BROWNIAN MOTION

Definition (Brownian Motion)

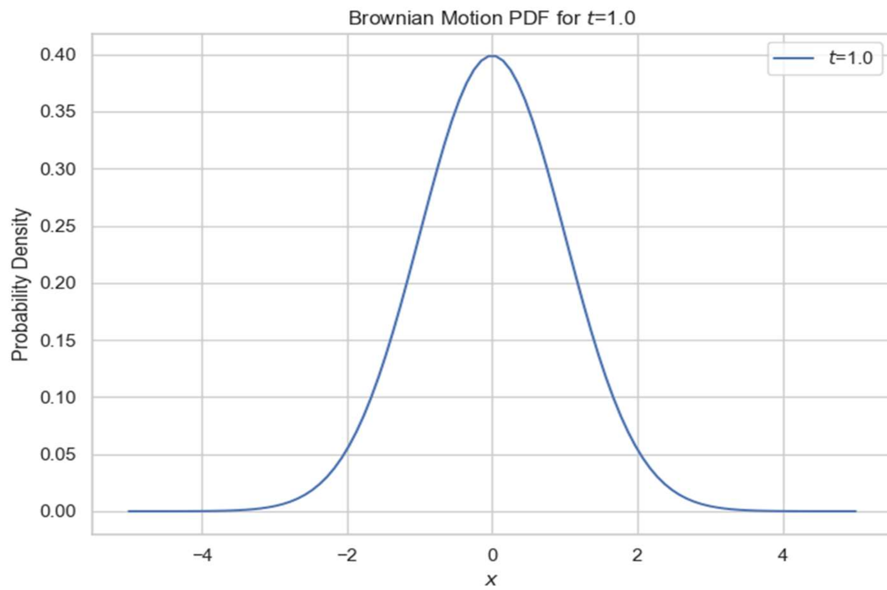
A d-dimensional Brownian motion starting at the origin is a stochastic process $\{B_t\}_{t\geq 0}$ with the following properties:
(a) if $0 \leq t_0 \leq t_1 \leq \cdots \leq t_n$ then $B(t_1) - B(t_0), B(t_2) - B(t_1), \ldots, B(t_n) - B(t_{n-1})$ are independent.
(b) if $0 \leq s \leq t$ then $B(t) - B(s) \sim N(0, (t-s))$
(c) $\mathbb{P}(\{\omega \in \Omega : B(\omega, 0) = 0 \text{ and } t \mapsto B(\omega, t) \text{ is continous }\}) = 1$

The process $x + B$ is a Brownian motion starting at $x \in \mathbb{R}^d$.

The definition implies that each marginal distribution B_t is normally distribution with $E[B_t] = 0$ and $Var[B_t] = t$. So we can simulate it using norm on python

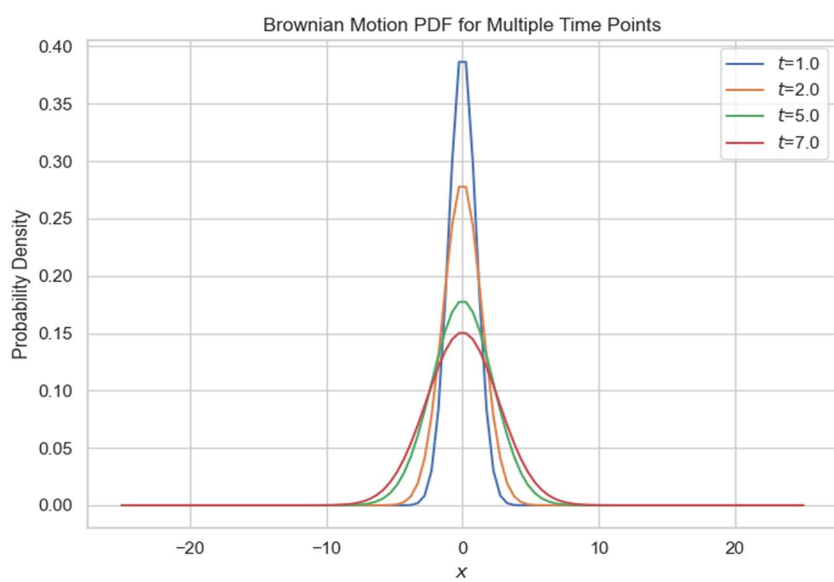
We will first start by illustrating the distrubtion for B_1

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

# Set the style for seaborn
sns.set(style="whitegrid")

# Function to visualize Brownian Motion PDF for a single time point
def plot_single_brownian_pdf(a, t):
    # Create a larger figure
    plt.figure(figsize=(8, 6))  # Adjust the size as needed
    x_values = np.linspace(-a, a, 100)  # Increased resolution
    bt_dist = norm(loc=0, scale=np.sqrt(t))

    plt.plot(x_values, bt_dist.pdf(x_values), label=f'$t$={t:.1f}')
    plt.title(f'Brownian Motion PDF for $t$={t:.1f}')
    plt.xlabel('$x$')
    plt.ylabel('Probability Density')
    plt.legend()

    # Adjust margins
    plt.subplots_adjust(left=0.15, right=0.95, top=0.9, bottom=0.15)

    plt.show()

# Call the function
plot_single_brownian_pdf(5, 1)
```



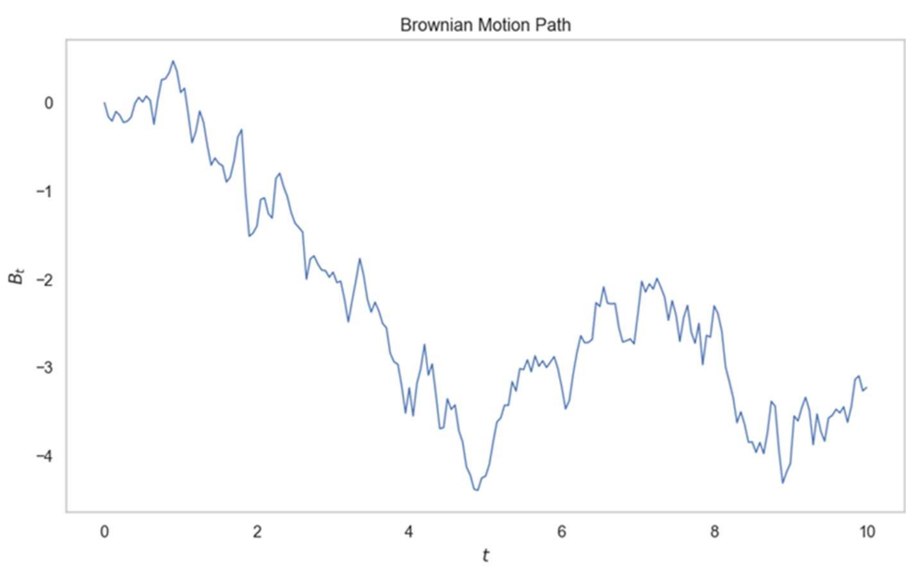
Now we will attempt to do it for multiple t

```
# Call the function with multiple time points
plot_brownian_pdfs(25, [1, 2, 5, 7])
```



PATHS SIMULATION

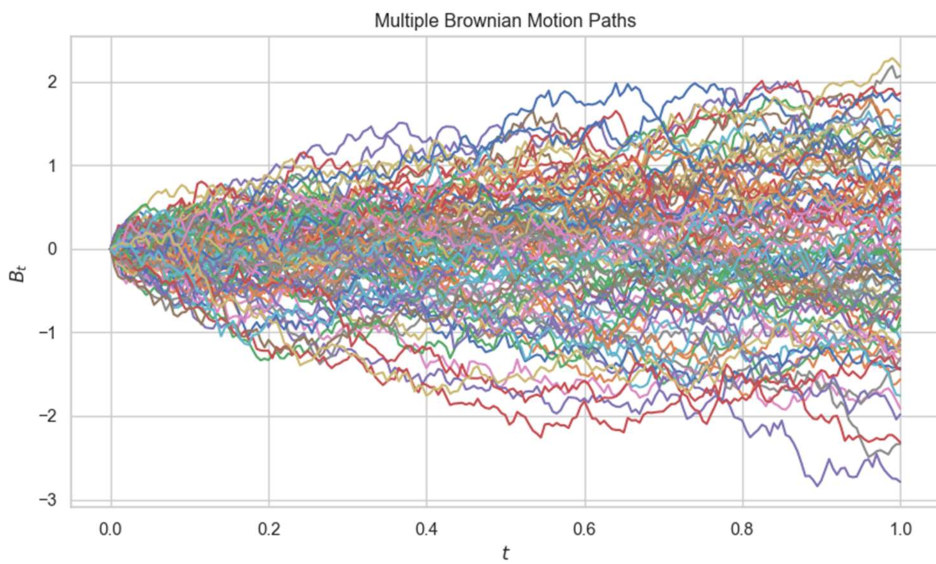
```
# Function to generate a time grid
def generate_time_grid(start=0.0, end=1.0, num_steps=30):
    delta_t = (end - start) / num_steps
    time_points = np.arange(start, end + delta_t, delta_t)
    return time_points

# Generate time points
time_series = generate_time_grid(start=0, end=10, num_steps=200)
num_points = len(time_series)
time_increment = (time_series[num_points - 1] - time_series[0]) / num_points

# Simulate increments of Brownian motion
increments = norm.rvs(loc=0, scale=np.sqrt(time_increment), size=num_points - 1)
increments = np.insert(increments, 0, 0)  # Initial condition B_0 = 0
brownian_path = increments.cumsum()

# Plot the Brownian motion path
plt.figure(figsize=(10, 6))  # Set figure size
plt.plot(time_series, brownian_path, '-', lw=1)
plt.title('Brownian Motion Path')
plt.xlabel('$t$', fontsize=12)
plt.ylabel('$B_t$', fontsize=12)
plt.subplots_adjust(left=0.15, right=0.95, top=0.9, bottom=0.15)  # Adjust margins
plt.grid()  # Optional: Add grid for better readability
plt.show()
```



Now we shall attempt to simulate multiple ones.

```
# Function to generate a time grid
def generate_time_grid(start=0.0, end=1.0, num_steps=30):
    delta_t = (end - start) / num_steps
    time_points = np.arange(start, end + delta_t, delta_t)
```

```python
    return time_points

# Generating a Brownian motion path
def simulate_brownian_motion(time_series, initial_value=0.0):
    n = len(time_series)
    delta_t = (time_series[-1] - time_series[0]) / n
    increments = norm.rvs(loc=0, scale=np.sqrt(delta_t), size=n-1)
    increments = np.insert(increments, 0, initial_value)  # Starting point
    return increments.cumsum()

# Create a larger figure with subplots
fig, axs = plt.subplots(2, 1, figsize=(10, 12))

# Plotting multiple Brownian motion paths
time_series = generate_time_grid(start=0, end=1, num_steps=200)
for _ in range(100):
    path = simulate_brownian_motion(time_series)
    axs[0].plot(time_series, path, lw=1.5)

axs[0].set_title('Multiple Brownian Motion Paths')
axs[0].set_xlabel('$t$')
axs[0].set_ylabel('$B_t$')
plt.show()
```

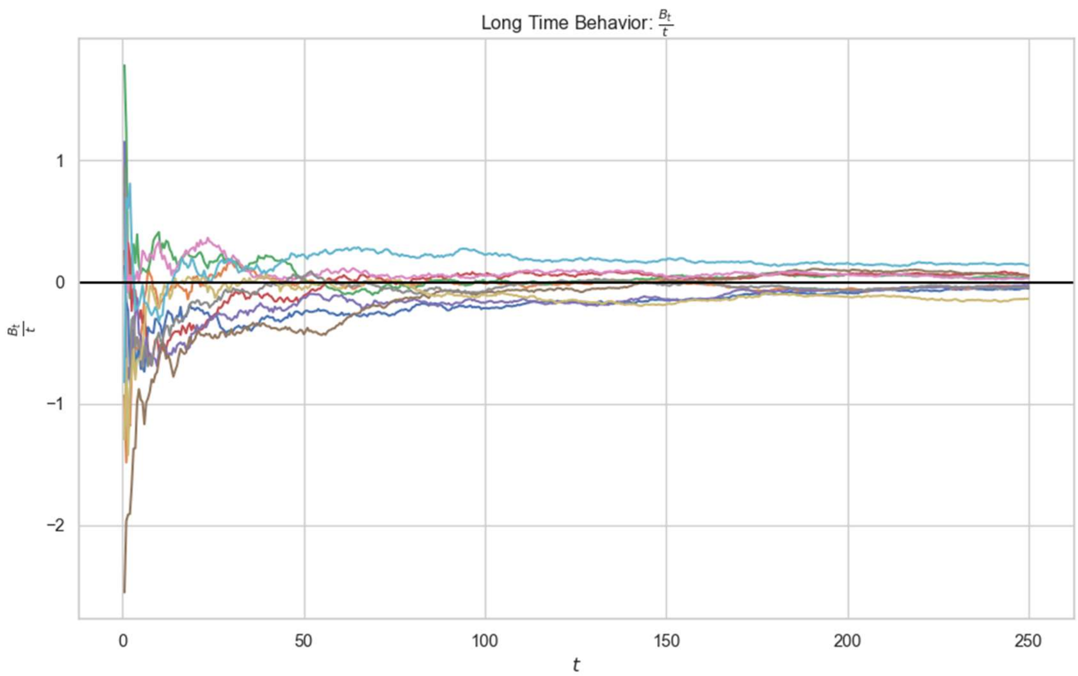


Long Time Behavior (Brownian Motion)

the Law of Large numbers gives, almost surely, that $\lim_{t\to\infty} \frac{B_t}{t} = 0$

```python
# Long time behavior of Brownian motion
long_time_series = generate_time_grid(end=250, num_steps=500)
for _ in range(10):
    long_path = simulate_brownian_motion(long_time_series)
    axs[1].plot(long_time_series[1:], long_path[1:] / long_time_series[1:], lw=1.5)
```

```
axs[1].axhline(y=0, lw=1.5, color='black')
axs[1].set_title('Long Time Behavior: $\\frac{B_t}{t}$')  # Escaped dollar sign
axs[1].set_xlabel('$t$')
axs[1].set_ylabel('$\\frac{B_t}{t}$')
# Adjust layout to prevent cut-off labels
plt.tight_layout()
plt.show()
```



Reflection Principle (Brownian Motion)

If B_t is a Brownian Motion and a> 0 then

$$P\left(\sup_{0\leq t\leq t} B_S \geq a\right) = 2P(B_t \geq a)$$

We will attempt to visualize this principle using python:

```
# Reflection principle demonstration
np.random.seed(5678)
reflection_time_series = generate_time_grid(end=10, num_steps=700)
reflection_path = simulate_brownian_motion(reflection_time_series)

threshold = 1.5
first_hit = np.where(np.isclose(reflection_path, threshold, rtol=0.01))[0][0]

plt.figure(figsize=(12, 8))
plt.plot(reflection_time_series, reflection_path, lw=1.5, color='lightcoral', label="$B
_t$")
plt.plot(reflection_time_series[first_hit:], -reflection_path[first_hit:] + 2 * thresho
ld, lw=1.5, color='lightblue', label='Reflection')
plt.axhline(y=threshold, lw=1, color='darkred')
plt.title('Reflection Principle in Brownian Motion')
plt.legend()
```

```
plt.tight_layout()  # Adjust layout to prevent cut-off labels
plt.show()
```