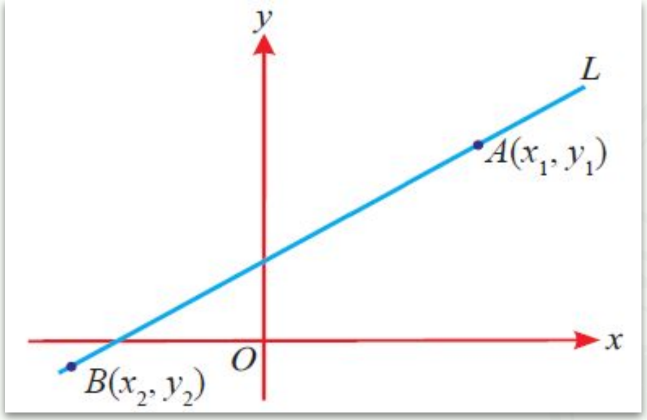
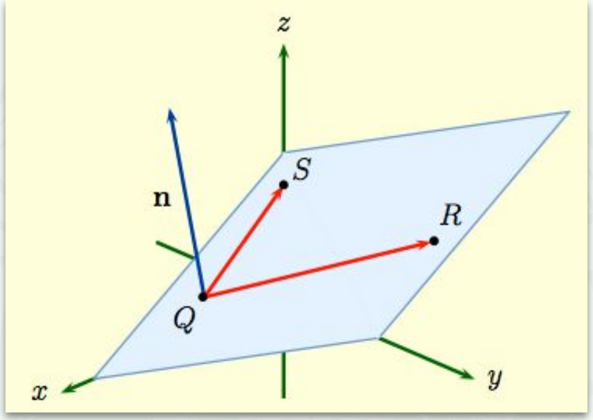
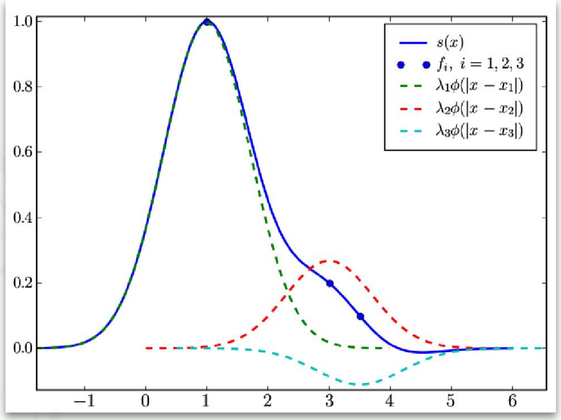
Linear Algebra II

Michael S. Murillo
Computational Mathematics, Science and Engineering
Michigan State University

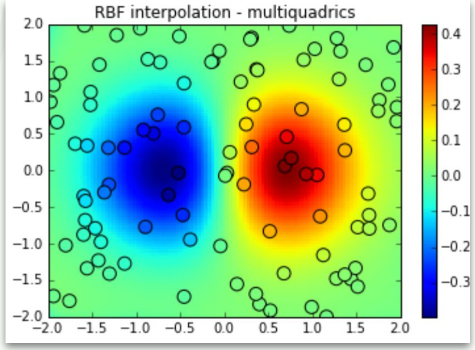
Review of Linear Regression







$$\mathbf{y} = K\mathbf{w} \qquad \mathbf{w} = K^{-1}\mathbf{y}$$



Multiple Linear Regression

In data science, we usually have many input features (independent variables):

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots$$

In general, every additional feature you add gives you more predictive power. There are two caveats:

1. The feature x_n is irrelevant. In this case, $w_n=0$; little harm done.
2. If two features are linearly dependent, you need to drop one of them. It adds no new information and it will likely cause mathematical problems.

Multiple Linear Regression: Iris

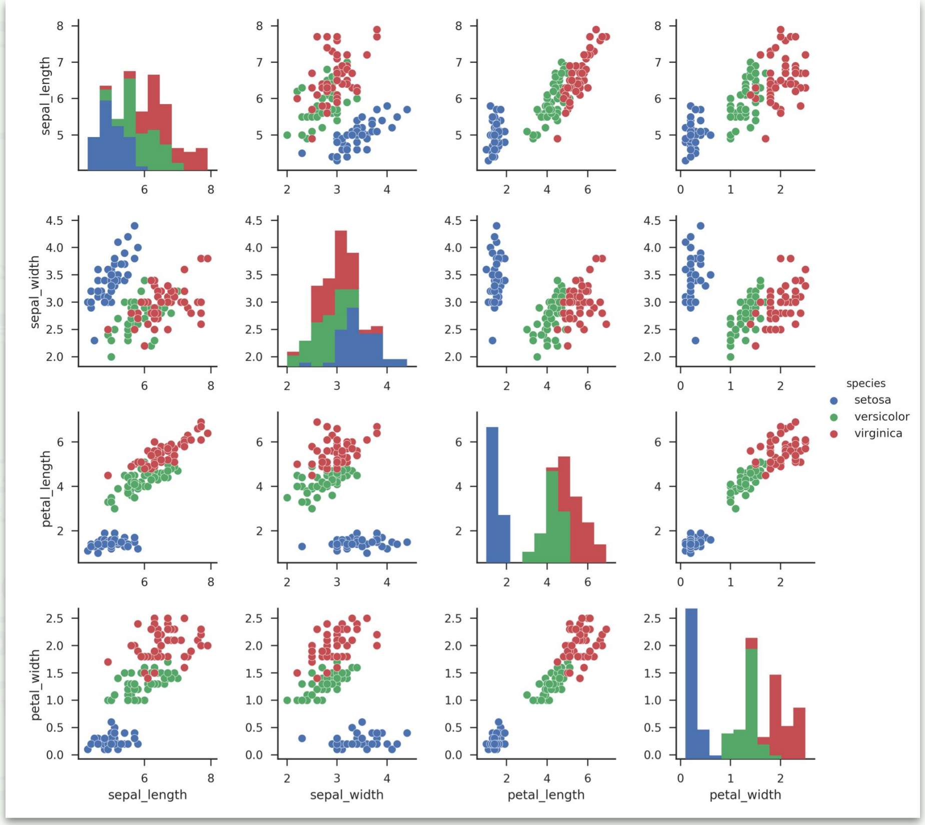
Suppose we want to predict petal width for versicolor (green).

petal_width = w_0

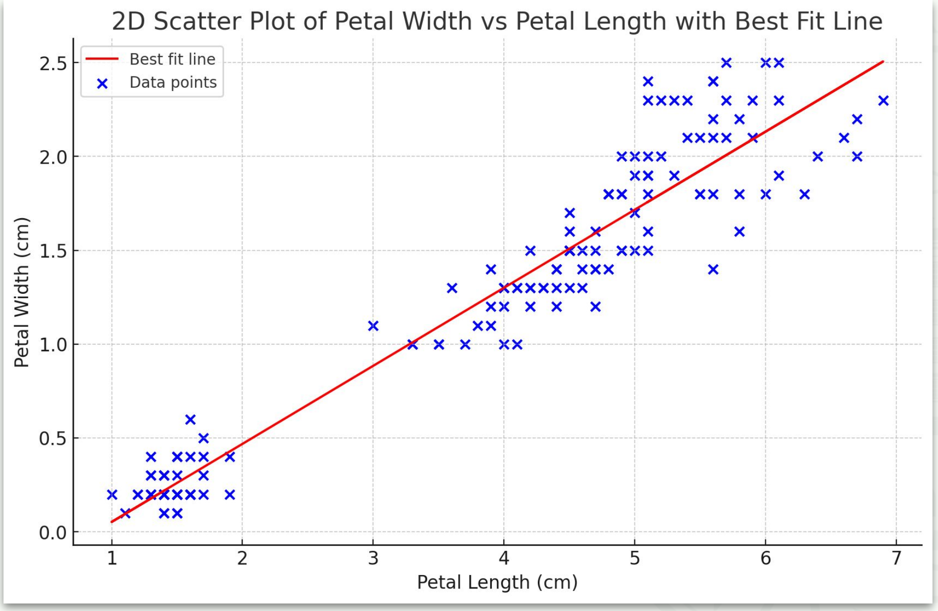
petal_width = $w_0 + w_1$ petal_length

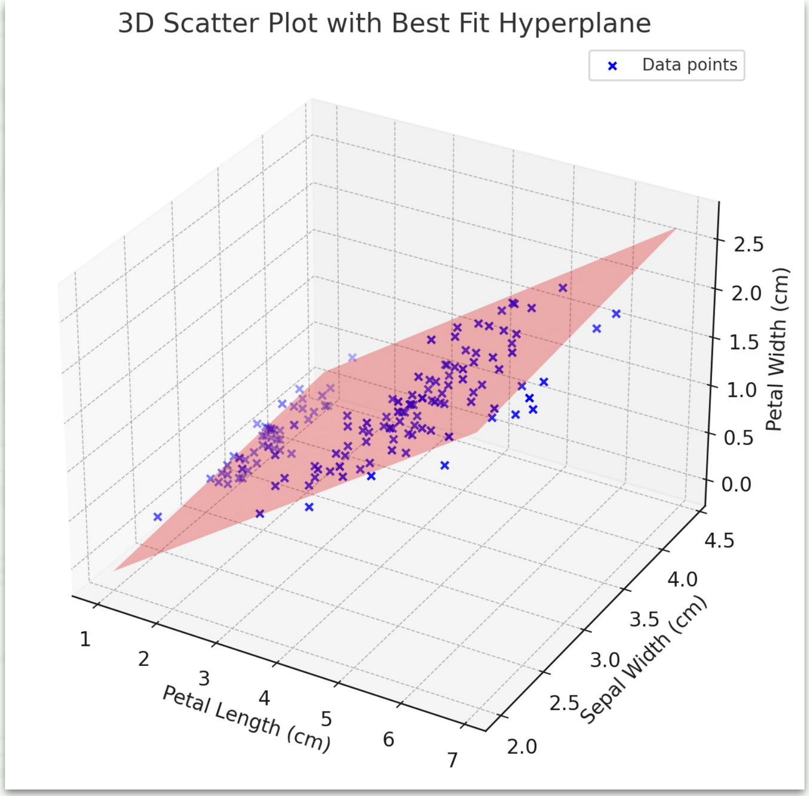
petal_width = $w_0 + w_1$ petal_length + w_2 sepal_width

Note that petal_width versus sepal_length is fairly flat.



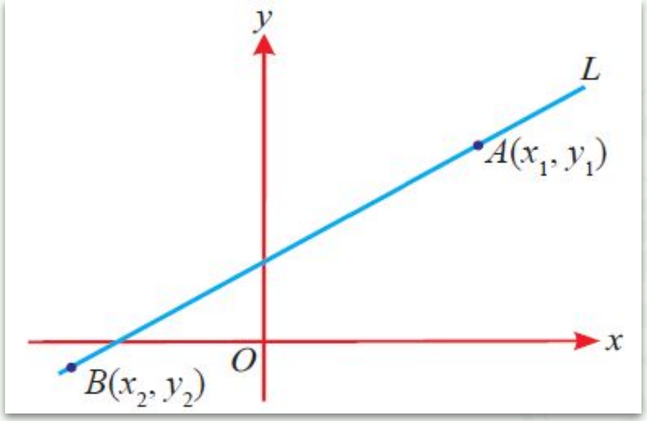
Multiple Linear Regression with Iris





Linear Algebra Details

Let’s find the line using linear algebra:

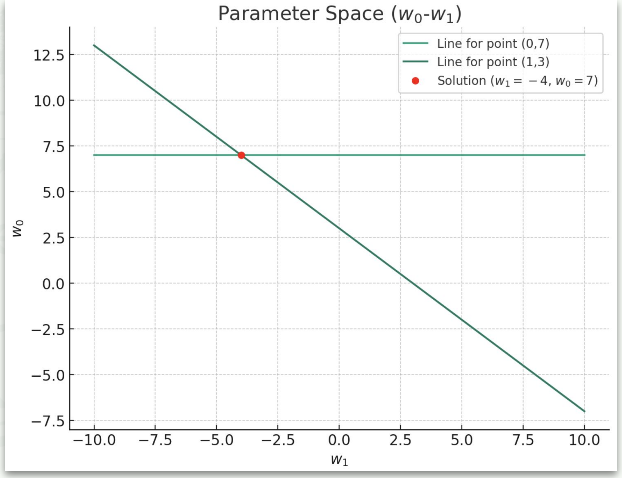


$$y = w_0 + w_1 x$$

Suppose we have these two points:

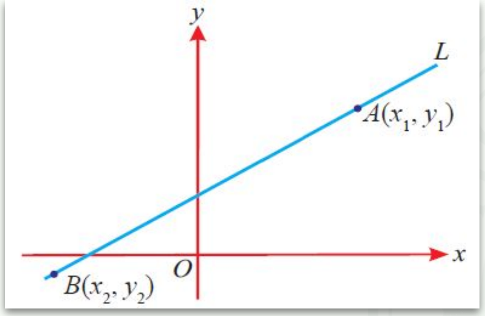
$$(0,7),(1,3)$$

We can substitute these points into two of our model equations to get two **relations between the weights**.



We don’t want to search for intersections of lines!

Let’s find the line using linear algebra:



$$y = w_0 + w_1 x$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\begin{bmatrix} 7 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 7 \\ 3 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 3 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 7 \\ -4 \end{bmatrix}$$

$$y = 7 - 4x$$

We have our final model!

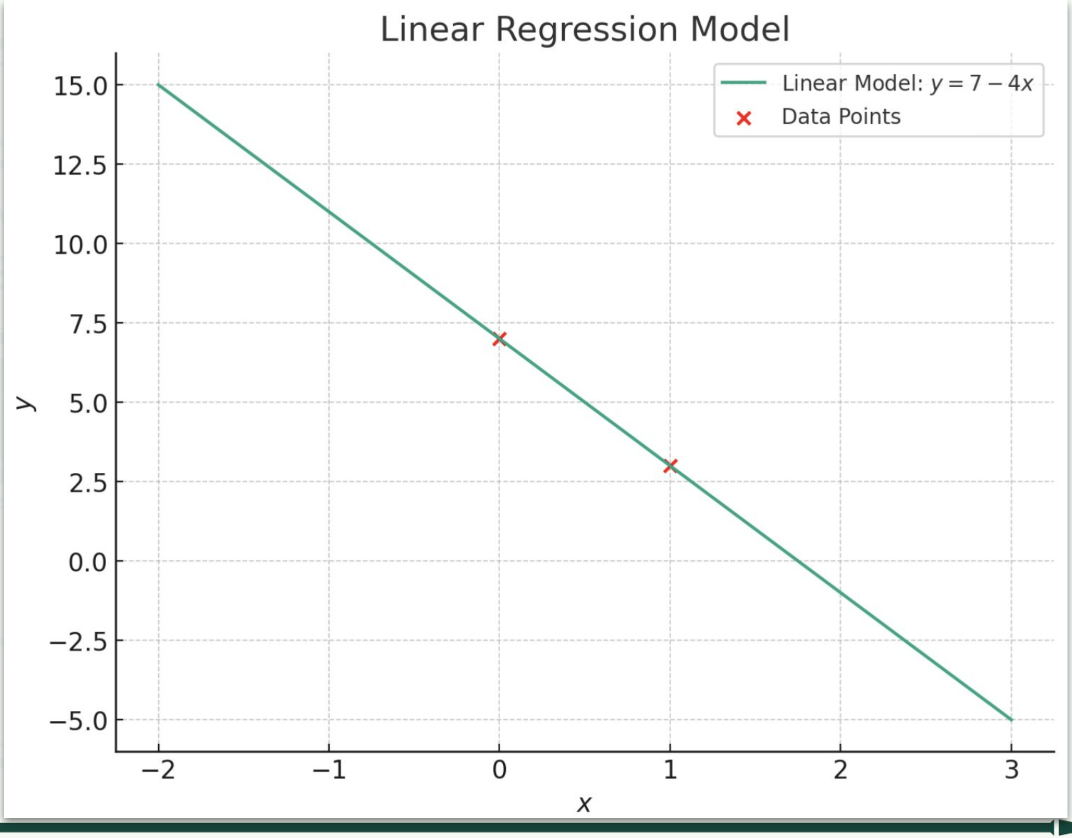
$$\begin{bmatrix}7\\3\end{bmatrix} = \begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}\begin{bmatrix}w_0\\w_1\end{bmatrix}$$

$$\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}^{-1}\begin{bmatrix}7\\3\end{bmatrix} = \begin{bmatrix}w_0\\w_1\end{bmatrix}$$

$$\begin{bmatrix}1 & 0\\-1 & 1\end{bmatrix}\begin{bmatrix}7\\3\end{bmatrix} = \begin{bmatrix}w_0\\w_1\end{bmatrix}$$

$$\begin{bmatrix}w_0\\w_1\end{bmatrix} = \begin{bmatrix}7\\-4\end{bmatrix}$$

$$y = 7 - 4x$$



Key Step: Finding the Inverse

The inverse is defined through:

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

For a 2x2 matrix:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

$$\det(A) = ad - bc$$

By hand:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} X^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$a = 1$$

$$b = 0$$

$$a + c = 0$$

$$b + d = 1$$

Note that the determinant must not be zero!

When is the determinant zero?

1. there is no solution
2. there are infinitely many solutions

Understanding Matrix Rank in Linear Algebra

Rank of a Matrix: The rank of a matrix is the dimension of the vector space spanned by its rows (row rank) or columns (column rank). It is equal to the maximum number of linearly independent row vectors or column vectors in the matrix.

- **Rank Deficient**: A matrix is rank deficient if its rank is less than its number of rows or columns. This implies that some rows or columns are linear combinations of others.

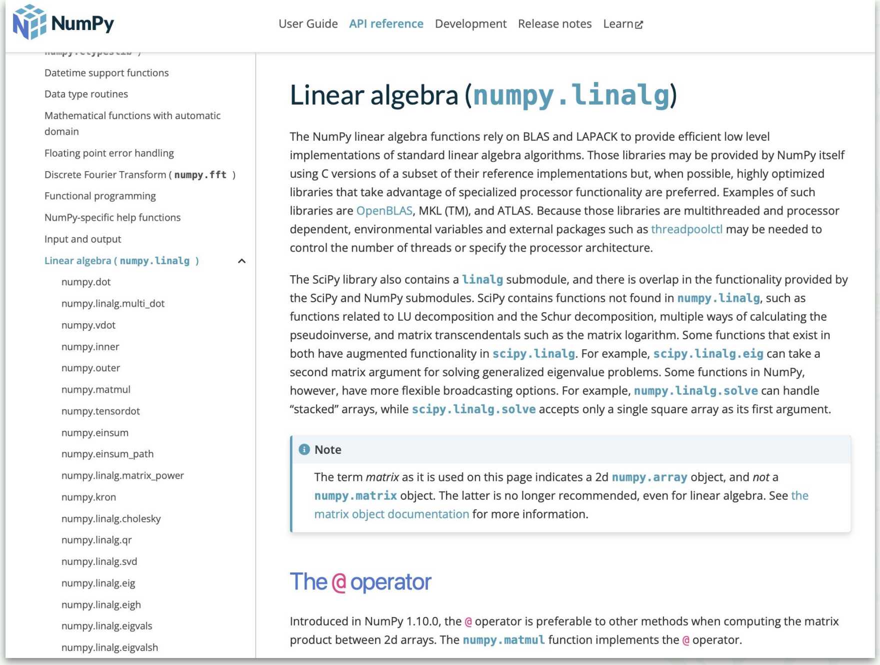
  a. **Geometric Interpretation**: Rank deficiency means that the space spanned by the row or column vectors of the matrix is "flattened" into a lower dimension than the matrix itself, indicating dependency among the vectors.

- **Inconsistent System**: A system of linear equations is inconsistent if there are no solutions that satisfy all equations simultaneously.

  a. **Geometric Interpretation**: Geometrically, an inconsistent system represents a set of planes (or lines in two dimensions) that do not intersect at a common point. Instead, at least two of them are parallel, which means there is no point that lies on all planes (or lines).

$$\begin{bmatrix} 1 & 8 & 13 & 12 \\ 14 & 11 & 2 & 7 \\ 4 & 5 & 16 & 9 \\ 15 & 10 & 3 & 6 \end{bmatrix}$$

Python Libraries



- Datetime support functions
- Data type routines
- Mathematical functions with automatic domain
- Floating point error handling
- Discrete Fourier Transform (`numpy.fft`)
- Functional programming
- NumPy-specific help functions
- Input and output
- Linear algebra (`numpy.linalg`)
  - numpy.dot
  - numpy.linalg.multi_dot
  - numpy.vdot
  - numpy.inner
  - numpy.outer
  - numpy.matmul
  - numpy.tensordot
  - numpy.einsum
  - numpy.einsum_path
  - numpy.linalg.matrix_power
  - numpy.kron
  - numpy.linalg.cholesky
  - numpy.linalg.qr
  - numpy.linalg.svd
  - numpy.linalg.eig
  - numpy.linalg.eigh
  - numpy.linalg.eigvals
  - numpy.linalg.eigvalsh

Linear algebra (`numpy.linalg`)

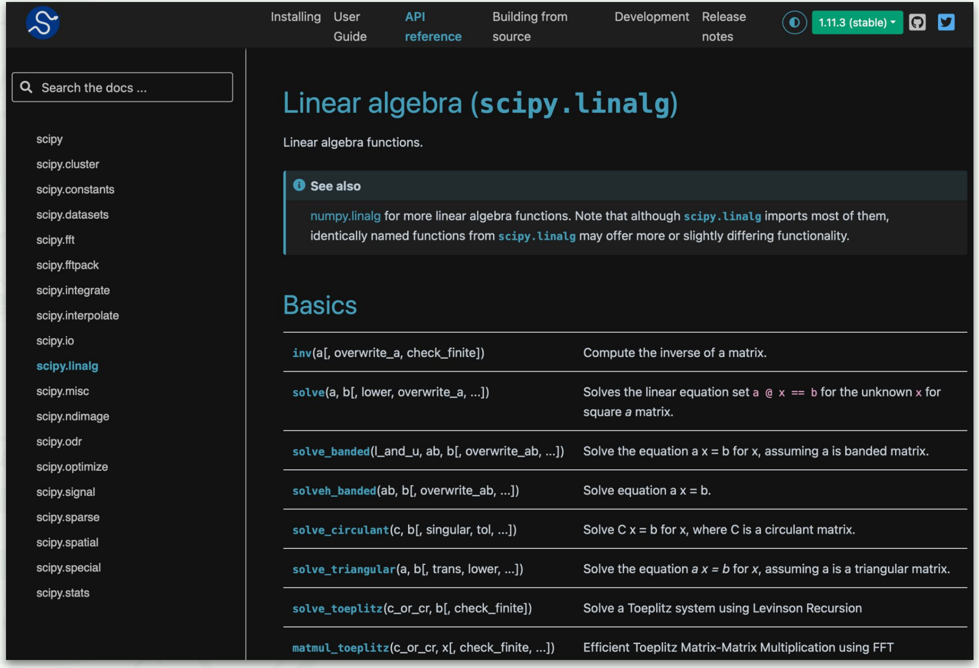
The NumPy linear algebra functions rely on BLAS and LAPACK to provide efficient low level implementations of standard linear algebra algorithms. Those libraries may be provided by NumPy itself using C versions of a subset of their reference implementations but, when possible, highly optimized libraries that take advantage of specialized processor functionality are preferred. Examples of such libraries are OpenBLAS, MKL (TM), and ATLAS. Because those libraries are multithreaded and processor dependent, environmental variables and external packages such as threadpoolctl may be needed to control the number of threads or specify the processor architecture.

The SciPy library also contains a `linalg` submodule, and there is overlap in the functionality provided by the SciPy and NumPy submodules. SciPy contains functions not found in `numpy.linalg`, such as functions related to LU decomposition and the Schur decomposition, multiple ways of calculating the pseudoinverse, and matrix transcendentals such as the matrix logarithm. Some functions that exist in both have augmented functionality in `scipy.linalg`. For example, `scipy.linalg.eig` can take a second matrix argument for solving generalized eigenvalue problems. Some functions in NumPy, however, have more flexible broadcasting options. For example, `numpy.linalg.solve` can handle “stacked” arrays, while `scipy.linalg.solve` accepts only a single square array as its first argument.

> **Note**
>
> The term *matrix* as it is used on this page indicates a 2d `numpy.array` object, and *not* a `numpy.matrix` object. The latter is no longer recommended, even for linear algebra. See the matrix object documentation for more information.

The @ operator

Introduced in NumPy 1.10.0, the `@` operator is preferable to other methods when computing the matrix product between 2d arrays. The `numpy.matmul` function implements the `@` operator.



- scipy
- scipy.cluster
- scipy.constants
- scipy.datasets
- scipy.fft
- scipy.fftpack
- scipy.integrate
- scipy.interpolate
- scipy.io
- scipy.linalg
- scipy.misc
- scipy.ndimage
- scipy.odr
- scipy.optimize
- scipy.signal
- scipy.sparse
- scipy.spatial
- scipy.special
- scipy.stats

Linear algebra (`scipy.linalg`)

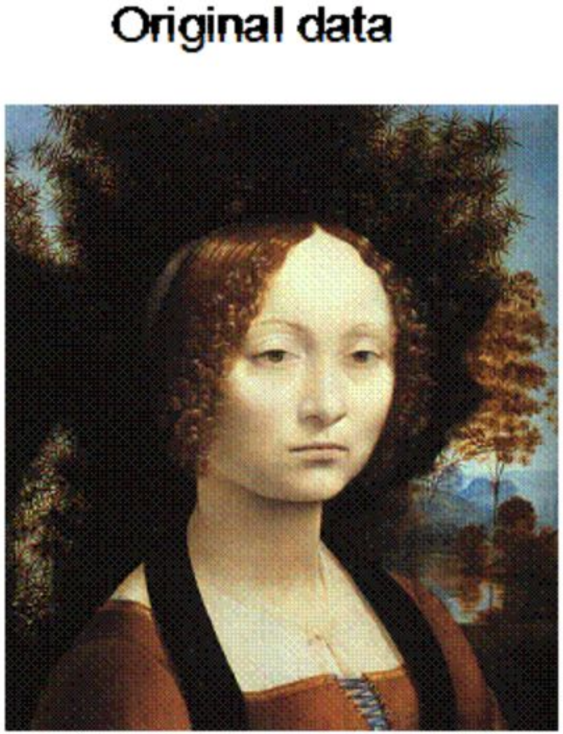
Linear algebra functions.

> **See also**
>
> numpy.linalg for more linear algebra functions. Note that although `scipy.linalg` imports most of them, identically named functions from `scipy.linalg` may offer more or slightly differing functionality.

Basics

`inv`(a[, overwrite_a, check_finite])	Compute the inverse of a matrix.
`solve`(a, b[, lower, overwrite_a, ...])	Solves the linear equation set `a @ x == b` for the unknown `x` for square *a* matrix.
`solve_banded`(l_and_u, ab, b[, overwrite_ab, ...])	Solve the equation a x = b for x, assuming a is banded matrix.
`solveh_banded`(ab, b[, overwrite_ab, ...])	Solve equation a x = b.
`solve_circulant`(c, b[, singular, tol, ...])	Solve C x = b for x, where C is a circulant matrix.
`solve_triangular`(a, b[, trans, lower, ...])	Solve the equation *a x = b* for *x*, assuming a is a triangular matrix.
`solve_toeplitz`(c_or_cr, b[, check_finite])	Solve a Toeplitz system using Levinson Recursion
`matmul_toeplitz`(c_or_cr, x[, check_finite, ...])	Efficient Toeplitz Matrix-Matrix Multiplication using FFT

Linear Algebra For Image Compression

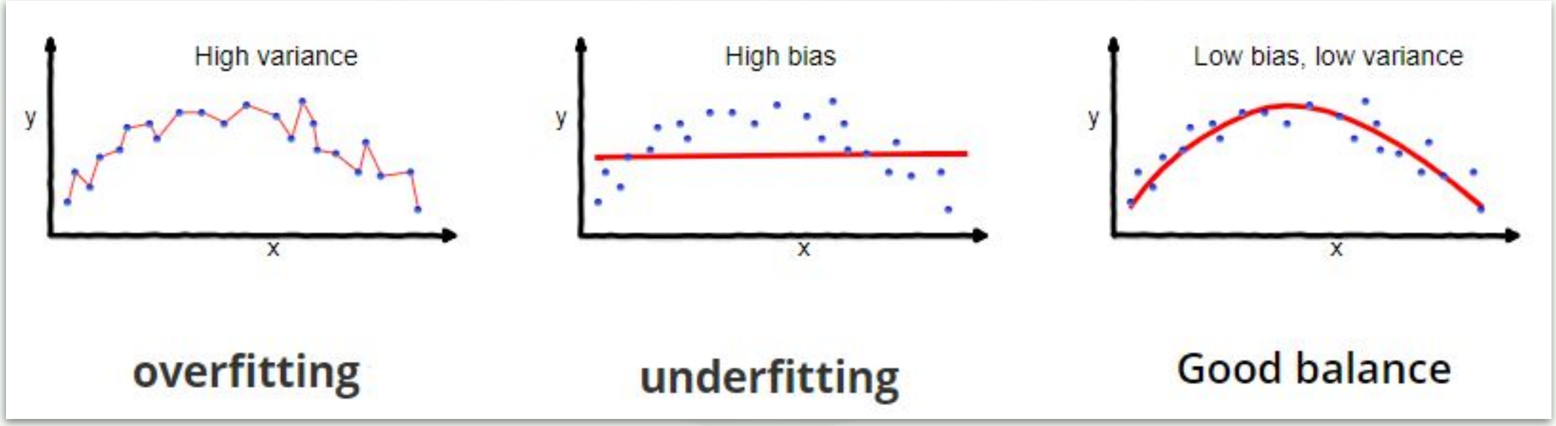


Bias-Variance Tradeoff

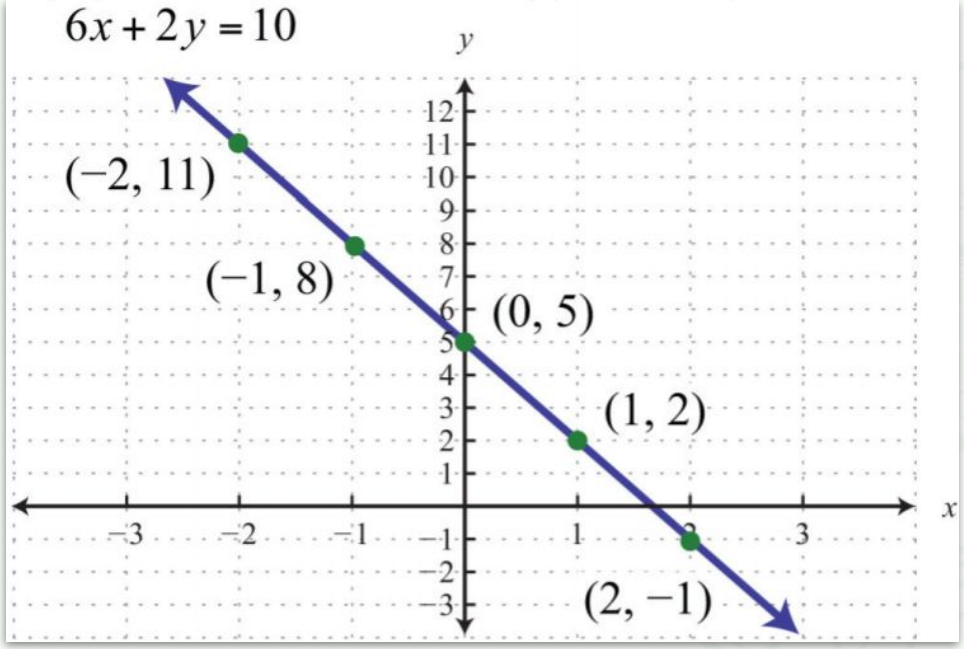
How should we pick our model?

- a line/hyperplane?
- radial basis function?
  - which one?

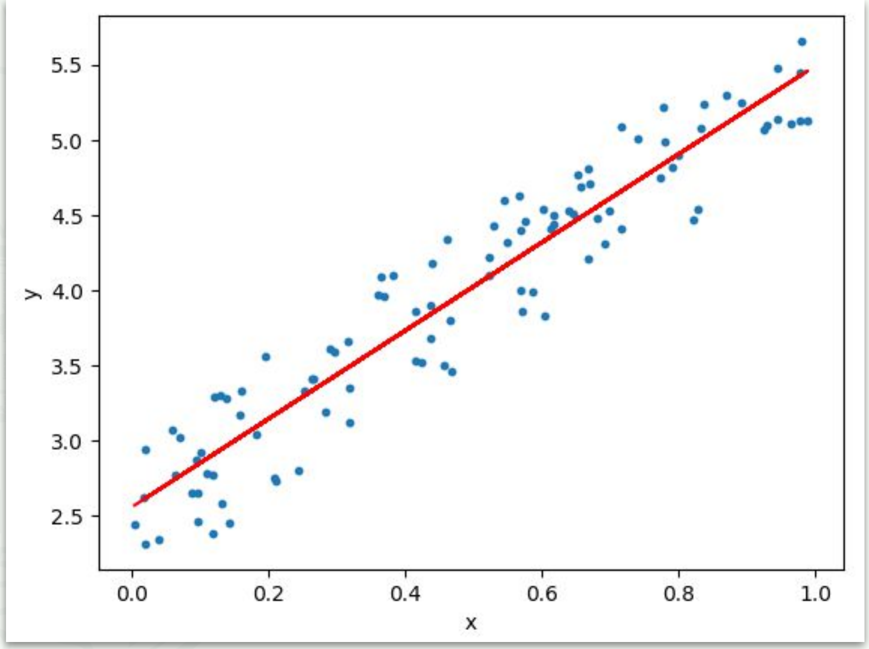
“Model selection” is the process of selecting the best model among a set of models using data.



Real Data: Too Many Equations, Not Enough Unknowns!



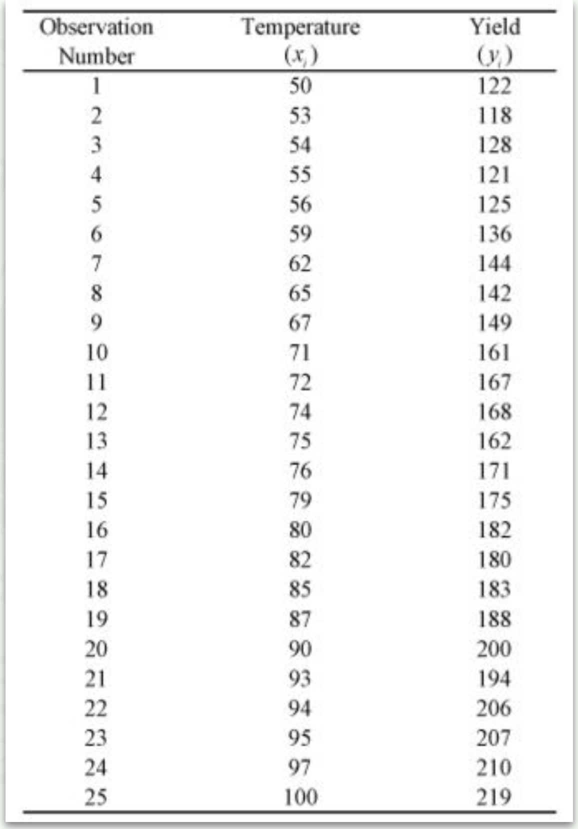
unrealistic!



realistic

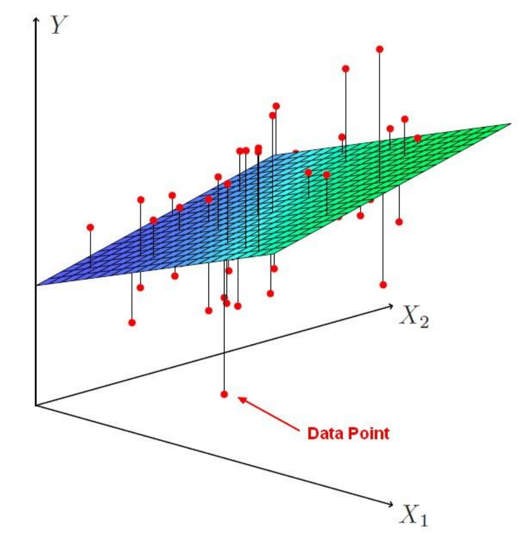
Data Matrix

- columns are features - perhaps as many as hundreds
- rows are data measurements - could have tens of thousands, or more



Observation Number	Temperature (x_i)	Yield (y_i)
1	50	122
2	53	118
3	54	128
4	55	121
5	56	125
6	59	136
7	62	144
8	65	142
9	67	149
10	71	161
11	72	167
12	74	168
13	75	162
14	76	171
15	79	175
16	80	182
17	82	180
18	85	183
19	87	188
20	90	200
21	93	194
22	94	206
23	95	207
24	97	210
25	100	219

Linear Algebra for Realistic Data Matrices



$$y = w_0 + w_1 x_1 + w_2 x_2,$$

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{N\times 1} = \underbrace{\begin{bmatrix} 1 & x_{11} & x_{21} \\ 1 & x_{12} & x_{22} \\ \vdots & & \\ 1 & x_{1N} & x_{2N} \end{bmatrix}}_{N\times 3} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}}_{3\times 1}$$

$$\mathbf{y} = X\mathbf{w}$$

We cannot invert this to find the weights because the data matrix *X* is not square.

We Solved This Previously Using Optimization

$$L(w_0, w_1, w_2) = \sum_d \left(y_d - (w_0 + w_1 x_{1d} + w_2 x_{2d})\right)^2,$$

$$\frac{\partial L}{\partial w_0} = 0,$$
$$\frac{\partial L}{\partial w_1} = 0,$$
$$\frac{\partial L}{\partial w_2} = 0$$

Minimize the distance from the hyperplane to the data points.

It may feel like we have two, very-different methods.

Can we solve the realistic case with linear algebra?

Matrix Times Its Transpose

$$\underbrace{X}_{n\times m}\underbrace{X^T}_{m\times n} = \underbrace{A}_{n\times n},$$

$$\underbrace{X^T}_{m\times n}\underbrace{X}_{n\times m} = \underbrace{B}_{m\times m}$$

Because matrices don’t commute, *A* and *B* are not the the same matrix. Order matters.

Note that:

1. *A* and *B* are square, and therefore might have an inverse.
2. *A* and *B* are symmetric matrices.

The Normal Equation

$$\begin{aligned}
\mathbf{y} &= X\mathbf{w}, \\
X^T\mathbf{y} &= X^T X\mathbf{w}, \\
X^T\mathbf{y} &= (X^T X)\mathbf{w}, \\
(X^T X)^{-1}X^T\mathbf{y} &= (X^T X)^{-1}(X^T X)\mathbf{w}, \\
(X^T X)^{-1}X^T\mathbf{y} &= I\mathbf{w},
\end{aligned}$$

$$\boxed{\mathbf{w} = (X^T X)^{-1}X^T\mathbf{y}}$$

Most important equation in data science?

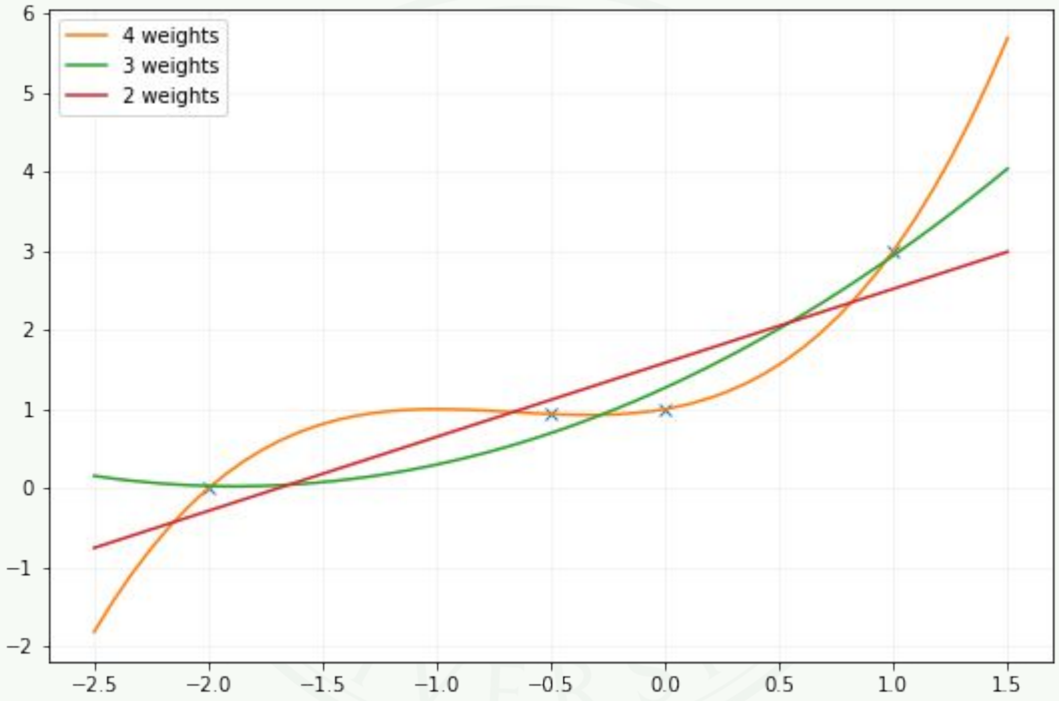
Pseudo-Inverse

A generalization of the inverse to non-square matrices is:

$$X^+ = (X^T X)^{-1} X^T$$

In this form, X^+ is referred to as the "*Moore-Penrose inverse*".

Homework Solution: Polynomial Regression



Matrix Decompositions: Big Picture

It is often convenient to write a matrix as a product of other matrices.

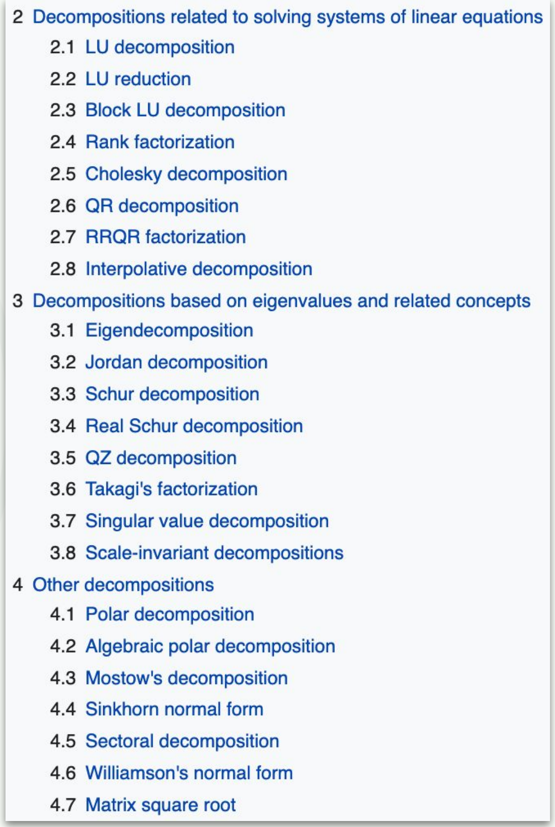
This might seem like a step backwards, but there are very good reasons for doing this.

Random example:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$A = LU$$

This decomposition is often used to find the inverse and/or determinant of a matrix. You use this when you use `linalg`.



2 Decompositions related to solving systems of linear equations
- 2.1 LU decomposition
- 2.2 LU reduction
- 2.3 Block LU decomposition
- 2.4 Rank factorization
- 2.5 Cholesky decomposition
- 2.6 QR decomposition
- 2.7 RRQR factorization
- 2.8 Interpolative decomposition

3 Decompositions based on eigenvalues and related concepts
- 3.1 Eigendecomposition
- 3.2 Jordan decomposition
- 3.3 Schur decomposition
- 3.4 Real Schur decomposition
- 3.5 QZ decomposition
- 3.6 Takagi's factorization
- 3.7 Singular value decomposition
- 3.8 Scale-invariant decompositions

4 Other decompositions
- 4.1 Polar decomposition
- 4.2 Algebraic polar decomposition
- 4.3 Mostow's decomposition
- 4.4 Sinkhorn normal form
- 4.5 Sectoral decomposition
- 4.6 Williamson's normal form
- 4.7 Matrix square root

Data Science: Singular Value Decomposition (SVD)

There is one decomposition that occurs across most of data science: the SVD. We'll focus on the SVD for this reason.

The SVD is important for two reasons:

- it generalizes ideas for square matrices to non-square data matrices,
- it reveals structure in the data that can be exploited.

$$A = U\Sigma V^T$$

SVD: Details

$$\underbrace{A}_{m\times n} = \underbrace{U}_{m\times m}\underbrace{\Sigma}_{m\times n}\underbrace{V^T}_{n\times n}$$

U and *V* are orthogonal (more generally: unitary):

$$U^T = U^{-1},$$
$$U^T U = I$$

Σ is a rectangular diagonal matrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Singular Values

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The diagonal values of Σ are called “singular values”, and by convention are taken to be ordered from largest to smallest:

$$\sigma_1 > \sigma_2 > \sigma_3 > \dots$$

Recall from last week that the columns of a matrix form a basis in a subspace. The number of linearly independent columns is called the “rank” of the matrix.

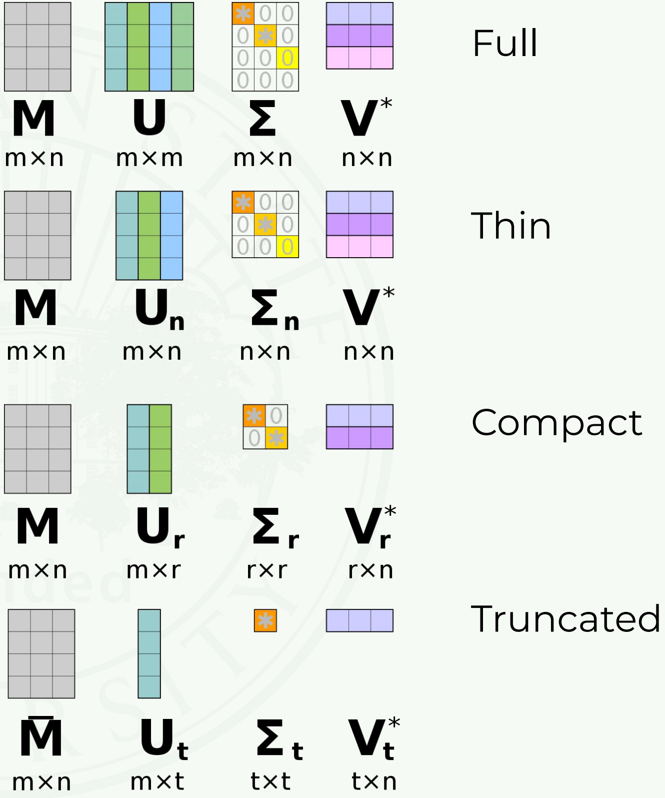
The number of non-zero singular values $\{\sigma_n\}$ is the rank of *A*.

Reduced SVDs: Thin, Compact, Skinny....

Sometimes you will see the SVD in a reduced form.

Usually this is for computational reasons. For example, reduced SVDs can save memory.

Often you can tell the library which form you want returned.



Linear Regression and Pseudoinverse Via SVD

$$\mathbf{y} = K\mathbf{w},$$

(now, *K* is not square - we can't easily invert this)

$$= U\Sigma V^T\mathbf{w},$$

$$U^T\mathbf{y} = \Sigma V^T\mathbf{w},$$

need to define this

$$\Sigma^{-1}U^T\mathbf{y} = V^T\mathbf{w},$$

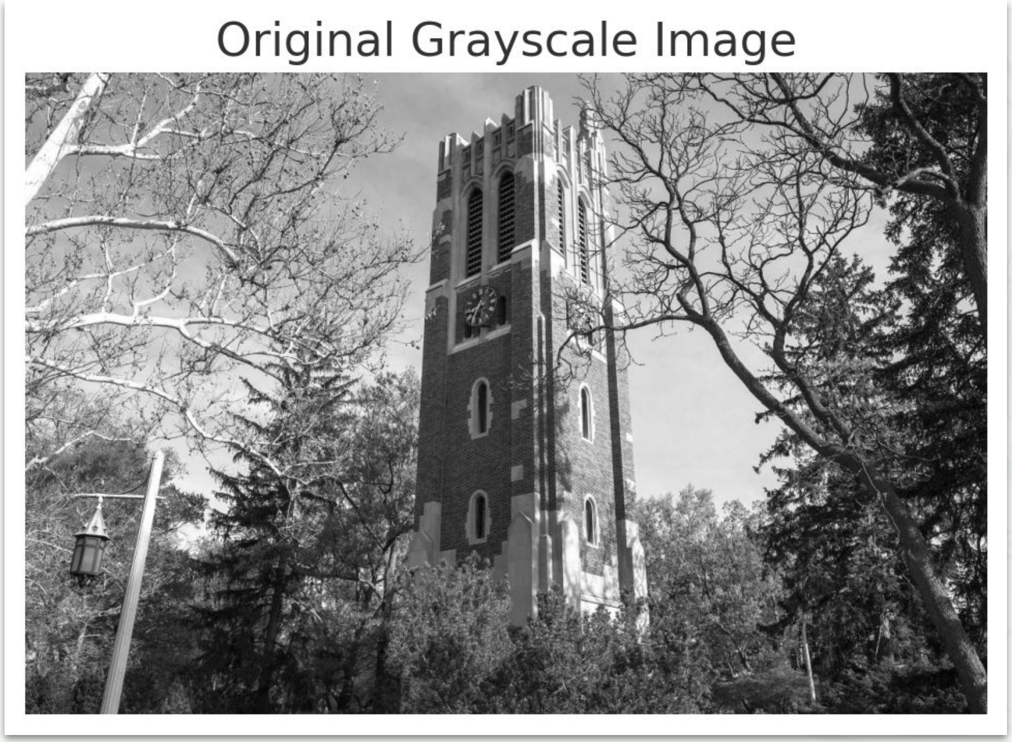
$$\mathbf{w} = V\Sigma^{-1}U^T\mathbf{y},$$

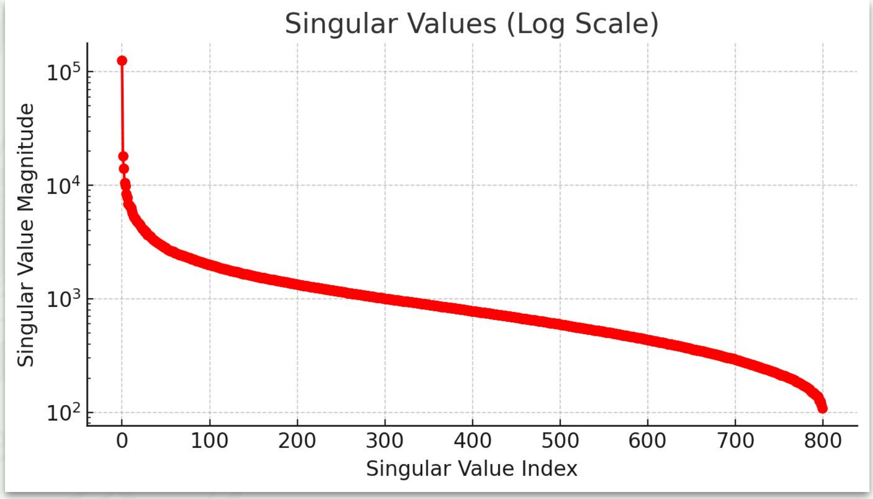
$$K^+ = V\Sigma^+U^T$$

Σ^+ is the pseudoinverse of Σ, formed by replacing all non-zero entries by their reciprocal and transposing.

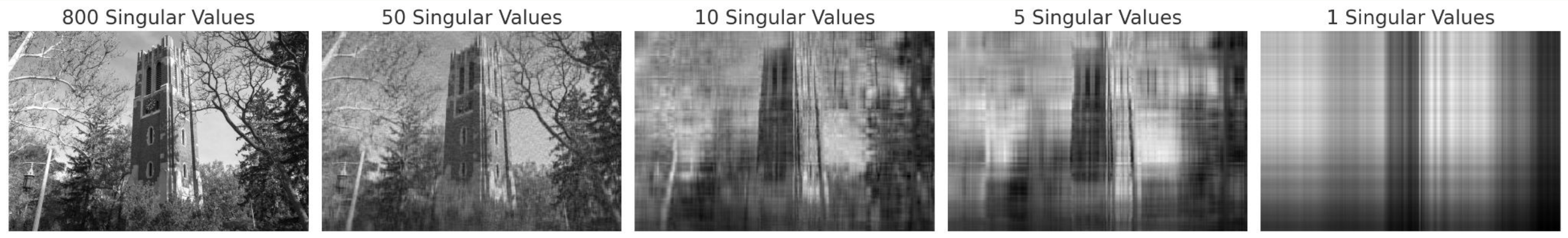
All zero entries remain zero.

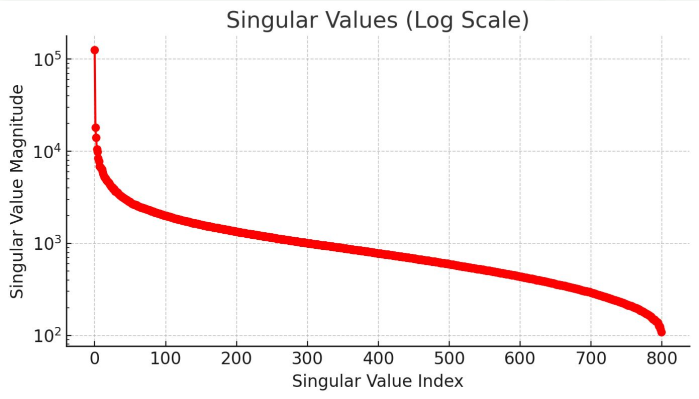
Let’s Play With SVD

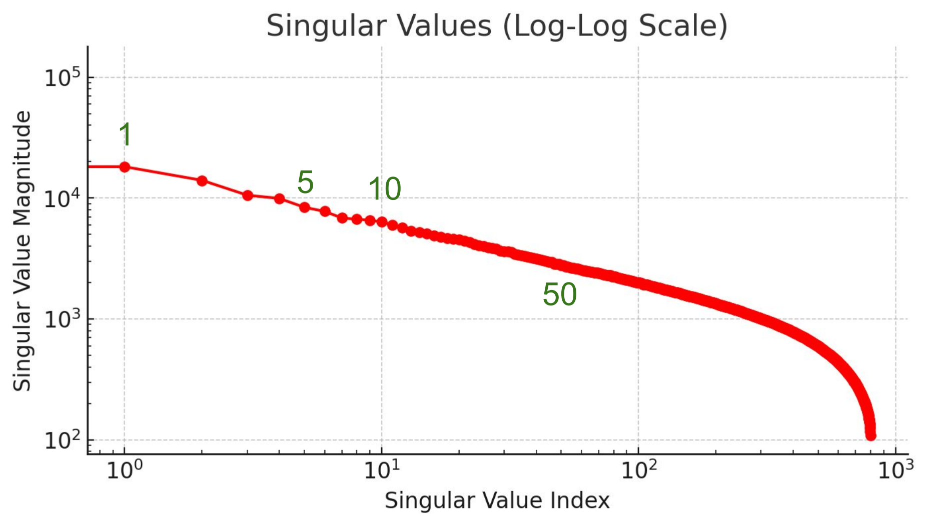

Original Grayscale Image



Let’s Play With SVD







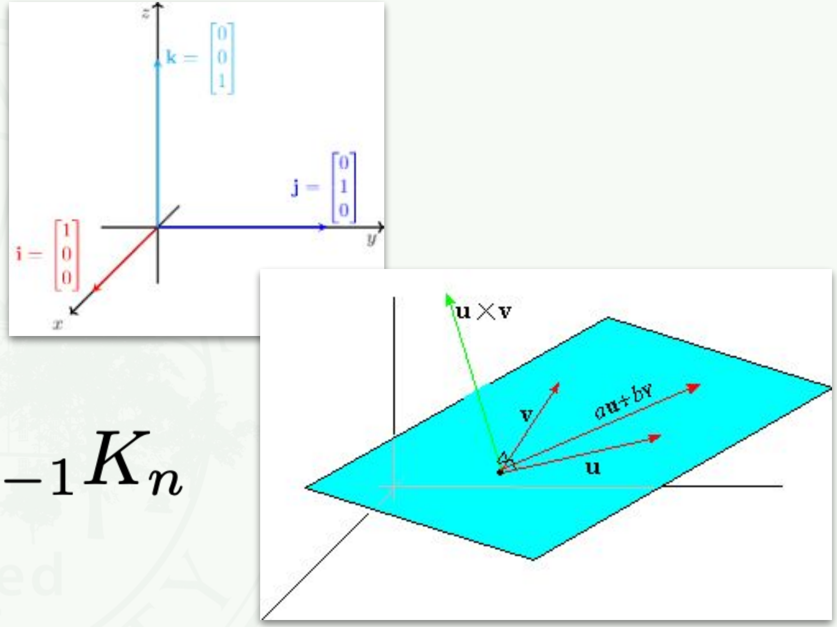
See you Wednesday!

Column Space: Geometric Interpretation

$$\mathbf{y} = K\mathbf{w},$$

$$= \begin{bmatrix} | & | & \cdots & | \\ K_1 & K_2 & \cdots & K_n \\ | & | & \cdots & | \end{bmatrix} \mathbf{w},$$

$$= w_0 K_1 + w_1 K_2 + \ldots + w_{n-1} K_n$$



Each column acts as a “unit vector”. If **y** is not in the “span” of the columns, then there is no solution.