Simple models of diffusion - Random walk

Exercise Report: 2D Random Walk

(Whole code at the end of the report)

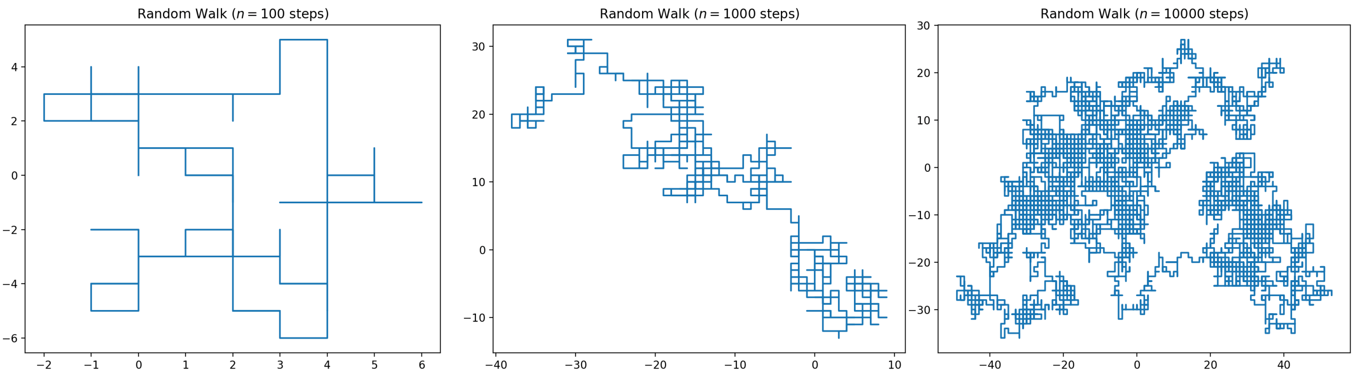
Introduction

We have drunk sailer that starts on x = (0,0) position and randomly choose to what 2D direction he is going to go, can take steps like (0,1), (0,-1), (1,0), (-1,0). He take steps of length = 1 and he loses all memory between any single steps.

We can simulate this situation by provided code:

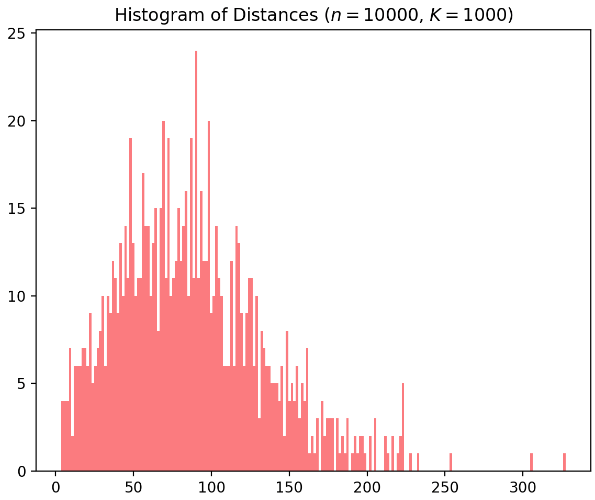
```
def random_walk(N, K):

    directions = np.array([[1, 0], [-1, 0], [0, 1], [0, -1]])
    walks = np.zeros((K, N, 2))
    for k in range(K):
        steps = directions[np.random.randint(0, 4, size=N-1)]
        walks[k, 1:] = steps.cumsum(axis=0)

    distances = np.linalg.norm(walks[:, -1, :], axis=1)
    return walks, distances
```
(Python 3.11.7)

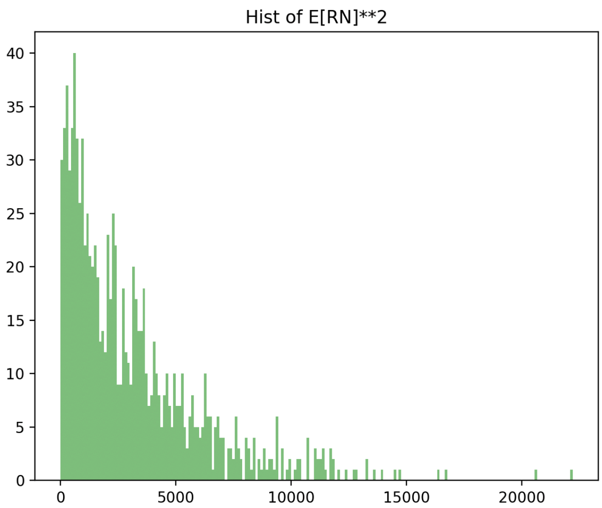
This are the result for one drunkard:



1) 2D random N = 100, N = 1000, N = 10 000

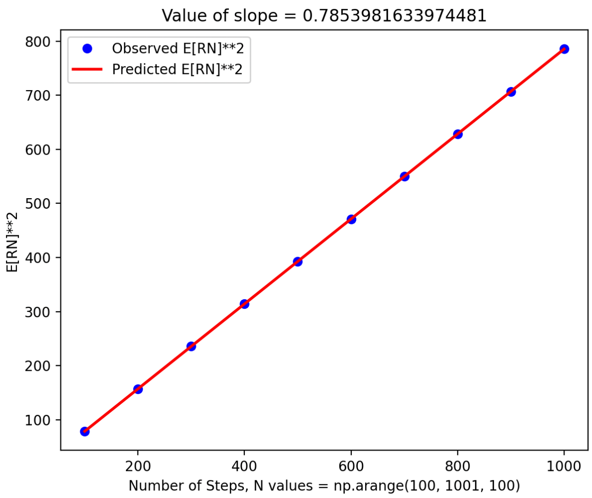
We can simulate this for K drunkards and plot $\mathbb{E}[R_N]$ where $\mathbb{E}[R_N] = \sqrt{x_N^2 + y_N^2}$, and then we perform $(R_N)^2$ and plotting the histogram with following results:





3) Histogram of $\mathbb{E}[R_N]$ and $\mathbb{E}[R_N]^2$

The linear regression for $\mathbb{E}[R_N]^2$to N steps:



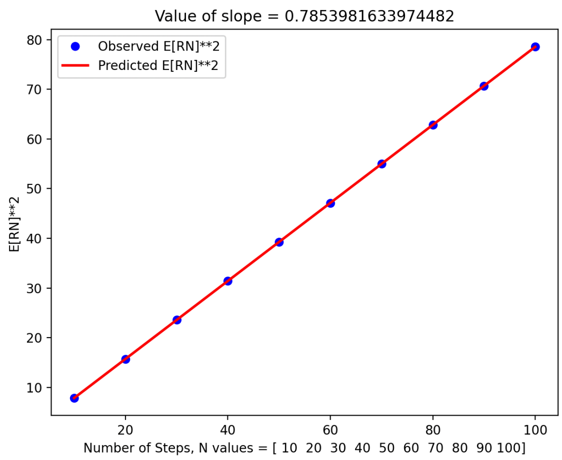
Linear regression for:

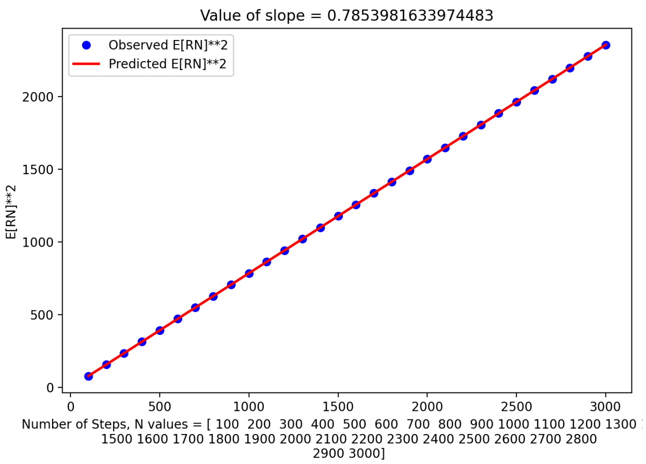
$$\mathbb{E}[R_N]^2 = \frac{\pi}{4} \times N$$

$m \approx 0.78$ *(Value of slope)*

4) Linear regression for N = {100, 200, …, 1000}

And some more examples for bigger N_step values:





4) Plot for N_values = {10,20, …, 100} and N_values = {100, 200, …, 3000}

```(WHOLE CODE)

~~~~python
import numpy as np
import matplotlib.pyplot as plt
import random
import math
from sklearn.linear_model import LinearRegression
from scipy.stats import linregress


def random_walk(N, K):
    """
    Simulates a random walk in 2D for N steps and K walkers.
    Returns the final positions and the Euclidean distances from the origin.
    """
    directions = np.array([[1, 0], [-1, 0], [0, 1], [0, -1]])
    walks = np.zeros((K, N, 2))
    for k in range(K):
        steps = directions[np.random.randint(0, 4, size=N-1)]
        walks[k, 1:] = steps.cumsum(axis=0)

    distances = np.linalg.norm(walks[:, -1, :], axis=1)
    return walks, distances


def calculate_avg_squared_dist(N_values):
    """
    Calculates the average squared distance for a set of steps.
    Theoretical calculation using (pi/4) * N.
    """
    return [(math.pi / 4) * N for N in N_values]


def main():
    # 36 GB RAM
    # N = 30000  # Number of steps
    # K = 100000  # Number of Drunkards

    N = 3000
    K = 1000

    fig, ax = plt.subplots(2, 2, figsize=(12, 10))

    # Perform random walk and plot the first one for visualization
    walks, distances = random_walk(N, K)
    ax[0, 0].plot(walks[0, :, 0], walks[0, :, 1])
    ax[0, 0].set_title("Random Walk ($n = {}$ steps)".format(N))

    # Histogram of distances
    ax[0, 1].hist(distances, bins=200, color='red', alpha=0.5)
    ax[0, 1].set_title(
        "Histogram of Distances ($n = {}$, $K = {}$)".format(N, K))

    # Prepare data for linear regression
    N_values = np.arange(100, 3001, 100)
    avg_squared_dists = calculate_avg_squared_dist(N_values)

    ax[1, 0].set_title("Hist of E[RN]**2")
    ax[1, 0].hist(np.power(distances, 2), bins=200,
                  color='g', alpha=0.5, label='X')

    # Fit linear regression
    model = LinearRegression()
    model.fit(N_values.reshape(-1, 1), avg_squared_dists)
    predictions = model.predict(N_values.reshape(-1, 1))

    slope = linregress(N_values, avg_squared_dists).slope

    # Plotting the linear regression results
    slope, _, _, _, _ = linregress(N_values, avg_squared_dists)
    ax[1, 1].plot(N_values, avg_squared_dists, 'bo', label="Observed E[RN]**2")
    ax[1, 1].plot(N_values, predictions, 'r-',
                  linewidth=2, label="Predicted E[RN]**2")
    ax[1, 1].set_title(
        f"Value of slope = {slope}")
    ax[1, 1].set_xlabel(
        f'Number of Steps, N values = {N_values}')
    ax[1, 1].set_ylabel('E[RN]**2')
    ax[1, 1].legend()

    plt.tight_layout()
    plt.show()


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

```Python 3.11.7)