Unit 4: Covariance Matrices, PCA, and Stochastic Calculus

Charles Rambo

UCLA Anderson

2024

Table of Contents I

Covariance Matrices

Covariance Matrix

Definition

Suppose $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)^T$ is a multivariate random variable, and $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_n)^T$. The **covariance matrix** of X is

$$\Sigma = E\left[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^T\right].$$

Notice that

$$\Sigma_{ij} = \begin{cases} \text{Var}(X_i), & i = j \\ \text{Cov}(X_i, X_j), & i \neq j. \end{cases}$$

Multivariate Normal Distribution

Definition

The **multivariate normal distribution** or **Gaußian distribution** of dimension k has probability density function

$$f(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right),$$

where $\boldsymbol{\mu}$ is in $\mathbb{R}^k$ and Σ is the distribution's $k \times k$ covariance matrix. To denote that $\boldsymbol{X}$ follows a multivariate normal distribution, we write $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ or $\boldsymbol{X} \sim \mathcal{N}_k(\boldsymbol{\mu}, \Sigma)$.

Bivariate Normal Distribution Python Example

Example

Sample 100 points from $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\mu} = (0, 0)^T$ and $\Sigma =$

(a) $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

(b) $\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$

(c) $\begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$

(d) $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$.

Graph the results.

Bivariate Normal Distribution Example

```
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Use LaTeX
plt.rcParams['text.usetex'] = True

# Use Seaborn style
plt.style.use('seaborn')

# Set random seed
np.random.seed(0)

# Create list for problem parts
parts = ['a', 'b', 'c', 'd']

# Create list of covariance matrices
covs = [np.array([[1, 0],
          [0, 1]]),
          np.array([[1, 0.5],
          [0.5, 1]]),
          np.array([[1, -0.5],
          [-0.5, 1]]),
          np.array([[1, 1],
          [1, 1]])]

# Set up subplots
fig, ax = plt.subplots(2, 2, sharex = True,
       sharey = True, figsize = (10, 7))
```

```
# Loop over titles and covariance matrices
for i, part, cov in zip(range(4), parts,
     covs):

    # Get the row and column
    row, col = i // 2, i%2

    # Generate values
    vals = multivariate_normal.rvs(mean =
     np.zeros(2), cov = cov, size = 100)

    # Get x- and y-coordinates
    x, y = zip(*vals)

    # Plot the values
    ax[row, col].scatter(x, y)

    # Get title
    title = part + r': $\rho$ = ' + str(cov
     [0, 1])

    # Give the plot a title
    ax[row, col].title.set_text(title)

# Give entire figure title
fig.suptitle('Bivariate Normals')

# Save the figure
plt.savefig(path + r'ex4-1.png')

plt.show()
```

Bivariate Normal Distribution Example Result

Sample Covariance

In the sample covariance matrix, we divide by $n-1$ instead of n. Using the `pandas` data frame `df` the sample covariance matrix would be `df.cov()`.

Sample Covariance Matrix Example

Here is some code to get the sample covariance matrix for the *current* S&P constituents. The uploaded data are the daily constituents' returns from January 1, 2019 to May 31, 2024 which are available over the whole time range. The data source is *Yahoo! Finance*. Check out the Unit 4 Code Snippets to see how the data were extracted.

```
# Import modules
import pandas as pd

# Load in data; make date the index
data = pd.read_csv(data_path, index_col = 'Date')

# Get covariance matrix
S = data.cov()

S
```

Sample Covariance Matrix Result

The output looks like this:

	A	AAL	AAPL	ABBV	ABT	ACGL	ACN	ADBE	ADI	ADM	...	WTW	WY	WYNN	XEL	XOM
A	0.000342	0.000208	0.000191	0.000105	0.000178	0.000155	0.000192	0.000221	0.000225	0.000131	...	0.000141	0.000226	0.000218	0.000105	0.000122
AAL	0.000208	0.001444	0.000233	0.000093	0.000125	0.000303	0.000236	0.000192	0.000309	0.000240	...	0.000209	0.000447	0.000684	0.000067	0.000305
AAPL	0.000191	0.000233	0.000396	0.000102	0.000162	0.000153	0.000217	0.000296	0.000265	0.000127	...	0.000153	0.000245	0.000259	0.000119	0.000128
ABBV	0.000105	0.000093	0.000102	0.000249	0.000113	0.000117	0.000107	0.000105	0.000107	0.000088	...	0.000089	0.000127	0.000122	0.000081	0.000106
ABT	0.000178	0.000125	0.000162	0.000113	0.000264	0.000132	0.000159	0.000176	0.000170	0.000107	...	0.000132	0.000182	0.000119	0.000123	0.000080
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
XYL	0.000198	0.000322	0.000191	0.000097	0.000155	0.000216	0.000206	0.000192	0.000242	0.000169	...	0.000172	0.000286	0.000278	0.000131	0.000176
YUM	0.000122	0.000239	0.000137	0.000079	0.000114	0.000163	0.000149	0.000127	0.000160	0.000120	...	0.000135	0.000216	0.000238	0.000105	0.000132
ZBH	0.000155	0.000311	0.000153	0.000107	0.000142	0.000188	0.000175	0.000146	0.000194	0.000138	...	0.000144	0.000235	0.000312	0.000096	0.000178
ZBRA	0.000256	0.000337	0.000275	0.000107	0.000183	0.000200	0.000252	0.000297	0.000329	0.000174	...	0.000166	0.000325	0.000339	0.000102	0.000179
ZTS	0.000198	0.000162	0.000197	0.000108	0.000170	0.000149	0.000189	0.000213	0.000196	0.000104	...	0.000147	0.000215	0.000200	0.000119	0.000102

87 rows × 487 columns

Note: There are fewer than 500 columns because some of the current S&P constituents weren't publicly traded companies in 2019.

Principal Component Analysis (PCA)

Eigenvectors and Eigenvalues

From the spectral theorem, we know that Σ is diagonalizable, since it is symmetric. If Σ is $k \times k$, and the respective eigenvectors and eigenvalues are $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ and $\lambda_1, \lambda_2, \ldots, \lambda_k$. Then the total variance is $\lambda_1 + \lambda_2 + \ldots + \lambda_k$, and the fraction of variance explained by eigenvector $\boldsymbol{v}_i$ is

$$\frac{\lambda_i}{\lambda_1 + \lambda_2 + \ldots + \lambda_i + \ldots + \lambda_k}.$$

Eigenvectors and Eigenvalues Example

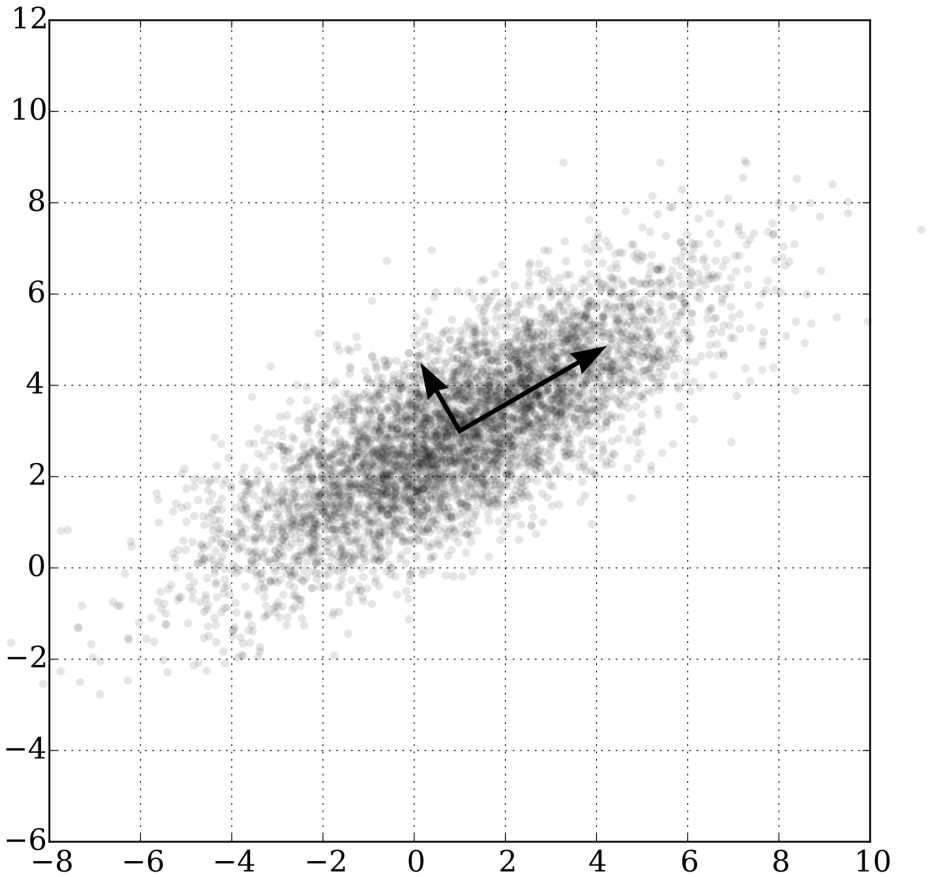
Example

Consider $\Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$. Find the eigenvectors and eigenvalues.

Principal Component Analysis

Principal component analysis (PCA) is a dimension reduction technique. It explains some of the variance of the original data in terms of a few eigenvectors with the largest eigenvalues.

Principal Component Analysis Figure



Principal Component Analysis Steps

1. Collect your data.
2. Standardize or demean your data.
3. Determine how much of the variance you need to explain or how many components are usable for your project.
4. Get the orthonormal basis of eigenvectors and eigenvalues.
5. Subset the orthonormal basis to the vectors corresponding to the largest ℓ eigenvalues, where the value of ℓ is based on step 3.
6. Calculate the "loadings" of each observation on the remaining basis elements.

Convention

Let's assume that

$$i < j \qquad \text{implies} \qquad \lambda_i \geq \lambda_j.$$

In other words, we've rearrange our eigenvalues and corresponding eigenvectors so that the eigenvalues are in descending order.

What are Loadings?

Suppose Σ's eigenvectors are the orthonormal eigenbasis $(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k)$. Then observation $\boldsymbol{u}$ can be written as

$$\boldsymbol{u} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \ldots + \alpha_\ell \boldsymbol{v}_\ell + \ldots + \alpha_k \boldsymbol{v}_k \qquad \text{where} \qquad \alpha_i = \frac{\boldsymbol{u} \bullet \boldsymbol{v}_i}{\|\boldsymbol{v}_i\|^2} = \boldsymbol{u} \bullet \boldsymbol{v}_i.$$

We say that $\boldsymbol{u}$ has a **loading** of α_i on $\boldsymbol{v}_i$.

A good approximation of $\boldsymbol{u}$ is the first ℓ components. Therefore, we can think of $\boldsymbol{u}$ as more or less the same as

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_\ell \end{pmatrix}$$

where the basis for this vector is $(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_\ell)$.

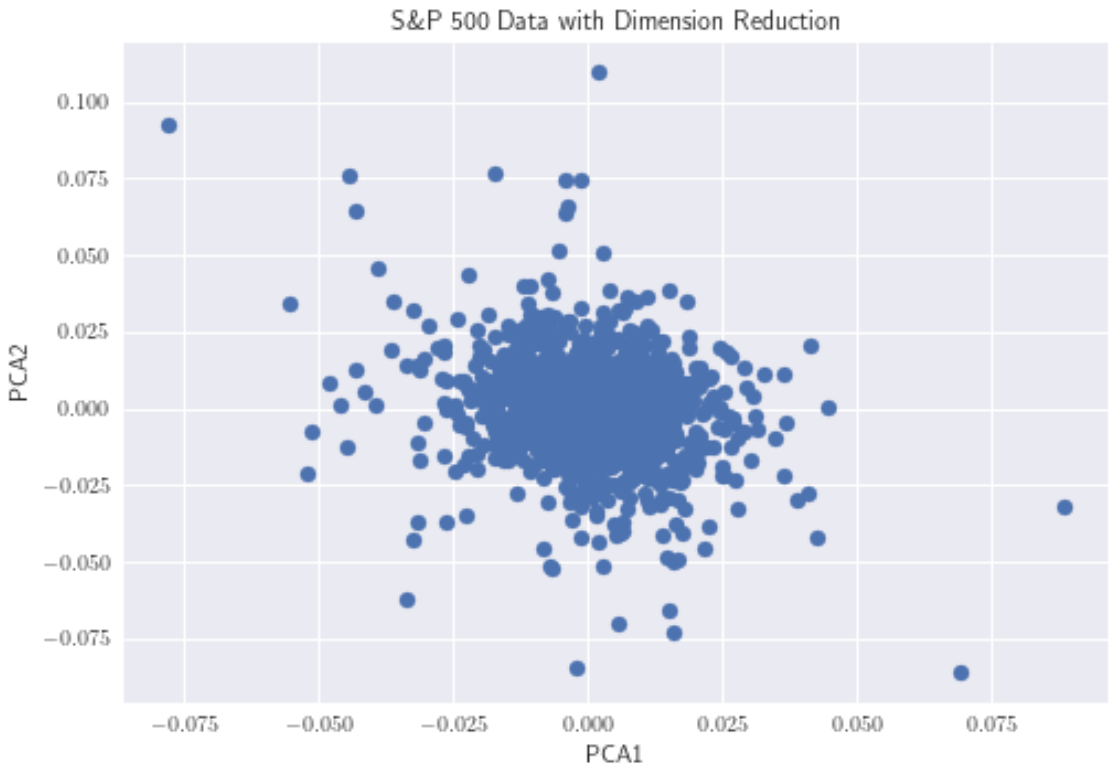
Principal Component Analysis Example

Example

Use PCA to represent the S&P 500 constituent data from before on a two dimensional scatter plot.

```
import numpy as np
import matplotlib.pyplot as plt

# Use Seaborn style
plt.style.use('seaborn')

# S and data are as calculated previously

# Get the eigenvalues and eigenvectors
evals, evecs = np.linalg.eigh(S)

# Indices in descending order
idx = evals.argsort()[::-1]

# Change order
evecs, evals = evecs[idx], evals[idx]

# Convert the observations to a numpy array
X = data.values

# Get the mean of each firm's return
x_bar = X.mean(axis = 0)

# Demean X
X -= x_bar

# Calculate loadings using the dot product
loadings = X @ evecs[:, 0:2]

# Unpack results
x, y = zip(*loadings)

# Get scatter plot
plt.scatter(x, y)

# Create x- and y-abels
plt.xlabel('PCA1'); plt.ylabel('PCA2')

# Give the plot a title
plt.title(r'S\&P 500 Data with Dimension
    Reduction')

# Save the figure
plt.savefig(path + r'ex4-2.png')

plt.show()
```

Principal Component Analysis Result



Reconstruction

If we have data with mean $\boldsymbol{\mu}$, and we used the loadings of the first ℓ eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_\ell$ to approximate $\boldsymbol{u}$, then this is making the assumption

$$\boldsymbol{u} \approx \boldsymbol{\mu} + \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \ldots + \alpha \boldsymbol{v}_\ell,$$

where α_i is the loading corresponding to $\boldsymbol{v}_i$.

Why Standardize?

PCA results are affected by the scale of your data. However, in many applications scale is already known and users are more interested in correlations within the data.

Clipping Covariance Matrices

Positive Definite Matrices

Using the dot product, a matrix S is *positive definite* if

$$\boldsymbol{x}^T S\boldsymbol{x} > 0 \qquad \text{for all} \qquad \boldsymbol{x} \neq \mathbf{0}.$$

Since a covariance matrix is diagonalizable due to the spectral theorem, a covariance matrix will be positive definite if and only if all its eigenvalues are positive.

Negative and Zero Eigenvalues

- When S is a covariance matrix, it never makes sense to have negative eigenvalues. These results are simply numerical errors.
- A zero eigenvalue indicates one variable can be written as a linear combination of the others, which may or may not make sense given the context.

Negative and Zero Eigenvalues Example

```
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Set the random seed
np.random.seed(0)

# Generate normal random variables
X = norm.rvs(size = (50, 100))

# Get the covariance matrix
S = np.cov(X, rowvar = False)

# Get the eigenvalues
evals, _ =  np.linalg.eigh(S)

print(f'The sample covariance matrix has {np.sum(np.isclose(evals, 0))}',
      r'eigenvalues numerically indistinguishable from 0.')
```

It says there are 51 eigenvalues numerically indistinguishable from 0. The true covariance matrix is the identity I and therefore the only true eigenvalue is 1.

Clip Covariance Matrices

One solution is to "clip" the sample covariance matrix. Simply replace all the eigenvalues that are too small with something bigger and reconstruct the covariance matrix.

Clip Covariance Matrices Example

Using the results from before.

```
# Get the standard deviations
stds = np.sqrt(np.diag(S))

# Get the correlation matrix
C = np.diag(1/stds) @ S @ np.diag(1/stds)

# Get the eigenvalues and vectors of C
evals_c, evecs_c = np.linalg.eigh(C)

# Make lam_min small positive outside of np
    .isclose threshold
lam_min = 1e-7

# Replace eigenvalues that are too small
evals_c[evals_c < lam_min] = lam_min

# Reconstruct correlation matrix
C_new = evecs_c @ np.diag(evals_c) @
    evecs_c.T

# Make sure still correlation matrix
C_new = (np.diag(np.sqrt(1/np.diag(C_new)))
     @ C_new
         @ np.diag(np.sqrt(1/np.diag(
    C_new))))

# Multiply by standard deviations to make
    it covariance matrix
S_new = np.diag(stds) @ C_new @ np.diag(
    stds)

# Get the eigenvalues
evals_new, _ =  np.linalg.eigh(S_new)

print(f'The new covariance matrix has {np.
    sum(np.isclose(evals_new, 0))}',
     r'eigenvalues numerically
    indistinguishable from 0.')
```

Now it says all the eigenvalues are positive! Inspection shows that the covariance matrix estimate is otherwise very similar to the sample covariance matrix.

Marchenko-Pastur Theorem

Theorem

Consider a matrix of independent and identically distributed random observations X of size $T \times N$, where the underlying process generating the observations has mean 0 and variance σ^2. If $q = T/N > 1$ is constant, the matrix $C = \frac{1}{T} X^T X$ has eigenvalues that converges to a distribution with probability density function

$$f(\lambda) = \begin{cases} \frac{q}{2\pi\sigma^2} \frac{\sqrt{(\lambda_+ - \lambda)(\lambda - \lambda_-)}}{\lambda}, & \lambda_- \leq \lambda \leq \lambda_+ \\ 0, & \text{otherwise} \end{cases}$$

where

$$\lambda_- = \sigma^2 \left(1 - \sqrt{\frac{1}{q}}\right)^2 \qquad \text{and} \qquad \lambda_+ = \sigma^2 \left(1 + \sqrt{\frac{1}{q}}\right)^2 .$$

Marchenko-Pastur Example

```
# Import modules
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import time

# Use LaTeX
plt.rcParams['text.usetex'] = True

# Use Seaborn style
plt.style.use('seaborn')

# Set the random seed
np.random.seed(0)

# Start the clock
start_time = time.perf_counter()

# Generate normal random variables
X = norm.rvs(size = (100_000, 10_000))
```

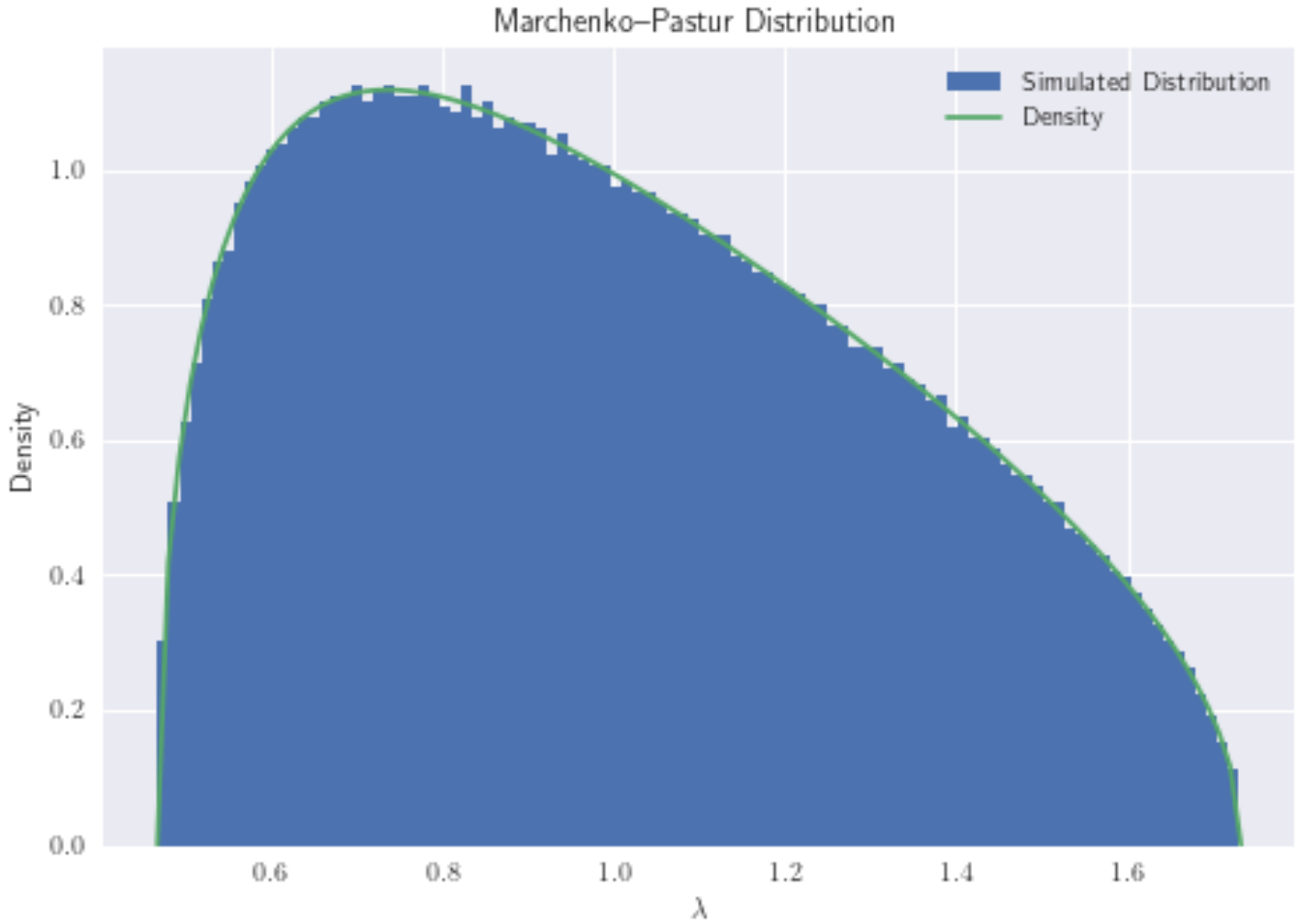
Marchenko-Pastur Example

```
# Get the number of observations and variables
T, N = X.shape

# Get correlation matrix
C = np.corrcoef(X, rowvar = False)

# q is the number of observations divided by the number of variables
q = T/N

# Get eigenvalues
evals, _ = np.linalg.eigh(C)

# Get support of MarchenkoPastur distribution
lam_minus, lam_plus = (1 - np.sqrt(1/q))**2, (1 + np.sqrt(1/q))**2

# Define pdf
def f(lam):

    # Support of MarchenkoPastur distribution
    if lam_minus <= lam <= lam_plus:

        return q/(2 * np.pi) * np.sqrt((lam_plus - lam) * (lam - lam_minus))/lam

    else:

        return 0
```

Marchenko-Pastur Example Cont.

```
# Get lam_vals
lam_vals = np.linspace(lam_minus, lam_plus, 100)

# Get density values
f_vals = f(lam_vals)

# Plot histogram
plt.hist(evals, density = True, bins = int(np.sqrt(N)), label = 'Simulated Distribution')

# Plot density
plt.plot(lam_vals, f_vals, label = 'Density')

# Add legend
plt.legend()

# Add x-label
plt.xlabel(r'$\lambda$')

# Add y-label
plt.ylabel(r'Density')

# Add title to plot
plt.title(r'Marchenko Pastur Distribution')

# Save the figure
plt.savefig(path + r'ex4-3.png')

plt.show()

print(f'This script took {(time.perf_counter() - start_time)/60:.2f} minutes to run.')
```

Marchenko-Pastur Result



How to use it?

Consider eigenvalue λ of the covariance matrix.

- If $\lambda > \lambda_+$, then result consistent with signal.
- If $\lambda \leq \lambda_+$, the result is probably just random noise. Replace values less than λ_+ with mean of values less than λ_+.

S&P 500 Constituent Example

Suppose we have the S&P 500 constituent data from before.

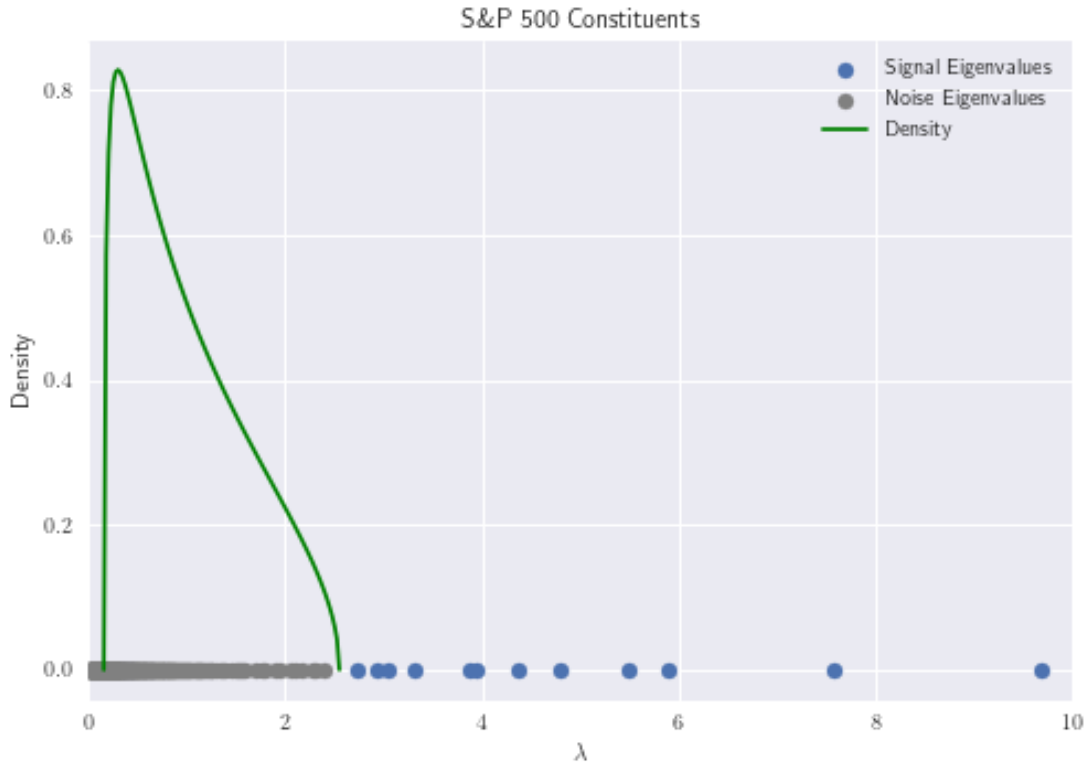
```
# Get the standard deviations
stds = np.sqrt(np.diag(S))

# Calculate correlation matrix
C = np.diag(1/stds) @ S @ np.diag(1/stds)

# Get the eigenvalues and vectors
evals, evecs =  np.linalg.eigh(C)

# Save q
q = data.shape[0]/data.shape[1]

# Get support of Marchenko-Pastur distribution
lam_minus, lam_plus = (1 - np.sqrt(1/q))**2, (1 + np.sqrt(1/q))**2

# Define pdf
def f(lam):

    # Support of MarchenkoPastur distribution
    if lam_minus <= lam <= lam_plus:
        return q/(2 * np.pi) * np.sqrt((lam_plus - lam) * (lam - lam_minus))/lam
    else:
        return 0

# Get lam_vals
lam_vals = np.linspace(lam_minus, lam_plus, 100)

# Get density values
f_vals = f(lam_vals)
```

S&P 500 Constituent Example

```
# Plot histogram
plt.scatter(evals[evals > lam_plus], np.zeros(np.sum(evals > lam_plus)),
            label = 'Signal Eigenvalues')

plt.scatter(evals[evals <= lam_plus], np.zeros(np.sum(evals <= lam_plus)),
            label = 'Noise Eigenvalues', color = 'gray')

# Plot density
plt.plot(lam_vals, f_vals, label = 'Density', color = 'green')

# There are 3 eigenvalues much larger than 10
plt.xlim([0, 10])

# Add legend
plt.legend()

# Add x-label
plt.xlabel(r'$\lambda$')

# Add y-label
plt.ylabel(r'Density')

# Add title to plot
plt.title(r'S\&P 500 Constituents')

# Save the figure
plt.savefig(path + r'ex4-4.png')

plt.show()
```

S&P 500 Constituent Example

Only the values past λ_+ are consistent with signal. There are three eigenvalues greater than 10 which are not shown.



Clip Matrix

```
# Initialize new eigenvalues
evals_new = evals

# Replace noise eigenvalues with mean
evals_new[evals_new < lam_plus] = np.mean(evals_new[evals_new < lam_plus])

# Construct new correlation matrix
C_new = evecs @ np.diag(evals_new) @ evecs.T

# Make sure still correlation matrix
C_new = np.diag(1/np.sqrt(np.diag(C_new))) @ C_new @ np.diag(1/np.sqrt(np.diag(C_new)))

# Make new covariance matrix
S_new = np.diag(stds) @ C_new @ np.diag(stds)
```

Stochastic Calculus

Source

I'm following these notes very closely: `http://www.columbia.edu/~mh2078/FoundationsFE/IntroStochCalc.pdf`

Probability Triple

We assume we have the probability space $(\Omega, \mathcal{F}, P)$ where

- Ω is the universe of possible outcomes.
- $\mathcal{F}$ represents the σ-algebra of events in Ω.
- P is the "true" or physical probability measure.

Filtration

There is also a **filtration** $\{\mathcal{F}_t\}_{t\geq 0}$ of σ-algebras that models the evolution of information through time. Since information increases over time $\mathcal{F}_s \subseteq \mathcal{F}_t$ for $s < t$.

If it is know by time t whether or not an event E has occurred, then we have $E \in \mathcal{F}_t$. If we are working with a finite horizon $[0, T]$, then we can take $\mathcal{F} = \mathcal{F}_T$.

Stochastic Process

Definition

For a given probability space $(\Omega, \mathcal{F}, P)$, a **stochastic process** is a collection of random variables indexed by $\mathcal{T}$. We often write $\{X_t | t \in \mathcal{T}\}$ to denote a stochastic process, and we think of $\mathcal{T}$ as the time index.

Stochastic Process Example

Example

For a high yield bond portfolio, we can model the total number of defaulted bonds this year up to day t as a stochastic process. Denote the number of defaulted bonds on day t by N_t. In this case, $\mathcal{T} = \{1, 2, 3, \ldots, 252\}$, assuming there are 252 days when the market is open. Using our prior notation, the stochastic process is $\{N_t | t \in \mathcal{T}\}$.

$\mathcal{F}_t$-Adapted

Definition

We say that a stochastic process X_t is $\mathcal{F}_t$-**adapted** if for every t in $\mathcal{T}$ the information about X_t is contained in $\mathcal{F}_t$.

Brownian Motion

Definition

A stochastic process $\{W_t | t \geq 0\}$ is a **standard Brownian motion** if the following hold.

- BM.1 $W_0 = 0$.
- BM.2 *It has continuous sample paths.*
- BM.3 $W_{t_1} - W_{s_1}$ *and* $W_{t_2} - W_{s_2}$ *are independent for* $0 \leq s_1 \leq t_1 \leq s_2 \leq t_2$.
- BM.4 $W_t - W_s \sim \mathcal{N}(0, t - s)$ *for* $0 \leq s \leq t$.

Simulating Brownian Motion

Suppose that we want to simulate a Brownian motion on the interval $[0, T]$. Then construct a partition of the interval

$$0 = t_0 < t_1 < \ldots < t_{n-1} < t_n = T.$$

For $i = 1, 2, \ldots, n$, generate $Z_i \sim \mathcal{N}(0, 1^2)$. Then

$$\widetilde{W}_{t_k} = \begin{cases} 0, & k = 0 \\ \sum_{i=1}^{k} Z_i \sqrt{\Delta t_i}, & k = 1, 2, \ldots, n \end{cases}$$

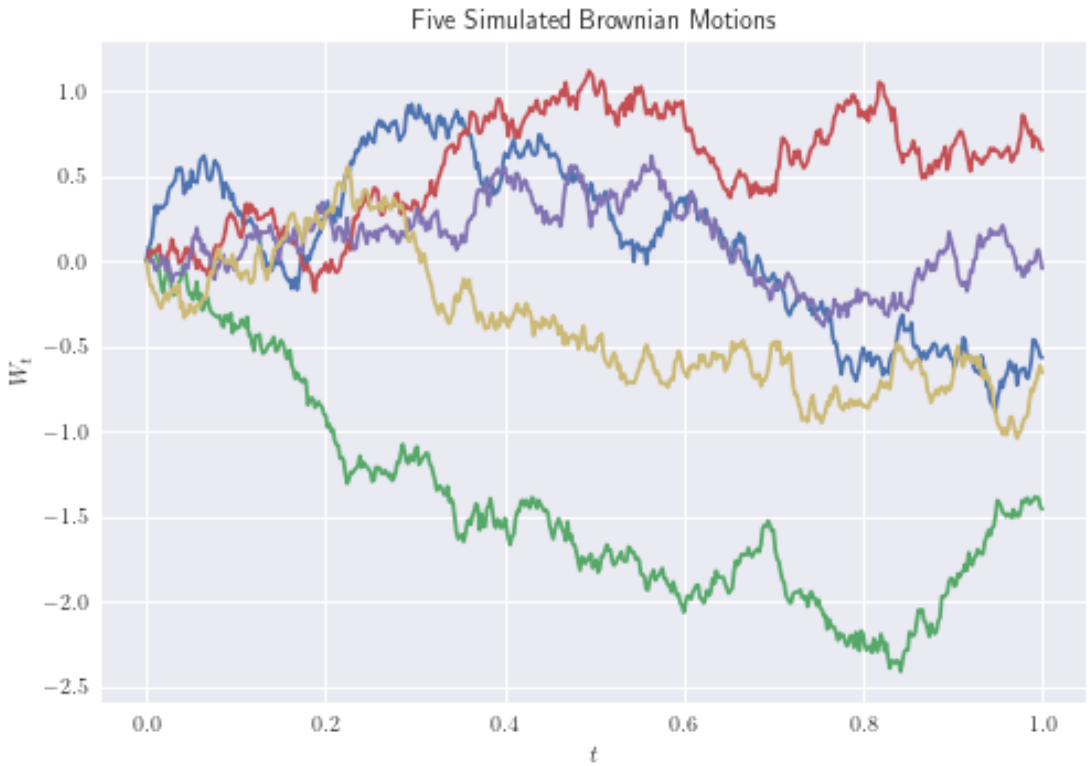
is "approximately" Brownian motion.

Python Code: Five Brownian Motions on [0, 1]

```
import numpy as np, matplotlib.pyplot as
    plt
from scipy.stats import norm

# Use LaTeX
plt.rcParams['text.usetex'] = True

# Use Seaborn style
plt.style.use('seaborn')

# Set the random seed
np.random.seed(0)

# Break up into n discrete intervals
n = 500

# Simulate five Brownian motions
for _ in range(5):

    # Simulate Brownian motion for t in [0,
      1]
    Z = norm.rvs(scale = np.sqrt(1/n), size
      = n)

    # Take the cumulative sum; add 0 for
      the t = 0 value
    W = np.insert(np.cumsum(Z), 0, 0)

    # Plot results
    plt.plot(np.linspace(0, 1, n + 1), W)

# Add x-label
plt.xlabel(r'$t$')

# Add y-label
plt.ylabel(r'$W_t$')

# Add title to plot
plt.title(r'Five Simulated Brownian Motions
    ')

# Save the figure
plt.savefig(path + r'ex4-5.png')

plt.show()
```

Output

Five Brownian motions on the interval $[0, 1]$, using a uniform partition that breaks $[0, 1]$ into 500 subintervals.



Martingale

Definition

A stochastic process $\{X_t | 0 \leq t \leq \infty\}$ is a **martingale** with respect to the filtration $\mathcal{F}_t$ if the following hold.

M.1 $E[|X_t|] < \infty$ *for all* $t \geq 0$.

M.2 $E[X_{t+s}|\mathcal{F}_t] = X_t$ *for all* $t, s \geq 0$.

Martingale Example

Example

Prove the following are martingales.

(a) W_t

(b) $W_t^2 - t$

(c) $\exp\left(\theta W_t - \frac{\theta^2 t}{2}\right)$

Quadratic Variation

Definition

Let X_t be some stochastic process. The **quadratic variation** of a stochastic process X_t is

$$\lim_{\|\mathcal{P}\| \to 0} \sum_{k=1}^{n} \left(X_{t_k} - X_{t_{k-1}} \right)^2,$$

where $\mathcal{P}$ is an arbitrary partition of $[0, T]$ and $\|\mathcal{P}\| = \max_k \{\Delta t_k\}$ is the mesh of the partition.

Quadratic Variation Example

Example

Compute the quadratic variation of the deterministic process $X_t = t^2$.

Quadratic Variation Differentiable Function

Any differentiable function will end up having quadratic variance 0, like we saw for t^2 in our example.

Quadratic Variation of Brownian Motion

Theorem

The quadratic variation of a standard Brownian motion is equal to T with probability 1.

Theorem (Levey's Theorem)

A continuous martingale is a standard Brownian motion if and only if its quadratic variation over each interval $[0, t]$ is equal to t.

Approximate Quadratic Variation Python Code

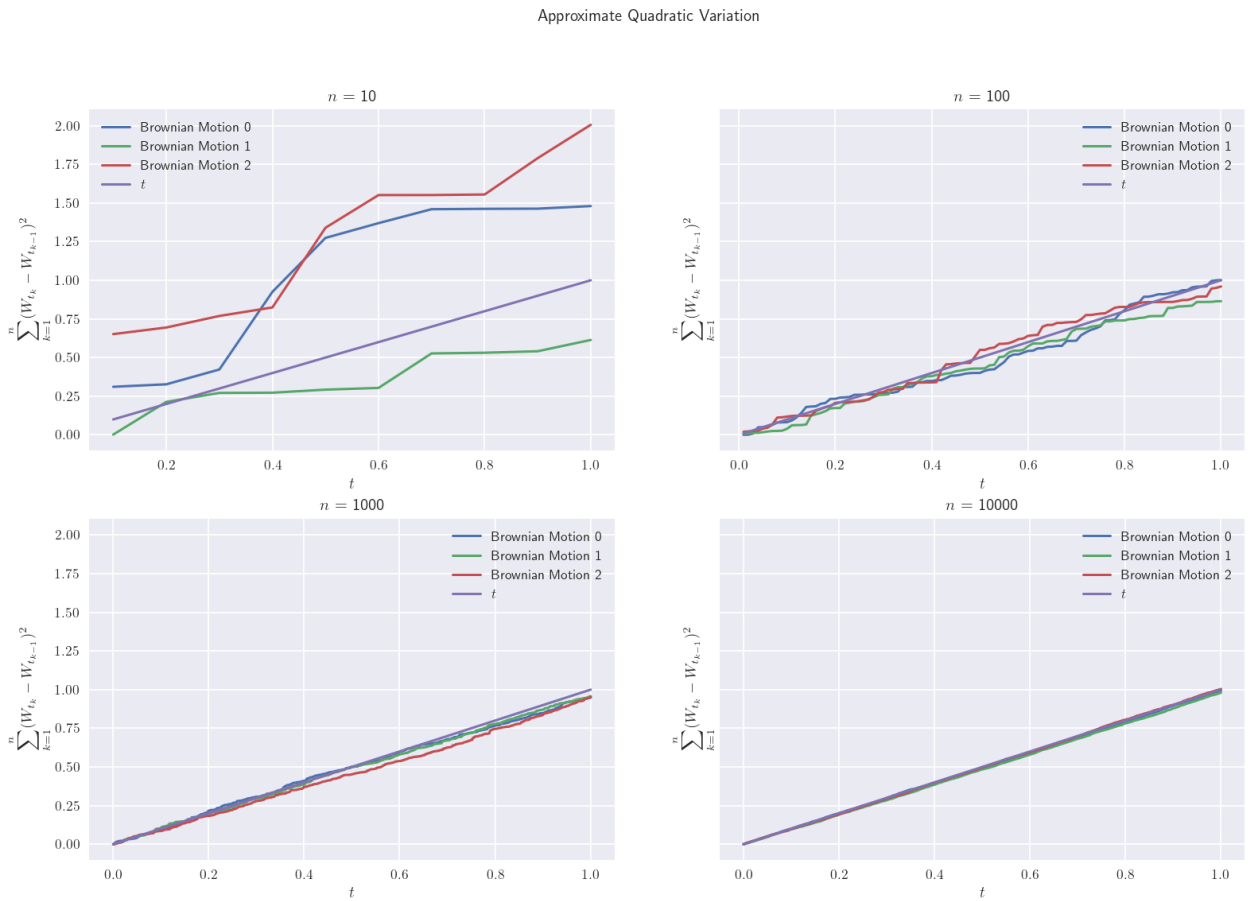
```
import numpy as np, matplotlib.pyplot as plt
from scipy.stats import norm

# Use LaTeX
plt.rcParams['text.usetex'] = True

# Use Seaborn style
plt.style.use('seaborn')

# Set the random seed
np.random.seed(0)

# Define n-values
n_vals = [10, 100, 1000, 10000]

# Set up subplots
fig, ax = plt.subplots(2, 2, sharey = True, figsize = (15, 10), dpi = 125)

# Simulate five Brownian motions
for i, n in enumerate(n_vals):

    # Get row and column
    row, col = i//2, i % 2

    # Difference of two Brownian motions is normal
    dW = norm.rvs(scale = np.sqrt(1/n), size = (3, n))

    # Calculate quadratic variance
    quad_var = np.cumsum(dW**2, axis = 1)
```

Approximate Quadratic Variation Python Code

```
    # Plot results
    for j in range(3):
        ax[row, col].plot(np.linspace(1/n, 1, n), quad_var[j, :], label = f'Brownian
      Motion {j}')
    # Plot t
    ax[row, col].plot(np.linspace(1/n, 1, n), np.linspace(1/n, 1, n),
          label = r'$t$')
    # Add x-label
    ax[row, col].set_xlabel(r'$t$')
    # Add y-label
    ax[row, col].set_ylabel(r'$\displaystyle\sum_{k = 1}^{n} (W_{t_k} - W_{t_{k - 1}})^2$
      ')
    # Give plot a title
    ax[row, col].set_title(f'$n$ = {n}')
    # Add legend
    ax[row, col].legend()
# Add title to plot
plt.suptitle(r'Approximate Quadratic Variation')
# Save the figure
plt.savefig(path + r'ex4-6.png')
plt.show()
```

Approximate Quadratic Variation Result

From the four subplots, we see that $\lim_{n\to\infty} \sum_{k=1}^{n} \left(W_{t_k} - W_{t_{k-1}}\right)^2 = T.$



Itô Integrals

Definition

The **Itô Integral** of X_t with respect to standard Brownian motion

$$\int_0^T X_t \, dW_t = \lim_{\|P\| \to 0} \sum_{k=1}^{n} X_{t_{k-1}} (W_{t_k} - W_{t_{k-1}}),$$

where $\mathcal{P} = (t_0, t_1, \ldots, t_n)$ is an arbitrary partition of $[0, T]$.

Itô Integrals Example

Example

Assuming the Itô integral exists, compute $\int_0^T W_t \, dW_t$.

Solution. Consider the uniform partition $\Delta t = T/n$ and $t_k = k\Delta t$. Then

$$\begin{aligned}
\int_0^T W_t \, dW_t &= \lim_{n\to\infty} \sum_{k=1}^{n} W_{t_{k-1}} \left(W_{t_k} - W_{t_{k-1}}\right) \\
&= \lim_{n\to\infty} \frac{1}{2} \sum_{k=1}^{n} \left[W_{t_k}^2 - W_{t_{k-1}}^2 - (W_{t_k} - W_{t_{k-1}})^2\right] \\
&= \frac{1}{2}(W_T^2 - W_0^2) - \frac{1}{2} \lim_{n\to\infty} \sum_{k=1}^{n} (W_{t_k} - W_{t_{k-1}})^2 \\
&= \frac{1}{2}(W_T^2 - W_0^2) - \frac{1}{2} T \\
&= \frac{1}{2} W_T^2 - \frac{1}{2} T.
\end{aligned}$$

Law of Iterated Expectations

Suppose that $s \leq t \leq T$. Then

$$E[X_T|\mathcal{F}_s] = E\Big[E[X_T|\mathcal{F}_t]\Big|\mathcal{F}_s\Big].$$

Instead of writing $E[X_T|\mathcal{F}_t]$ we often write $E_t[X_T]$. Using this notation, the Law of Iterated Expectations is

$$E_s[X_T] = E_s\Big[E_t[X_T]\Big].$$

Itô Isometry

Theorem (Itô Isometry)

We have

$$E\left[\left(\int_0^T X_t \, dW_t\right)^2\right] = E\left[\int_0^T X_t^2 \, dt\right]$$

whenever

$$E\left[\int_0^T X_t^2 \, dt\right] < \infty.$$