Simple models of diffusion - Random walk

Exercise Report: 1D Random Walk

(Whole code at the end of the report)

Introduction

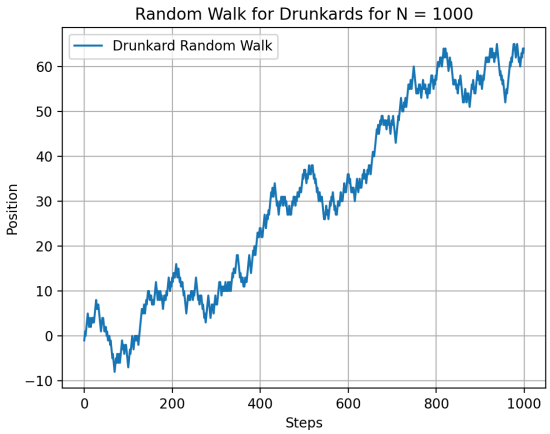
We have drunk sailer that starts on x = 0 position and randomly choose to what direction he is going to go. He take steps of length = 1 and he loses all memory between any single steps.

We can simulate this situation by provided code:

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

def random_walk():
  N = 1000  # Number of steps

  position = np.cumsum(np.random.choice([-1, 1], size=N))

  plt.plot(position, label='Drunkard Random Walk')
  plt.xlabel('Steps')
  plt.ylabel('Position')
  plt.title(f'Random Walk for Drunkards for N = {N}')
  plt.grid(True)
  plt.legend()
  plt.show()

if __name__ == "__main__":
  random_walk()
```

(Python 3.11.7)

When we run this code, we are going to get following result:



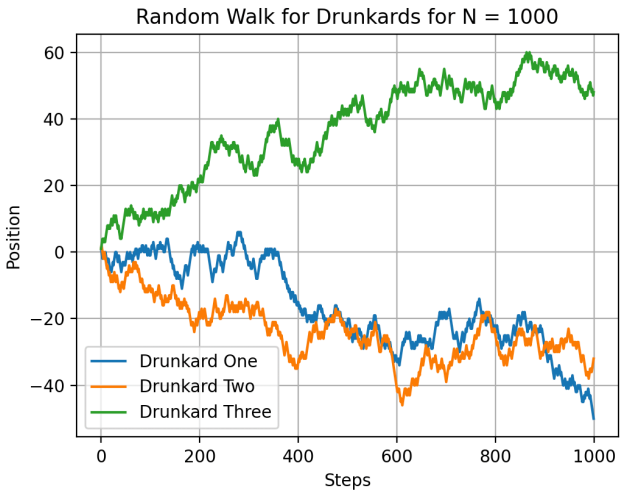
We can see position changing over time, where:

```
position = np.cumsum(np.random.choice([-1, 1], size=N))

# position[t] = Σ (random step_i) for i = 0 to t, where t ranges from 0 to N-1.
```

(Python 3.11.7)

1) Graf 1D random walk in 1000 steps.

By performing this function n-times, we can showcase the differences between each drunkard position:



2) Graf showcasing differences between random walk for n = 3 drunkards for N = 1000 steps.

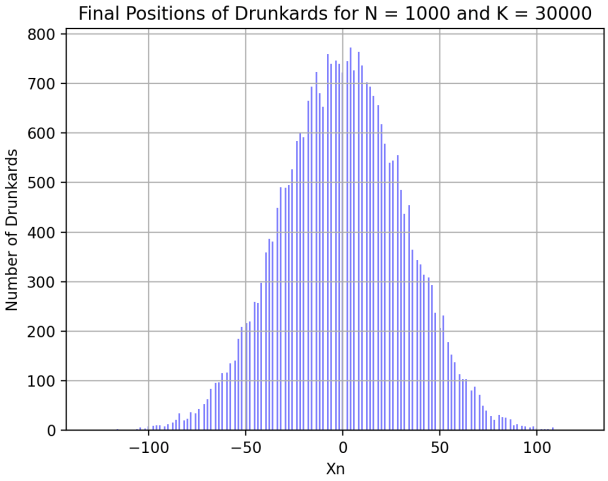
What we can see is that every random walk is different in some way.

What if we simulate walk for Drunkards >> 1000?

So, we set K - number of drunkards (K = 30 000) and N - number of steps (N = 1000). We are going to perform function that return us $a_n = a_1 + \ (n-1)d$ position of the K-th drunkard.

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

def random_walk(N, K):
    # Simulate random walk for K Drunkards
    positions = np.zeros(K)
    for _ in range(N):
        steps = np.random.choice([-1, 1], size=K)
        positions += steps
    return positions

def main():
    N = 1000  # Number of steps
    K = 30000  # Number of Drunkards

    positions = random_walk(N, K)
    plt.hist(positions, bins=300, density=False, alpha=0.5, color='b')

    plt.title(f"Final Positions of Drunkards for N = {N} and K = {K}")
    plt.xlabel('Xn')
    plt.ylabel('Number of Drunkards')
    plt.grid(True)

    plt.show()


if __name__ == "__main__":
    main()
```
(Python 3.11.7)

And the results that we are getting are:



We can see that our distribution is very similar to Gaussian distribution (PDF):

$$p(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

3) Histogram showing as the last position for K drunkards in N steps.

What relationship is between steps taken by the drunkard and the distance covered over time?

We are going to analyze it using linear regression: y = ax + b

Variance calculation $V(X_n) = E(X_n^2) - [E(X_n)]^2$

Ex. for population of [-18, -4] we are going to get $V(X_n) = 49$.

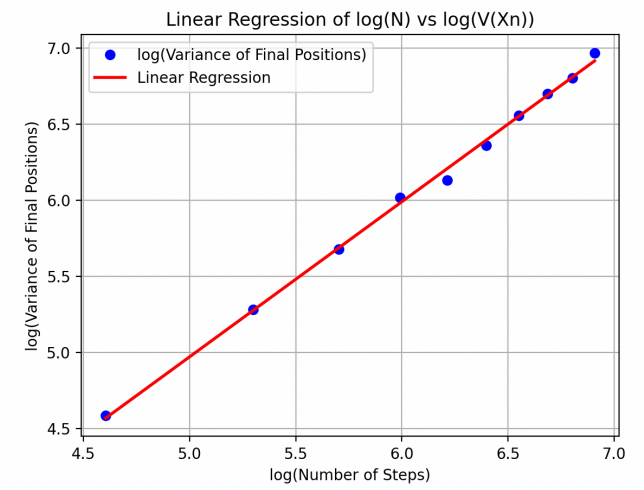
We are going present our results in log(N) and log(V(Xn)).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression


def random_walk(N, K):
    positions = np.zeros(K)
    for _ in range(N):
        steps = np.random.choice([-1, 1], size=K)
        positions += steps
    return positions


def calculate_variance(N_values, K):
    variances = []
    for N in N_values:
        positions = random_walk(N, K)
        variance = np.var(positions) # Variance calculation
        variances.append(variance)
    return np.array(variances)


def main():
    K = 1000 # Number of drunkards
    N_values = np.arange(100, 1001, 100)

    variances = calculate_variance(N_values, K)
```

```
    # Compute logarithms of N and V(Xn)
    log_N_values = np.log(N_values)
    log_variances = np.log(variances)

    # Perform linear regression on log(N) and log(V(Xn))
    log_N_values_reshaped = log_N_values.reshape(-1, 1)
    model = LinearRegression()
    model.fit(log_N_values_reshaped, log_variances)
    log_variances_pred = model.predict(log_N_values_reshaped)

    # Plot log(N) vs log(V(Xn))
    plt.plot(log_N_values, log_variances, 'bo',
             label='log(Variance of Final Positions)')
    plt.plot(log_N_values, log_variances_pred, color='red',
             linewidth=2, label='Linear Regression')

    plt.title(
        'Linear Regression of log(N) vs log(V(Xn)’)
    plt.xlabel('log(Number of Steps)')
    plt.ylabel('log(Variance of Final Positions)')
    plt.grid(True)
    plt.legend()
    plt.show()


if __name__ == "__main__":
    main()
```

```(Python 3.11.7)

# Results: Observing the Normal Diffusion




On the graph, we observe that the diffusion pattern closely resembles that of **normal diffusion**

$$\frac{\partial}{\partial t}P(x,t) = D\frac{\partial^2}{\partial x^2}P(x,t)$$

*4) Variance of the final positions changes with respect to the number of steps taken in the random walk in K = 1000*

```***WHOLE CODE***
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression


def random_walk(N, K):
    # Simulate random walk for K Drunkards
    positions = np.zeros(K)
    for _ in range(N):
        steps = np.random.choice([-1, 1], size=K)
        positions += steps
    return positions
```

```
def calculate_variance(N_values, K):
    variances = []
    for N in N_values:
        positions = random_walk(N, K)
        variance = np.var(positions)
        variances.append(variance)
    return np.array(variances)


def main():
    N = 1000  # Number of steps
    K = 30000  # Number of Drunkards

    N_values = np.arange(100, 1001, 100)

    variances = calculate_variance(N_values, K)

    # Compute logarithms of N and V(Xn)
    log_N_values = np.log(N_values)
    log_variances = np.log(variances)

    # Perform linear regression on log(N) and log(V(Xn))
    log_N_values_reshaped = log_N_values.reshape(-1, 1)
    model = LinearRegression()
    model.fit(log_N_values_reshaped, log_variances)
    log_variances_pred = model.predict(log_N_values_reshaped)

    # Perform random walk
    positions = random_walk(N, K)

    fig, axis = plt.subplots(2, 2)

    position_one = np.cumsum(np.random.choice([-1, 1], size=N))
    position_two = np.cumsum(np.random.choice([-1, 1], size=N))
    position_three = np.cumsum(np.random.choice([-1, 1], size=N))

    axis[0, 0].plot(position_one, label='Drunkard One')
    axis[0, 0].plot(position_two, label='Drunkard Two')
    axis[0, 0].plot(position_three, label='Drunkard Three')
    axis[0, 0].set_xlabel('Steps')
    axis[0, 0].set_ylabel('Position')
    axis[0, 0].set_title(f'Random Walk for Drunkards for N = {N}')
    axis[0, 0].grid(True)
    axis[0, 0].legend()

    # Create histogram with more bins
    axis[0, 1].hist(positions, bins=300, density=False, alpha=0.5, color='b')

    axis[0, 1].set_title(f"Final Positions of Drunkards for N = {N} and K = {K}")
    axis[0, 1].set_xlabel('Xn')
    axis[0, 1].set_ylabel('Number of Drunkards')
    axis[0, 1].grid(True)

    axis[1, 0].plot(log_N_values, log_variances, 'bo',
                    label='log(Variance of Final Positions)')
    axis[1, 0].plot(log_N_values, log_variances_pred, color='red',
                    linewidth=2, label='Linear Regression')

    axis[1, 0].set_title(
        'Drunkard Linear Regression')
    axis[1, 0].set_xlabel('log(Number of Steps)')
    axis[1, 0].set_ylabel('log(Variance of Final Positions)')
    axis[1, 0].grid(True)
    axis[1, 0].legend()

    fig.suptitle('N - number of steps, K - number of drunkard')

    plt.show()


if __name__ == "__main__":
    main()
```

(Python 3.11.7)