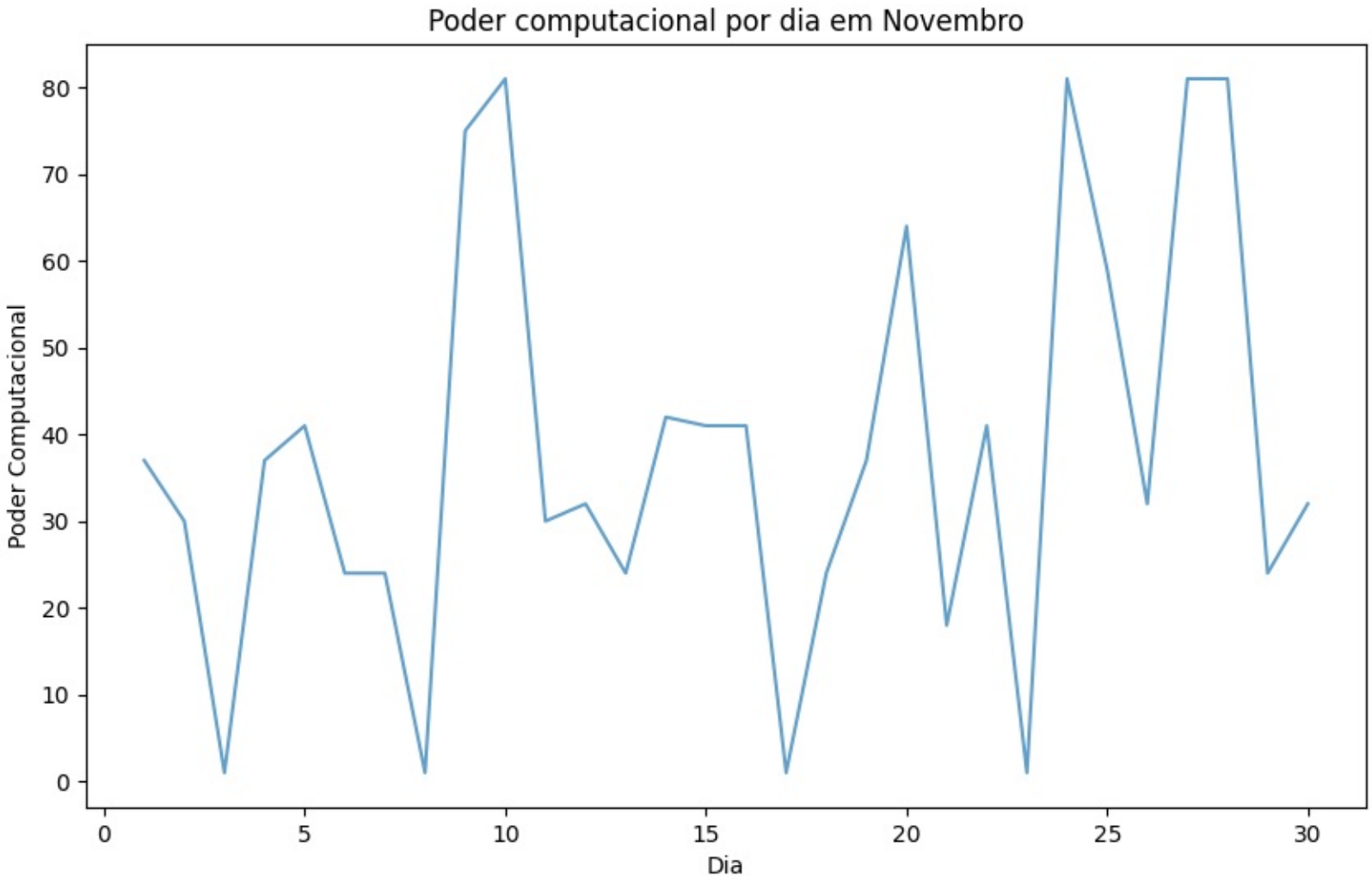
```python
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from itertools import groupby
from scipy import stats

# use: pip install scipy

# Constantes
BLOCK = "block_integer_array.npy"
SIGNIFICANCE_LEVEL = 0.05


def load_data(filename):
    """Carrega os dados do arquivo especificado."""
    try:
        return np.load(filename)
    except FileNotFoundError:
        print(f"Erro: O arquivo {filename} não foi encontrado.")
        return None


def split_into_months(data_array):
    """Divide o array de dados em 12 partes, uma para cada mês. Porém, se o número
 de elementos não for divisível por 12, seria necessário adaptar essa função para
redimensionar os dados para o maior múltiplo de 12."""

    trunc_len = (len(data_array) // 360) * 360
    return data_array[:trunc_len].reshape(-1, 30)
    # return np.array_split(data_array, 12)


# def get_month(monthly_data, month):
#     return monthly_data[month - 1]  # 0-indexed


# def plot_data(monthly_data):
#     plt.hist(monthly_data, bins=30)
#     plt.title("Poder computacional por dia em Novembro")
#     plt.show()


def plot_data(monthly_data):
    # Extrai os dias e o poder computacional
    days = range(1, len(monthly_data) + 1)
    power = monthly_data

    plt.figure(figsize=(6, 4))
    plt.plot(days, power, alpha=0.7)
    plt.title("Poder computacional por dia em Novembro")
    plt.xlabel("Dia")
    plt.ylabel("Poder Computacional")
    plt.show()
```

```python
def get_most_powerful_miner(monthly_data):
    """Retorna o minerador mais poderoso com base nos dados mensais fornecidos, a partir do primeiro elemento da primeira tupla (mais comum) da lista de tuplas retornada pelo método most_common do objeto Counter."""
    counter = Counter(monthly_data)
    return counter.most_common(1)[0][0]


def count_sequences(month_data, miner):
    """Conta o número de sequências consecutivas de mineração realizadas pelo minerador especificado (mp). O método groupby do módulo itertools é usado para agrupar os elementos consecutivos do array de dados"""
    return [sum(1 for _ in group) for key, group in groupby(month_data) if key == miner]


def generate_permutations(data, miner, num_permutations=1000):
    """Gera permutações aleatórias dos dados e conta o número de sequências consecutivas de mineração realizadas pelo mp em cada permutação."""
    perm_sequences = []
    for _ in range(num_permutations):
        perm = np.random.permutation(data)
        perm_sequences.append(count_sequences(perm, miner))
    return perm_sequences


def calc_pvalue(sequences, perm_sequences):
    """Calcula o p-value para a sequência de mineração do minerador mais poderoso com base nas permutações geradas e retorna o percentil do valor máximo da sequência de mineração do mp em relação às permutações."""
    return stats.percentileofscore([max(seq) for seq in perm_sequences], max(sequences))


def print_conclusion(pvalue, significance_level):
    pvalue_rounded = round(pvalue, 2)
    print(f"p-value: {pvalue_rounded}")
    if pvalue < significance_level:
        print(f"Mineração egoísta provavelmente ocorreu (p-value: {pvalue_rounded} < {significance_level}).\nA sequência de minerações do minerador mais poderoso é significativamente mais longa do que o esperado por acaso, isto é, diferente do que seria esperado se hipótese nula fosse verdadeira.")
    else:
        print(f"Mineração egoísta provavelmente não ocorreu (p-value: {pvalue_rounded} >= {significance_level}).\nA sequência de minerações do minerador mais poderoso não é significativamente\nmais longa do que o esperado por acaso, isto é, não é diferente do que seria esperado se hipótese nula fosse verdadeira.")
```

```
def main():
    data_array = load_data(BLOCK)
    monthly_data = split_into_months(data_array)
    # november = get_month(monthly_data, 11)
    november = monthly_data[10]  # 0-indexed
    plot_data(november)
    mp = get_most_powerful_miner(november)
    sequences = count_sequences(november, mp)
    perm_sequences = generate_permutations(november, mp)
    pvalue = calc_pvalue(sequences, perm_sequences)
    print_conclusion(pvalue, SIGNIFICANCE_LEVEL)


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

```
selfish-mining> python selfmin.py
p-value: 69.4
Mineração egoísta provavelmente não ocorreu (p-value: 69.4 >= 0.05).
A sequência de minerações do minerador mais poderoso não é significativamente
mais longa do que o esperado por acaso, isto é, não é diferente do que seria esperado se hipótese nula fosse verdadeira.
selfish-mining>
```


Poder computacional por dia em Novembro
Poder Computacional
Dia
0
10
20
30
40
50
60
70
80
0
5
10
15
20
25
30