begin{vmatrix} + & - & + & \dots \\ - & + & - & \dots \\ + & - & + & \dots \\ \vdots & \vdots & \vdots & \ddots \end{vmatrix} \quad (19)$$

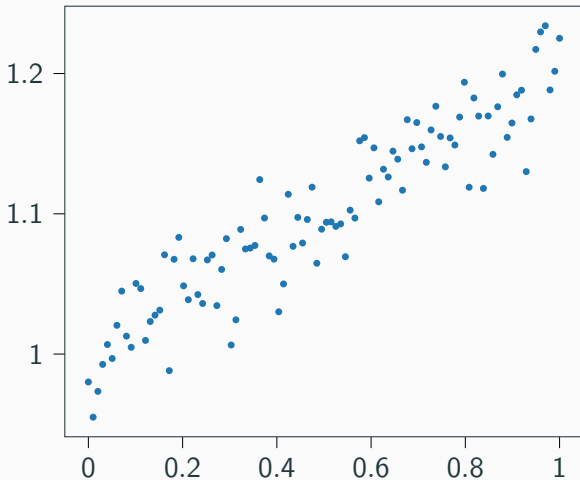
The determinant can be expanded along any column as long as the sign pattern is respected.

Summary

- We saw some of the most important operations in linear algebra.
- Let's use these to do something useful next.

Linear curve fitting

What is the best line connecting measurements?



Problem Formulation

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

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}. \quad (20)$$

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 exists for square or n by n matrices. Nonsquare \mathbf{A} such as the one we just saw, require the pseudoinverse,

$$\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T. \quad (21)$$

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

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (22)$$

$$(23)$$

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 exists for square or n by n matrices. Nonsquare \mathbf{A} such as the one we just saw, require the pseudoinverse,

$$\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T. \quad (21)$$

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

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (22)$$

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

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

$$\min_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (24)$$

$$(25)$$

At the optimum we expect,

$$0 = \nabla_x \frac{1}{2} |\mathbf{Ax} - \mathbf{b}|^2 \quad (26)$$

$$= \nabla_x \frac{1}{2} (\mathbf{Ax} - \mathbf{b})^T (\mathbf{Ax} - \mathbf{b}) \quad (27)$$

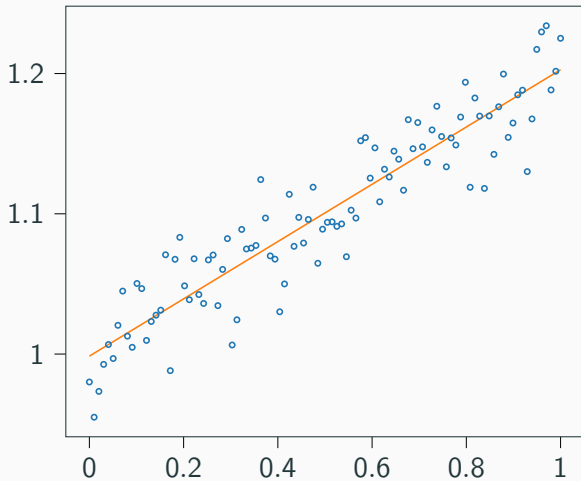
$$= (\mathbf{Ax} - \mathbf{b}) \mathbf{A}^T \quad (28)$$

$$= \mathbf{A}^T \mathbf{Ax} - \mathbf{A}^T \mathbf{b} \quad (29)$$

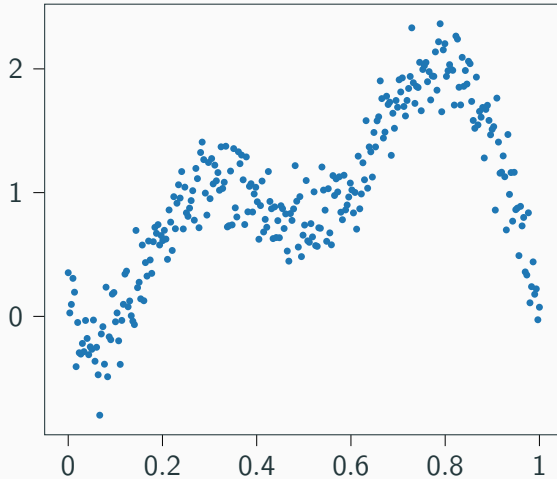
$$\mathbf{A}^T \mathbf{b} = \mathbf{A}^T \mathbf{Ax} \quad (30)$$

$$(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} = \mathbf{x} \quad (31)$$

Linear regression



What about harder problems?



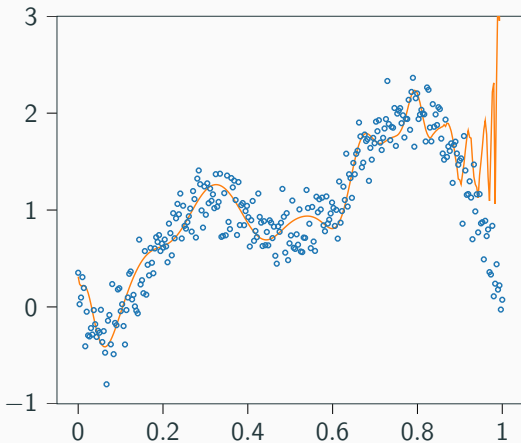
Fitting higher order polynomials

$$\underbrace{\begin{pmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^m \\ 1 & x_2^1 & x_2^2 & \dots & x_2^m \\ 1 & x_3^1 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^1 & x_n^2 & \dots & x_n^m \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}}_{\mathbf{x}} = \underbrace{\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}}_{\mathbf{b}}. \quad (32)$$

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

Overfitting

The figure below depicts the solution for a polynomial of 7th degree, that is $m = 7$.



Summary

- We saw how linear algebra lets us fit polynomials to curves.
- For the 7th degree polynomial 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**₁ and **x**₂,

$$\mathbf{A} = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix}, \mathbf{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{x}_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad (33)$$

we observe

$$\mathbf{Ax}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \mathbf{Ax}_2 = \begin{pmatrix} 8 \\ 2 \end{pmatrix} \quad (34)$$

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

$$\mathbf{Ax}_1 = 1\mathbf{x}_1, \mathbf{Ax}_2 = 2\mathbf{x}_2 \quad (35)$$

Eigenvalues and Eigenvectors

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

$$\mathbf{Ax} = \lambda \mathbf{x}. \quad (36)$$

Subtracting $\lambda \mathbf{x}$ leads to,

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = 0 \quad (37)$$

$$(38)$$

The interesting solutions are those where $\mathbf{x} \neq 0$, which means

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0 \quad (39)$$

Linear Algebra for Machine Learning in Python

└ Regularization

└ Eigenvalues and Eigenvectors

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

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad (36)$$

Subtracting $\lambda\mathbf{x}$ leads to,

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0 \quad (37)$$

$$(38)$$

The interesting solutions are those where $\mathbf{x} \neq 0$, which means

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \quad (39)$$

On the board, compute the eigenvalues and vectors for the initial example.

$$\mathbf{A} = \begin{pmatrix} 1 & 4 \\ 0 & 2 \end{pmatrix} \rightarrow \begin{vmatrix} 1 - \lambda & 4 \\ 0 & 2 - \lambda \end{vmatrix} = (1 - \lambda) * (2 - \lambda) - 0 * 4 = 0 \quad (40)$$

$$\rightarrow \lambda_1 = 1, \lambda_2 = 2. \quad (41)$$

$$\begin{pmatrix} 1 - 1 & 4 \\ 0 & 2 - 1 \end{pmatrix} = \begin{pmatrix} 0 & 4 \\ 0 & 1 \end{pmatrix} \mathbf{x}_1 = 0 \rightarrow \mathbf{x}_1 = \begin{pmatrix} p \\ 0 \end{pmatrix} \text{ for } p \in \mathbb{R} \quad (42)$$

$$\begin{pmatrix} 1 - 2 & 4 \\ 0 & 2 - 2 \end{pmatrix} = \begin{pmatrix} -1 & 4 \\ 0 & 0 \end{pmatrix} \mathbf{x}_1 = 0 \rightarrow \mathbf{x}_2 = \begin{pmatrix} q \\ \frac{1}{4}q \end{pmatrix} \text{ for } q \in \mathbb{R} \quad (43)$$

Determinant not useful numerically, software packages use QR-Method.

Eigenvalues let us look into the heart of a square system-matrix $\mathbf{A} \in \mathbb{R}^{n,n}$.

$$\mathbf{A} = \mathbf{S} \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{pmatrix} \mathbf{S}^{-1} = \mathbf{S} \mathbf{\Lambda} \mathbf{S}^{-1}, \quad (44)$$

with $\mathbf{S} \in \mathbb{R}^{n,n}$ and $\mathbf{\Lambda} \in \mathbb{C}^{n,n}$.

Singular-Value-Decomposition [Str+09]

What about a non-square matrix $\mathbf{A} \in \mathbb{R}^{n,m}$? Idea:

$$\mathbf{A}^T \mathbf{A} = \mathbf{V} \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{pmatrix} \mathbf{V}^{-1}, \mathbf{A} \mathbf{A}^T = \mathbf{U} \begin{pmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_n^2 \end{pmatrix} \mathbf{U}^{-1}. \quad (45)$$

Using the eigenvectors of the $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A} \mathbf{A}^T$ we construct,

$$\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T, \quad (46)$$

with $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{U} \in \mathbb{R}^{m,m}$, $\mathbf{\Sigma} \in \mathbb{R}^{m,n}$ and $\mathbf{V} \in \mathbb{R}^{n,n}$.

Singular values and matrix inversion [GK65]

$$\mathbf{A}^\dagger = \mathbf{V}\Sigma^\dagger\mathbf{U}^T = \mathbf{V} \left(\begin{array}{ccc} \sigma_1^{-1} & & \\ & \ddots & \\ & & \sigma_m^{-1} \\ \hline & 0 & \end{array} \right) \mathbf{U}^T \quad (47)$$

Regularization via Singular Value Filtering

Originally we had a problem computing $\mathbf{A}^\dagger \mathbf{b} = \mathbf{x}$. To solve it, we compute,

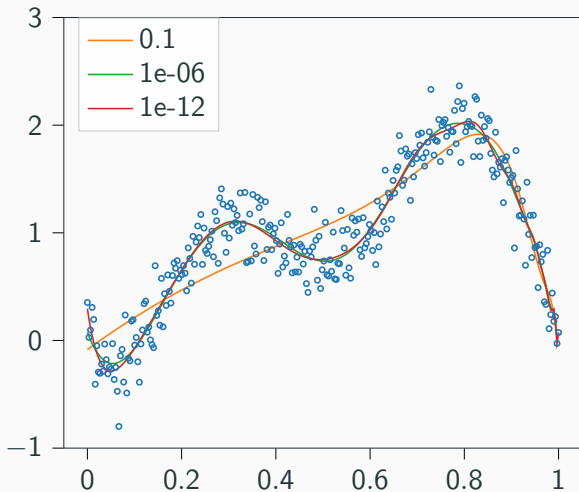
$$\mathbf{x}_{reg} = \sum_{i=1}^n f_i \frac{\mathbf{u}_i^T \mathbf{b}}{\sigma_i} \mathbf{v}_i \quad (48)$$

The filter factors are computed using $f_i = \sigma_i^2 / (\sigma_i^2 + \epsilon)$. Singular values $\sigma_i < \epsilon$ are filtered. Expressing equation 48 using matrix notation:

$$\mathbf{x}_{reg} = \mathbf{V} \mathbf{F} \begin{pmatrix} \sigma_1^{-1} & & & \\ & \ddots & & \\ & & \sigma_m^{-1} & \\ \hline & & & 0 \end{pmatrix} \mathbf{U}^T \mathbf{b}_{noise} \quad (49)$$

with $\mathbf{A} \in \mathbb{R}^{m,n}$, $\mathbf{U} \in \mathbb{R}^{m,m}$, $\mathbf{V} \in \mathbb{R}^{n,n}$, $\mathbf{F} \in \mathbb{R}^{m,m}$, $\Sigma^\dagger \in \mathbb{R}^{n,m}$ and $\mathbf{b} \in \mathbb{R}^{n,1}$.

Regularized solution



Conclusion

- True scientists know what linear can do for them!
- Think about matrix shapes. If you are solving a problem, rule out all formulations where the shapes don't work.
- Regularization using the SVD is also known as Tikhonov regularization.

References

- [DFO20] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.
- [GK65] Gene Golub and William Kahan. “Calculating the singular values and pseudo-inverse of a matrix.” In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.
- [Str+09] Gilbert Strang, Gilbert Strang, Gilbert Strang, and Gilbert Strang. *Introduction to linear algebra*. Vol. 4. Wellesley-Cambridge Press Wellesley, MA, 2009.