The functional CLT (Donsker's invariance principle): proof and simulations

Pietro Colaguori 1936709

November 2023

1 Proof

Donsker's Invariance Principle, also known as the Functional Central Limit Theorem (CLT), is a mathematical result in probability theory that establishes a convergence result for certain types of stochastic processes to a particular limiting process. This principle is named after the mathematician Michael Donsker.

Intuitively this principle tells us that, under certain conditions, the empirical distribution function (based on a sample) behaves like a random process that converges to a Brownian bridge as the sample size increases. This result provides a powerful tool for understanding the asymptotic behavior of statistical estimators and testing procedures.

Let $X_1, X_2, ...$ be a sequence of i.i.d. random variables with mean 0 and variance 1. Let

$$S_n = \sum_{i=1}^{n} X_i$$

The stochastic process $S = (S_n)_{n \in N}$ is a random walk. We can now define the rescaled random walk as:

$$W^{(n)}(t) = \frac{S_{[nt]}}{\sqrt{n}}, t \in [0, 1]$$

By CLT, $W^{(n)}(1)$ converges to a standard Gaussian random variable as $n \to \infty$. Donsker's invariance principle extends this statement to the whole function $W^{(n)}$. Claiming that the sequence of empirical processes converges to a standard Brownian bridge $B(x)$.

In practical terms, Donsker's Invariance Principle helps researchers and statisticians analyze the behavior of statistical processes, providing insights into the convergence properties of empirical distributions to continuous-time stochastic processes.

2 Simulations

I created the following Python script, it generates random samples, calculates and plots an empirical distribution function, then we can see that as we increase the value of the sample size n the S_n converges to a Brownian motion.

```python
import numpy as np
import streamlit as st
import matplotlib.pyplot as plt

# number of processes
mean = 0
variance = 1
n = 10000

# Generate n i.i.d. variables
X_i = np.random.normal(mean, variance, n)

# Compute the sum
S_n = np.zeros(n)
for i in range(n):
    S_n[i] = np.sum(X_i[:i+1]) / np.sqrt(i+1)

# Plot the graph
st.line_chart(S_n)
```

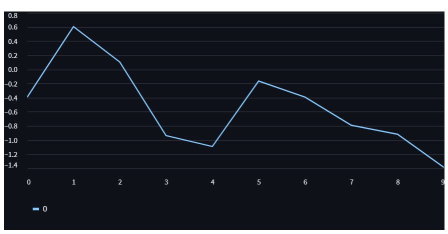

Figure 1: Simulation run with $n = 10$.

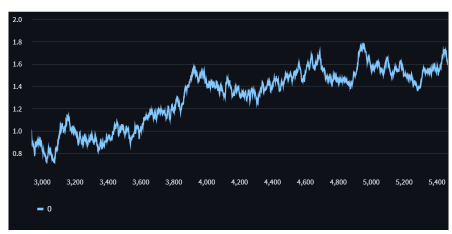

Figure 2: Simulation run with $n = 10000$.