Behind The Models: Cholesky Decomposition

The 19th Century Map-Maker's Trick That Runs Today's Linear Models and Monte Carlo Simulations

Tony Pistilli May 24, 2019 · 5 min read ★

André-Louis Cholesky is a bit of an oddity among mathematicians: his work was published posthumously after he died in battle during WWI. He discovered the linear algebra method that carries his name through his work as a late 19th century map maker, but it continues to be an efficient trick that fuels many machine learning models. This article will discuss the mathematical underpinnings of the method, and show two applications to linear regression and Monte-Carlo simulation.

How It Works

I'll try to keep the linear algebra short, but it's unavoidable: appreciate that linear algebra is simply a method for solving systems of equations efficiently, and also appreciate that advanced linear algebra methods are core to machine learning. Apologies aside, let's dive in.

Cholesky decomposition reduces a symmetric matrix into a lower-triangular matrix which when multiplied by it's transpose produces the original symmetric matrix. If that made zero sense, this is how it looks:

Cholesky decomposition takes the form: A = L x L*

```
from numpy import array
from numpy.linalg import cholesky

# define a 3x3 matrix
A = array([[36, 30, 18], [30, 41, 23], [18, 23, 14]])
print(L)

# Cholesky decomposition
L = cholesky(A)
print(L)
print(L.T)

# reconstruct
B = L.dot(L.T)
print(B)
```

The guts of this method get a little tricky — I'll present it here, but this would be the part of the post to skip over for the feint of heart — the fun stuff (applications) is below). We'll use A[1,1] notation to refer to row = 1, column= 1 in matrix A.

Cholesky decomposition is an iterative process. I'll stick to systems of equations notation below, but you'll see when we get to the third row that notating this using linear algebra would make a lot of sense. This write-up does a good job of explaining the matrix-algebra notation. Note that there are other methods for finding the Cholesky decomposition — Wikipedia explains several. All follow a similar type of flow.

First Row:

$$L[1,1] = \sqrt{A[1,1]} \quad \Rightarrow \quad \begin{bmatrix} \sqrt{36} = 6 & 0 & 0 \\ \_ & \_ & 0 \\ \_ & \_ & \_ \end{bmatrix}$$

Second Row:

$$L[2,1] = {}^{A[2,1]}\!/_{L[1,1]} \quad \Rightarrow \quad \begin{bmatrix} 6 & 0 & 0 \\ \frac{30}{6} = 5 & \_ & 0 \\ \_ & \_ & \_ \end{bmatrix}$$

$$L[2,2] = {}^{A[2,1]}\!/_{L[1,2]^2} \quad \Rightarrow \quad \begin{bmatrix} 6 & 0 & 0 \\ \frac{30}{6} = 5 & \frac{41}{5^2} = 4 & 0 \\ \_ & \_ & \_ \end{bmatrix}$$

L[1,1] * L[3,1] + L[1,2] * L[3,2] = A[3,1]

L[2,1] * L[3,1] + L[2,2] * L[3,2] = A[3,2]

Where L[3,1] = a and L[3,2] = b: 6a + 0b = 18 and 5a + 4b = 23 => $\begin{bmatrix} 6 & 0 & 0 \\ 5 & 4 & 0 \\ a = 3 & b = 2 & \_ \end{bmatrix}$

L[3, 3] = $\sqrt{A[3,3] - (L[3,1] * L[3,1] + L[3,2] * L[3,2])}$ = $\sqrt{14 - (3 * 3 + 2 * 2)}$

=> $\begin{bmatrix} 6 & 0 & 0 \\ 5 & 4 & 0 \\ 3 & 2 & 1 \end{bmatrix}$

Okay, so what? Now the cool part: using Cholesky decomposition we can solve systems of equations of any size in 2 steps. Suppose we want to solve for a, b, and c: remember that this could also be written as a system of 3 equations.

$$\begin{bmatrix} 36 & 30 & 18 \\ 30 & 41 & 23 \\ 18 & 23 & 14 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} * \begin{bmatrix} 288 \\ 296 \\ 173 \end{bmatrix}$$

A * x = b => solve for x

Ordinarily we'd stack the 3 equations on top of each other, solve for one variable dependent on the other two, plug it in, etc. Using Cholesky decomposition, we have a much more efficient method available.

*Where A * x = b*

$$L * c = b \quad => \begin{bmatrix} 6 & 0 & 0 \\ 5 & 4 & 0 \\ 3 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 48 \\ 14 \\ 1 \end{bmatrix} = \begin{bmatrix} 288 \\ 296 \\ 173 \end{bmatrix}$$

$$L^T * x = c \quad => \begin{bmatrix} 6 & 5 & 3 \\ 0 & 4 & 2 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 48 \\ 14 \\ 1 \end{bmatrix}$$

$$=> \ A * x = b \ => \begin{bmatrix} 36 & 30 & 18 \\ 30 & 41 & 23 \\ 18 & 23 & 14 \end{bmatrix} * \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 288 \\ 296 \\ 173 \end{bmatrix}$$

Solving for x using Cholesky Decomposition

A 3x3 matrix is a little underwhelming, but we can already begin to appreciate the efficiency of this method on a very large matrix.

Applications — Least-Squares Regression

finding the vector β where the squared differences between the predicted and actual Y values (i.e. RSS = “Residual Sum of Squares” or “Sum of Squared Errors”) are minimized.

$$min(RSS(\beta)) = min(y - X\beta)T\,(y - X\beta)$$

$$\frac{\partial RSS(\beta)}{\partial \beta} = -2X^T(Y - X\beta) = 0$$

$$X^T X\beta = X^T Y$$

We know that $X^T X$ can take the form of a Cholesky decomposition, which gives:

$$L^T L\beta = X^T Y$$

This allows us to solve for x = $L\beta$, and then solve for β.

If that doesn’t make sense, focus on this one take-away: the Cholesky decomposition is roughly twice as efficient as other methods for solving systems of linear equations.

Applications — Monte-Carlo Simulation

I saved the coolest application for last. Imagine that you want to generate many correlated normal random variables, but don’t want to deal with a massive multi-variate normal. Cholesky decomposition allows you to simulate uncorrelated normal variables and transform them into correlated noraml variables — cool!

Assume 3 Normal(0,1) random variables we want to follow the covariance matrix below, representing the underlying correlation and standard deviation matrices:

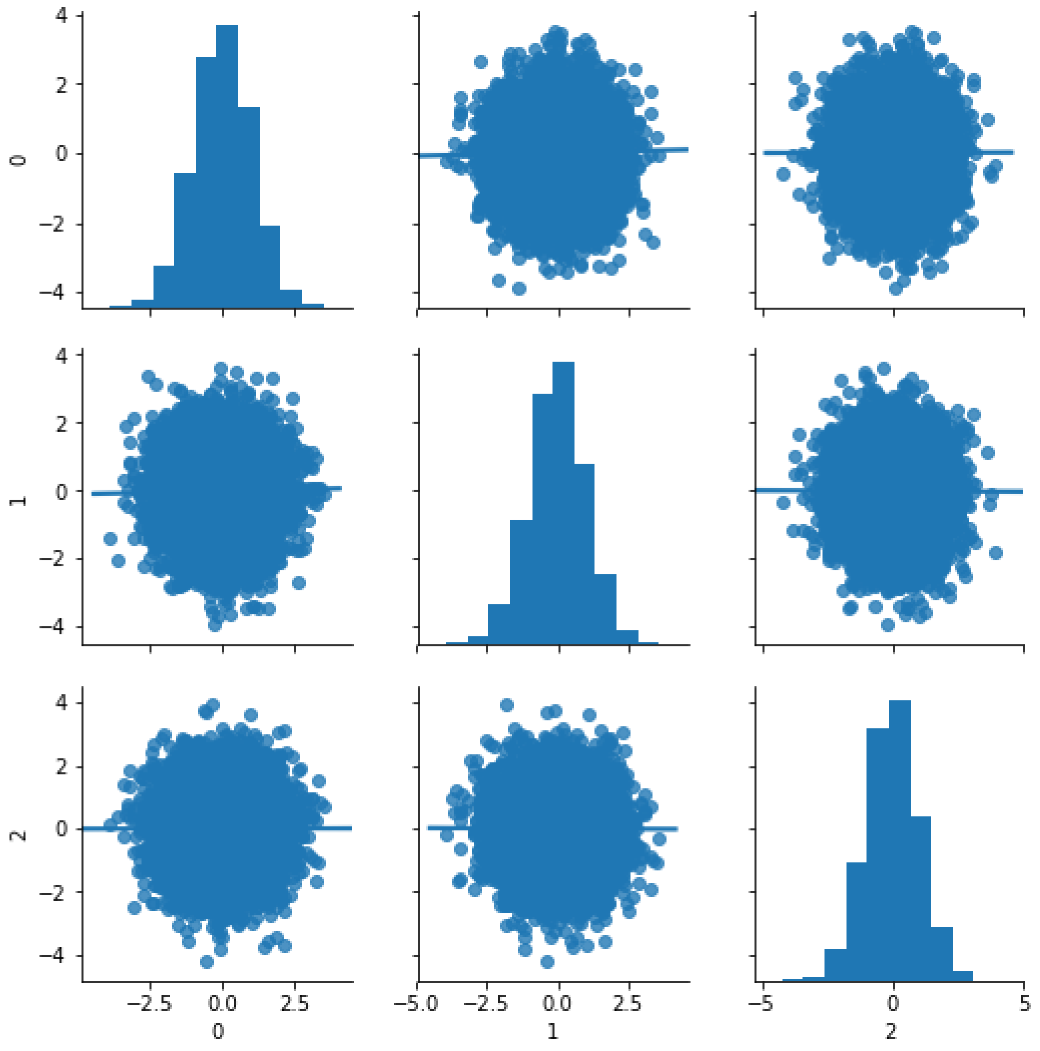
$$\text{Covariance Matrix} = \begin{bmatrix} 10.0 & -2.0 & 2.0 \\ -2.0 & 20.0 & 0.5 \\ 2.0 & 0.5 & 0.5 \end{bmatrix} \Rightarrow \text{Correlation Matrix} = \begin{bmatrix} 1.00 & -0.14 & 0.89 \\ -0.14 & 1.00 & 0.16 \\ 0.89 & 0.16 & 1.00 \end{bmatrix} \quad \text{Standard Deviations} = \begin{bmatrix} 3.16 \\ 4.47 \\ 0.71 \end{bmatrix}$$

We find the Cholesky decomposition of the covariance matrix, and multiply that by the matrix of uncorrelated random variables to create correlated variables.

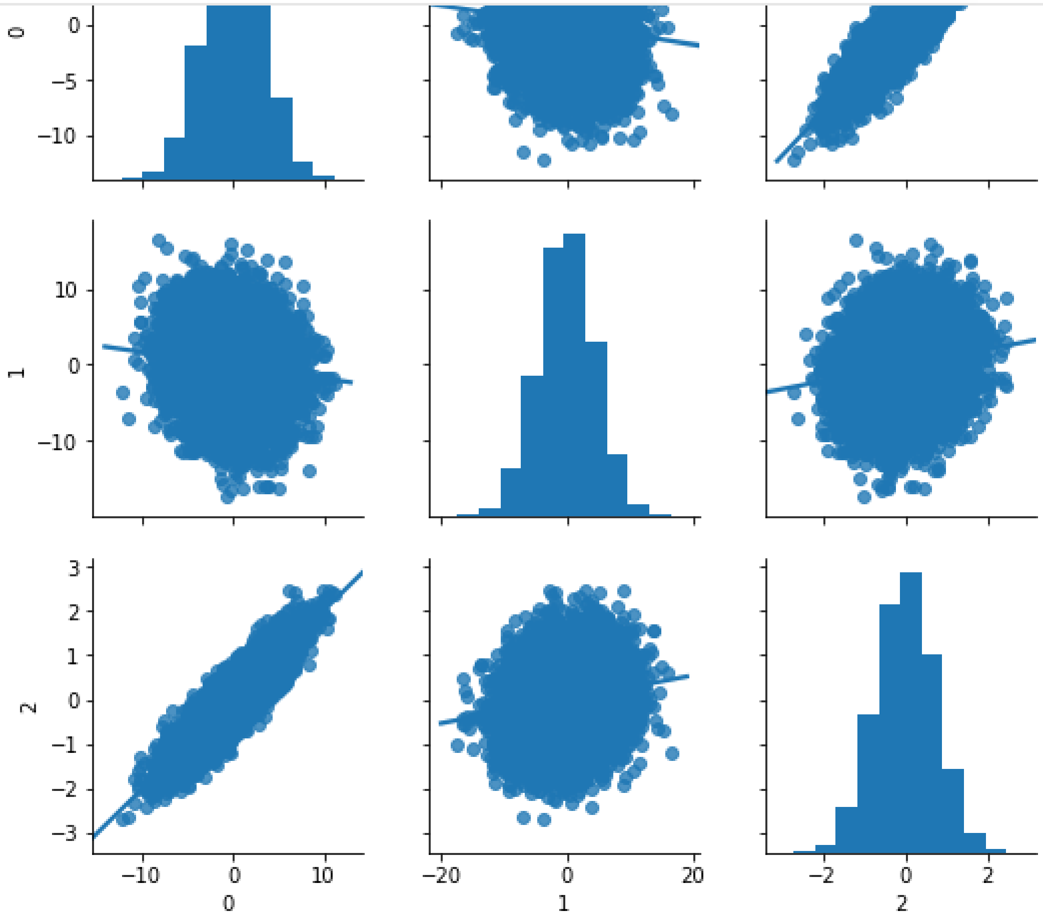
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

x_uncor = np.random.normal(0, 1, (3, 10000))
```

```
L = np.linalg.cholesky(cov)

x_cor = np.dot(L, x_uncor)
corr_simulated = pd.DataFrame(x_cor).T.corr()

std_ = np.sqrt(np.diag(cov))
corr_empirical = cov / np.outer(std_, std_)

x_uncor = pd.DataFrame(x_uncor.T)
x_cor = pd.DataFrame(x_cor.T)

sns.pairplot(x_uncor, kind="reg")
sns.pairplot(x_cor, kind="reg")
```

Voila! We go from uncorrelated:



To correlated:



Consistent with the correlation and standard deviation matrices presented above, columns 0 and 2 have a strongly positive correlation, 0 and 1 slightly negative, 1 and 2 slightly positive. The standard deviation of variable 2 is contained, while 0 and 1 are much wider.

Note that this does not work in the same way for non-normal random variables. In the above example, our correlated variables maintained a normal distribution. If we apply this method to gamma-generated random variables we see that the process does not hold.

Uncorrelated Gamma(1, 5) — everything looks good.

And Correlated:

is strictly positive. There is an easy back-door approximation that involves simulating correlated random variables, finding their inverse, and then drawing from the desired distribution using the inverse-correlated-normal values. This is not exact, but can get the job done. Exact methods tend to be fairly exotic.

Conclusion

Cholesky decomposition is a neat trick that underlies many machine learning applications, two of which were featured here: least-squares regression and Monte-Carlo simulation using correlated normal variables. While linear algebra can be a little scary, it's important to remember that it is just an efficient method for notating systems of linear equations, and that a high-level understanding of linear algebraic methods is crucial to understanding current machine learning algorithms.

Machine Learning Data Science Cholesky Decomposition Linear Algebra Monte Carlo