Linear Algebra: A Brief Overview

Ott Toomet

April 5, 2018

The Big Picture

- Introduction
- Methods
  - python, numpy, pandas
  - statistics
  - linear algebra
- Linear regression
  - causality
  - diff-in-diff
  - instruments

- ML
  - k-Nearest Neighbors
  - logistic regression
  - Naive Bayes
- Methods
  - cross validation, bootstrap
  - model selection, regularization
  - dimensionality reduction, PCA
- ML
  - trees and forests
  - support vector machines
  - neural networks

What Is Linear Algebra

- Math with vectors and matrices
  - includes "addition" and "multiplication" $\Rightarrow$ it is an "algebra"
  - Multiplication and addition are *linear*:

$$A \times (B + C) = A \times B + A \times C \qquad \text{(distributive)}$$
$$A \times (\lambda \cdot C) = \lambda \cdot (A \times C)$$

$\times$ matrix multiplication
$\cdot$ *scalar multiplication*

Why Is It Useful

Express complex relationships in a compact way

- Example: equation system with 3 variables
- Non-matrix notation

$$
\begin{aligned}
x_1 + 2x_2 &= 7 \\
-x_2 + x_3 &= 6 \\
x_1 - 4x_2 + x_3 &= -2
\end{aligned}
$$

- Matrix notation

$$
\begin{pmatrix} 1 & 2 & 0 \\ 0 & -1 & 1 \\ 1 & -4 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \\ -2 \end{pmatrix}
$$

It Is Easier to Solve

- Non-Matrix: long and tedious and error-prone
  http://www.sosmath.com/soe/SE311105/SE311105.html
- Matrix:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 \\ 0 & -1 & 1 \\ 1 & -4 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 7 \\ 6 \\ -2 \end{pmatrix}$$

... And Computers Can Do It Too!

```
A = np.matrix([[1,  2,  0],
               [0, -1,  1],
               [1, -4,  1]])
B = np.matrix([[7], [6], [-1]])
A1 = np.linalg.inv(A)
x = A1.dot(B)
```

Another Example: Linear Regression

Non-matrix problem (2 variables):

$$Y_i = \beta_1 + \beta_2 T_i + \beta_3 G_i + \epsilon_i \qquad \text{for} \quad i = 1 \dots n$$

Inserting the specific variables of the example, we have

$$\begin{aligned}
b_1 n \quad &+ b_2 \Sigma_i T_i &+ b_3 \Sigma_i G_i &= \Sigma_i Y_i,\\
b_1 \Sigma_i T_i &+ b_2 \Sigma_i T_i^2 &+ b_3 \Sigma_i T_i G_i &= \Sigma_i T_i Y_i,\\
b_1 \Sigma_i G_i &+ b_2 \Sigma_i T_i G_i &+ b_3 \Sigma_i G_i^2 &= \Sigma_i G_i Y_i.
\end{aligned}$$

A solution can be obtained by first dividing the first equation by n and rearranging it to obtain

$$\begin{aligned}
b_1 &= \bar{Y} - b_2 \bar{T} - b_3 \bar{G}\\
&= 0.20333 - b_2 \times 8 - b_3 \times 1.2873. \qquad \textbf{(3-7)}
\end{aligned}$$

a set of two equations:

$$\begin{aligned}
b_2 \Sigma_i (T_i - \bar{T})^2 \quad &+ b_3 \Sigma_i (T_i - \bar{T})(G_i - \bar{G}) = \Sigma_i (T_i - \bar{T})(Y_i - \bar{Y}),\\
b_2 \Sigma_i (T_i - \bar{T})(G_i - \bar{G}) &+ b_3 \Sigma_i (G_i - \bar{G})^2 = \Sigma_i (G_i - \bar{G})(Y_i - \bar{Y}).
\end{aligned} \qquad \textbf{(3-8)}$$

This result shows the nature of the solution for the slopes, which can be computed from the sums of squares and cross products of the deviations of the variables. Letting lowercase letters indicate variables measured as deviations from the sample means, we find that the least squares solutions for b_2 and b_3 are

$$b_2 = \frac{\Sigma_i t_i y_i \Sigma_i g_i^2 - \Sigma_i g_i y_i \Sigma_i t_i g_i}{\Sigma_i t_i^2 \Sigma_i g_i^2 - (\Sigma_i g_i t_i)^2} = \frac{1.6040(0.359609) - 0.066196(9.82)}{280(0.359609) - (9.82)^2} = -0.0171984,$$

$$b_3 = \frac{\Sigma_i g_i y_i \Sigma_i t_i^2 - \Sigma_i t_i y_i \Sigma_i t_i g_i}{\Sigma_i t_i^2 \Sigma_i g_i^2 - (\Sigma_i g_i t_i)^2} = \frac{0.066196(280) - 1.6040(9.82)}{280(0.359609) - (9.82)^2} = 0.653723.$$

With these solutions in hand, the intercept can now be computed using (3-7); $b_1 = -0.500639$.

Matrix Approach

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & T_1 & G_1 \\ 1 & 1_2 & G_2 \\ \vdots & \vdots & \vdots \\ 1 & T_n & G_n \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

or

$$\mathbf{Y} = \mathsf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Solution:

$$\boldsymbol{\beta} = \left(\mathsf{X}\mathsf{X}'\right)^{-1} \mathsf{X}'\mathbf{Y}$$

3Blue1Brown LA course

https://www.youtube.com/watch?v=kjBOesZCoqc

Summary

- Linear algebra is a very important language in statistics
  - A lot of multivariate stuff easily generalizes into matrix form
  - Good software libraries exist
    - much faster than looping
    - can be done in openCL/GPU
  - A lot less error prone
  - You should be able to read it ...
  - ...and code it

Matrix

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{bmatrix}$$

Symmetric matrix $a_{ij} = a_{ji}$

Diagonal Matrix $a_{ij} = 0$ iff $i \neq j$

Identity matrix $a_{ij} = \mathbb{1}(i = j)$

Transposition

$$A' = [a_{ji}] = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{N1} \\ a_{12} & a_{22} & \dots & a_{N2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1M} & a_{2M} & \dots & a_{NM} \end{bmatrix}$$

Matrix

```
np.matrix("1 2 3; 4 5 6; 7 8 9")
```

```
matrix(c(1,2,3, 4,5,6, 7,8,9, 10,11,12), ncol=3)
```

1	5	9
2	6	10
3	7	11
4	8	12

Diagonal Matrix

```
np.diag(np.arange(4) + 1)
```

```
diag(1:4)
```

1	0	0	0
0	2	0	0
0	0	3	0
0	0	0	4

Unit Matrix

```
np.eye(4)
```

```
diag(4)
```

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Transposition

A.T

```r
a <- matrix(c(1,2,3, 4,5,6, 7,8,9, 10,11,12), ncol=3)
a
```

1	5	9
2	6	10
3	7	11
4	8	12

```r
t(a)
```

1	2	3	4
5	6	7	8
9	10	11	12

Transposition (python)

```
np.matrix("1 2 3; 4 5 6; 7 8 9; 10 11 12")

matrix([[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9],
        [10, 11, 12]])

np.matrix("1 2 3; 4 5 6; 7 8 9; 10 11 12").T

matrix([[ 1,  4,  7, 10],
        [ 2,  5,  8, 11],
        [ 3,  6,  9, 12]])
```

Matrix Operations

- multiplication by scalar: elementwise

$$\lambda A = \lambda \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{bmatrix} = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \dots & \lambda a_{1M} \\ \lambda a_{21} & \lambda a_{22} & \dots & \lambda a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda a_{N1} & \lambda a_{N2} & \dots & \lambda a_{NM} \end{bmatrix}$$

  where $\lambda \in \mathbb{R}$
  Note: this is true for unary minus $-A$.

- Addition, subtraction: elementwise

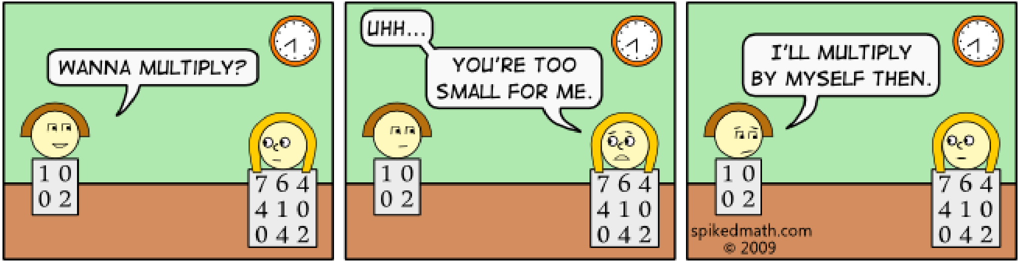
Matrix Multiplication

$$AB = [c_{ik}]$$

where

$$c_{ik} = \sum_{j=1}^{M} a_{ij} b_{jk}$$

- Number of rows of A must match the number of columns of B



Matrix Multiplication

- Properties:
  - Associative: $(AB)C = A(BC)$
  - Distributive: $A(B + C) = AB + AC$
  - Not commutative: $AB \neq BA$ (although $(AB)' = B'A'$)
- *Left-multiplication* and *right-multiplication*

Exercise

Create matrices

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 10 & 20 & 30 & 40 \end{pmatrix} \qquad C = \begin{pmatrix} -10 & -9 & -8 & -7 & -6 \\ -5 & -4 & -3 & -2 & -1 \end{pmatrix}$$

Compute:

1. $A \cdot B$
2. $A \cdot (B \cdot C')$ and $(A \cdot B) \cdot C'$
3. $A \cdot (B + C)$
4. $A \cdot B + A \cdot C$
5. $B' \cdot A'$
6. Check if the properties hold

Solution

```
a = np.matrix(np.arange(1,7).reshape(3,2))
print(a)
b = np.vstack((np.arange(5), 10*np.arange(5)))
print(b)
c = np.arange(-10,0).reshape((2,5))
print(c)
print(a@b)
print(a@(b + c))
print(a@b + a@c)
print(b.T @ a.T)
```

Vector

$1 \times n$ or $n \times 1$ matrix:

- Column vector

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

- Row vector

$$\mathbf{a}' = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix}$$

- Vector multiplication is regular matrix multiplication

$$aa' = [a_i a_j] \qquad \text{is } n \times n \text{ matrix}$$

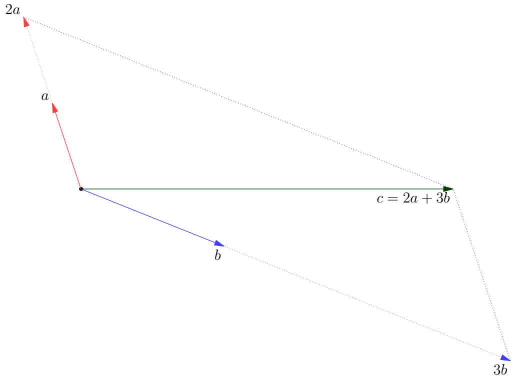
$$a'a = \left[\sum_{i=1}^{n} a_i^2\right] \qquad \text{is } 1 \times 1 \text{ matrix (scalar)}$$

Vector Addition

$\mathbf{a} = (-1, 3)$, $\mathbf{b} = (5, -2)$, compute $\mathbf{c} = 2\mathbf{a} + 3\mathbf{b}$.

- Solution: just multiply and sum the components:

$$\mathbf{c} = 2 \cdot (-1, 3) + 3 \cdot (5, -2) = (13, 0)$$



Rotation matrix

Let

$$\mathsf{A} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \dots & \dots \\ x_n & y_n \end{pmatrix}$$

Let

$$\mathrm{Rot}(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Now

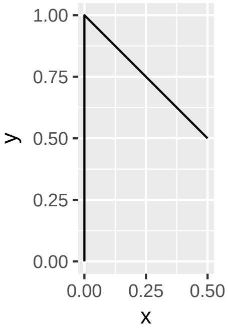
$$A \cdot \mathrm{Rot}(\alpha)$$

are the points of A rotated clockwise by α.

Rotation Example

Use data matrix A:

x	y
0.0	0.0
0.0	1.0
0.5	0.5

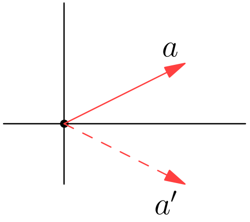


Exercise

Create transformation matrix F^y that flips the image vertically.

Solution

The matrix must keep x intact while negating y: it must turn each vector $\mathbf{a}$ into $\mathbf{a}'$:



Solution:

$$\mathbf{a} \cdot \mathsf{F}^y = (a_1, a_2) \cdot \begin{pmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{pmatrix} = (a_1 f_{11} + a_2 f_{21}, a_1 f_{12} + a_2 f_{22}) = (a_1, -a_2)$$

and hence

$$\mathsf{F}^y = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

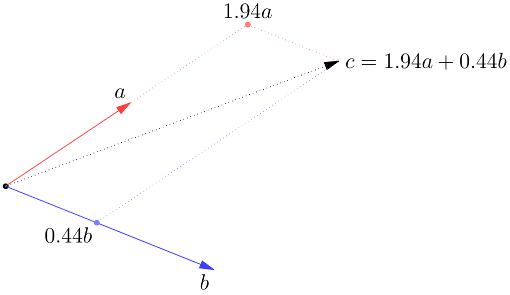
Vector Space

All the points we can describe with *base vectors*

- Using addition, scalar multiplication

Dimension:

- How many base vectors are needed to cover the space



$$\begin{pmatrix}\alpha & \beta\end{pmatrix}\begin{pmatrix}x_a & y_a \\ x_b & y_b\end{pmatrix} = \begin{pmatrix}x_c & y_c\end{pmatrix}$$

Linear Independence

Vectors $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$ are *linearly independent* iff

$$\alpha_1 \mathbf{a}_1 + \alpha_2 \mathbf{a}_2 + \cdots + \alpha_n \mathbf{a}_n = 0$$

$$\Updownarrow$$

$$\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$$

Column Space, Rank

Column space vector space, generated by the column vectors of the matrix

Column rank dimension of the column space

Row space vector space, generated by the column vectors of the matrix

Row rank dimension of the row space

Full column rank column rank equals to the number of columns

Full rank rank equals to the smallest of either number of rows or number of columns

Theorem

Row and column rank are equal

(and are called *matrix rank*)

Rank

```
a <- matrix(rnorm(4), 2, 2)
Matrix::rankMatrix(a)

## [1] 2
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 4.440892e-16
```

Numpy:

```
a = np.random.normal(size=(2,2))
np.linalg.matrix_rank(a)
```

Determinant

is a scalar function of matrix elements

- Useful descriptor of matrix properties in many contexts
- For 2×2 matrix $\mathsf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

$$\det \mathsf{A} \equiv |\mathsf{A}| = a_{11}a_{22} - a_{12}a_{21}$$

- for 3×3 matrix $\mathsf{B} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$

$$\det \mathsf{B} \equiv |\mathsf{B}| = b_{11}b_{22}b_{33} + b_{12}b_{23}b_{31} + b_{21}b_{32}b_{13} - \\ - b_{31}b_{22}b_{13} - b_{21}b_{12}b_{33} - b_{11}b_{23}b_{32}$$

Determinant

```
det(a)

## [1] -0.8946587
```

Numpy:

```
np.linalg.det(a)
```

Determinant 2

- Determinant of diagonal matrix $C = \begin{bmatrix} c_{11} & 0 & \dots & 0 \\ 0 & c_{22} & \dots & 0 \\ \dots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & c_{nn} \end{bmatrix}$

$$|C| = \prod_{i=1}^{n} c_{ii}$$

Determinant Property

Theorem

determinant is non-zero $\Leftrightarrow$ *matrix is full rank*

Easy to see for a diagonal matrix

Inverse Matrix

- A: square matrix

B is *inverse* A iff

$$BA = I,$$

and is denoted by A^{-1}.
Properties:

- $A^{-1} \cdot A = A \cdot A^{-1} = I$
- A^{-1} is also a square matrix
- A^{-1} is unique

For 2×2 matrix $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $A^{-1} = \frac{1}{|A|} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a11 \end{bmatrix}$

Only One Inverse

Theorem

Each square matrix has no more than one inverse matrix.

Proof.

Assume A *has two inverses:*

$$AB = I \quad \textit{and} \quad AC = I$$

Multiply first of these by C *from left:*

$$C(AB) = CI = C$$

But because matrix product is associative we have

$$C(AB) = (CA)B = IB = B$$

Hence $B = C$.

Inverse Matrix 2

- Inverse of diagonal matrix $C = \begin{bmatrix} c_{11} & 0 & \dots & 0 \\ 0 & c_{22} & \dots & 0 \\ \dots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & c_{nn} \end{bmatrix}$

$$C^{-1} = \begin{bmatrix} 1/c_{11} & 0 & \dots & 0 \\ 0 & 1/c_{22} & \dots & 0 \\ \dots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & 1/c_{nn} \end{bmatrix}$$

Inverse Properties

Definition

Matrix is *non-singular* $\Leftrightarrow$ inverse exists

Theorem

Matrix is non-singular $\Leftrightarrow$ *it is full rank*

Inverse

```
solve(a)
```

-0.045705	-1.0301813
-1.089282	-0.0965708

```
solve(a) %*% a
```

1	0
0	1

Numpy

```
np.linalg.inv(a)
```

Singular Matrix

Lets create a near-singular matrix:

```
a <- matrix(1:9, 3, 3)
Matrix::rankMatrix(a)

## [1] 2
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 6.661338e-16

solve(a)

## Error in solve.default(a):  Lapack routine dgesv:  system is
exactly singular:  U[3,3] = 0
```

Near-Singular Matrix

Make the matrix non-singular:

```
eps <- 1e-6
b <- a + eps*diag(3)
solve(b) %*% b
```

1	0	0
0	1	0
0	0	1

Yupee! It works!

- How does the size of ϵ affect the precision?

Characteristic Equation

Characteristic roots (eigenvalues) are solutions (λ) of the equation

$$\mathsf{A}\mathbf{c} = \lambda\mathbf{c}$$

Characteristic vectors (eigenvectors) are corresponding $\mathbf{c}$-s.
Rewrite:

$$(\mathsf{A} - \lambda\mathsf{I})\mathbf{c} = 0$$

$\Rightarrow$ $\mathsf{A} - \lambda\mathsf{I}$ must be singular $\Rightarrow |\mathsf{A} - \lambda\mathsf{I}| = 0$.

Symmetric Matrix

$n \times n$ symmetric matrix has

- n distinct characteristic vectors $\mathbf{c}_i$
- n real characteristic roots (not necessarily distinct)
- Characteristic vectors are orthogonal:

$$\mathbf{c}_i'\mathbf{c}_j = \mathbb{1}(i = j)$$

Example A

Matrix:

$$A = \begin{pmatrix} 30 & 28 \\ 28 & 30 \end{pmatrix}$$

Solve for eigenvalues (using the $\det A = 0$ condition $|A - \lambda I| = 0$):

$$|A| = (30 - \lambda)(30 - \lambda) - 28^2 = 0$$

The solution:

$$\lambda_1 = 58 \qquad \lambda_2 = 2 \tag{1}$$

Example B

The corresponding eigenvectors:

$$\begin{pmatrix} 30 & 28 \\ 28 & 30 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix} \tag{2}$$

and we have

$$\mathbf{c}_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \qquad \mathbf{c}_2 = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$$

Eigenvalues

```r
A <- matrix(c(30, 28, 28, 30), ncol=2)
eigen(A)

## eigen() decomposition
## $values
## [1] 58  2
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

Numpy:

`np.linalg.eig(A)`

Diagonalization

Characteristic equation in matrix form

$$AC = C\Lambda,$$

where

$$C = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_n \end{bmatrix}$$ and

$$\Lambda = \begin{bmatrix} \lambda_0 & 0 & 0 & \dots & 0 \\ 0 & \lambda_1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

Idempotent Matrix

```
C <- eigen(A)$vectors
C
```

0.7071068	-0.7071068
0.7071068	0.7071068

```
solve(C)
```

0.7071068	0.7071068
-0.7071068	0.7071068

```
C %*% t(C)
```

1	0
0	1

Characteristic roots

As characteristic vectors $\mathbf{c}$ orthogonal,

$$\mathsf{C}'\mathsf{C} = \mathsf{I}$$

- C is *idempotent* matrix
- $\mathsf{C}^{-1} = \mathsf{C}'$

Diagonalization:

$$\mathsf{A} = \mathsf{C}\Lambda\mathsf{C}'$$

or

$$\Lambda = \mathsf{C}'\mathsf{A}\mathsf{C}$$

Rank

Theorem

Rank of a symmetric matrix equals to the number of non-zero characteristic roots

$$\operatorname{rank} A = \operatorname{rank} \Lambda$$