Approximation HomeWork 4

Michael Kovaliov

19 6 2021

Home Work assignment 4

Exercise 1

Let G be a group that acts on M, let the stabilizer of $m \in M$ be G_m. Let $a \in G$, prove that $G_{am} = aG_ma^{-1}$.

Proof:

Let $g \in G_{am}$. Then we know that $gam = gm$. Then we have:

$gam = gm \iff a^{-1}ga \in G_m \iff (aga^{-1} \in G_{am} \iff g \in G_m) \rightarrow G_{am} = aG_ma^{-1}$

and we are done.

Exercise 2-5

All the code is in github: https://github.com/RedPenguin100/MRA-problem. If you have any problem accessing the code, please contact me at michaelkovaliov97@gmail.com

Exercise 2

I've optimized the code a bit, in the end this is the code:

```
import numpy as np
import scipy.linalg

L = 3 # Arbitrarily, not in the original code

def generate_data(x: np.array, N, sigma):
    """
    :param x: signal
    :param N: measurements amount
    :param sigma: standard deviation
    :return:
    """
    global INDEXES
    L = x.size
    group_members = np.random.randint(1, L, N)

    noise = np.random.normal(scale=sigma, size=(N, L))
    data = np.zeros((N, L))
    binomial = np.random.binomial(1, 0.5, N)
    one = binomial == 1
    zero = binomial == 0
    data[one] = x.take(INDEXES[group_members[one]]) + noise[one]
```

```
    data[zero] = x + noise[zero]

    return data
# some code.........
# L is defined somewhere
INDEXES = np.ones((L, L), dtype='int')
for i in range(L):
    INDEXES[i] = np.roll(np.arange(L), i)
```

The fact that we pull the indexes from the matrix and not move the `x` vector by `np.rolling`, the code is significantly faster.

Also the code is in its more final version, so we have the un-uniform distribution here.

Exercise 3

The mean is basically just mean:

```
def signal_mean_from_data(data):
    """
    :note: The mean of each row separately, and then
    averaging the results, is the same result as averaging
    everything together.
    """
    return np.mean(data)
```

Because the mean of the rows, and then taking the mean of the columns, is basically just the mean.

If by mean you meant the first moment, I have a function that does that as well:

```
def signal_first_moment(data):
    return np.mean(data, axis=0)
```

Which is also just calculating mean, but instead of calculating a single number, it gives a vector which estimates the convolution of the probability distribution with the original signal.

The power spectrum computation is given in the following code:

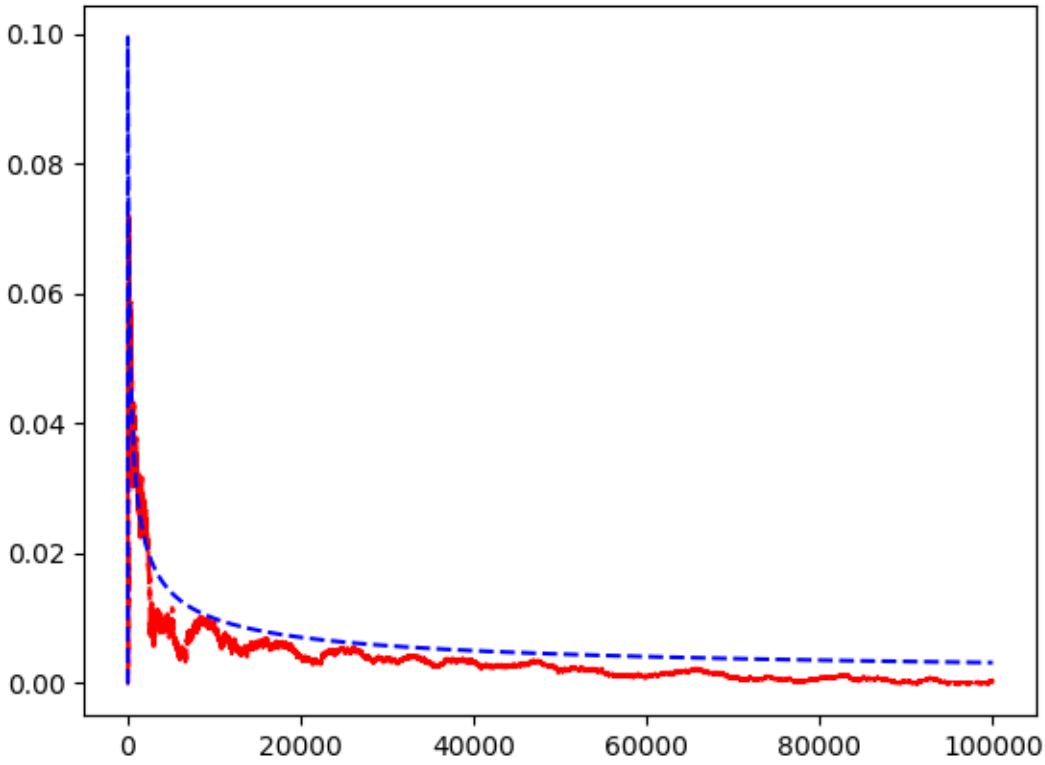
```
def signal_power_spectrum_from_data(data, sigma):
    N, L = data.shape
    power_spectra = np.power(np.abs(np.fft.fft(data)), 2.0)
    return np.mean(power_spectra, axis=0) - np.power(sigma, 2.0) * L
```

Instead of computing the conjugate I just used `abs` directly, I don't see any difference in the methods and I thought it would be more elegant to write this way.

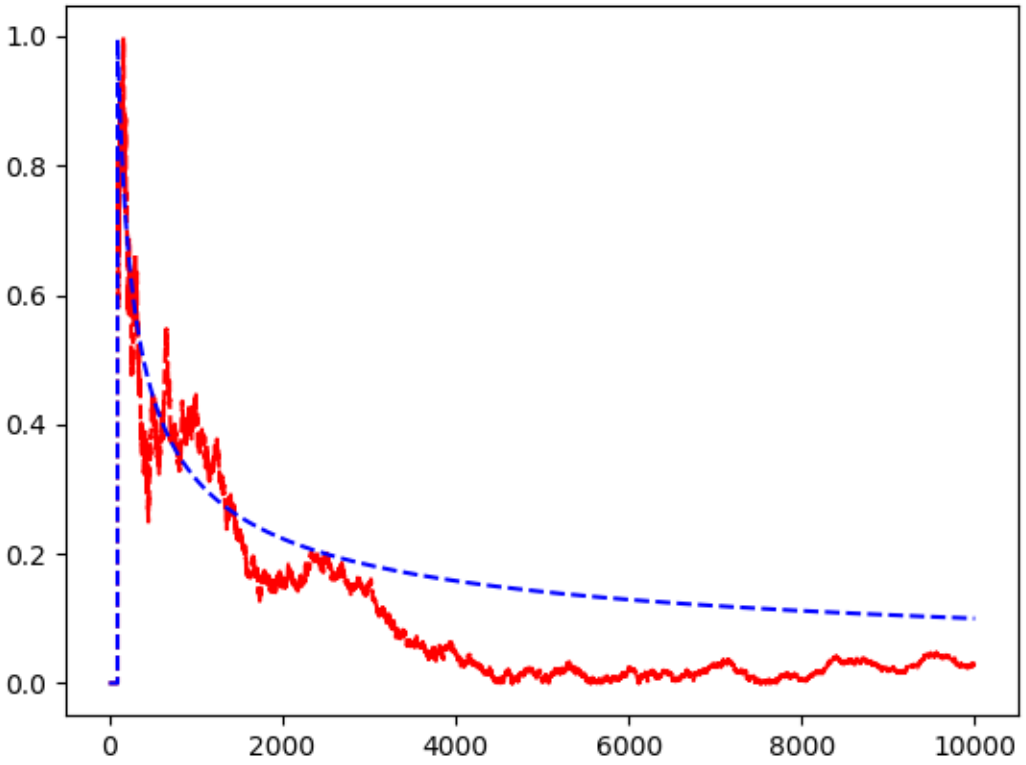
For signal $x = (1, 2, 3)$:

Proof of convergence of the mean:

For $N = 100,000, \sigma = 1$



For $N = 10,000, \sigma = 10$

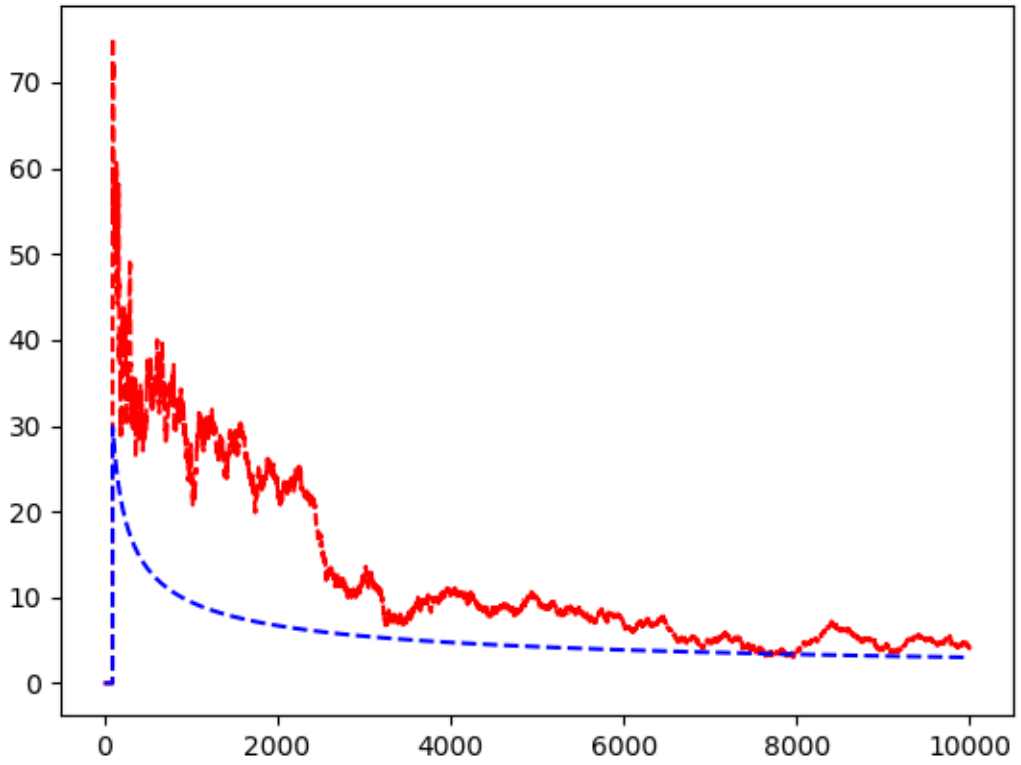


Notes:

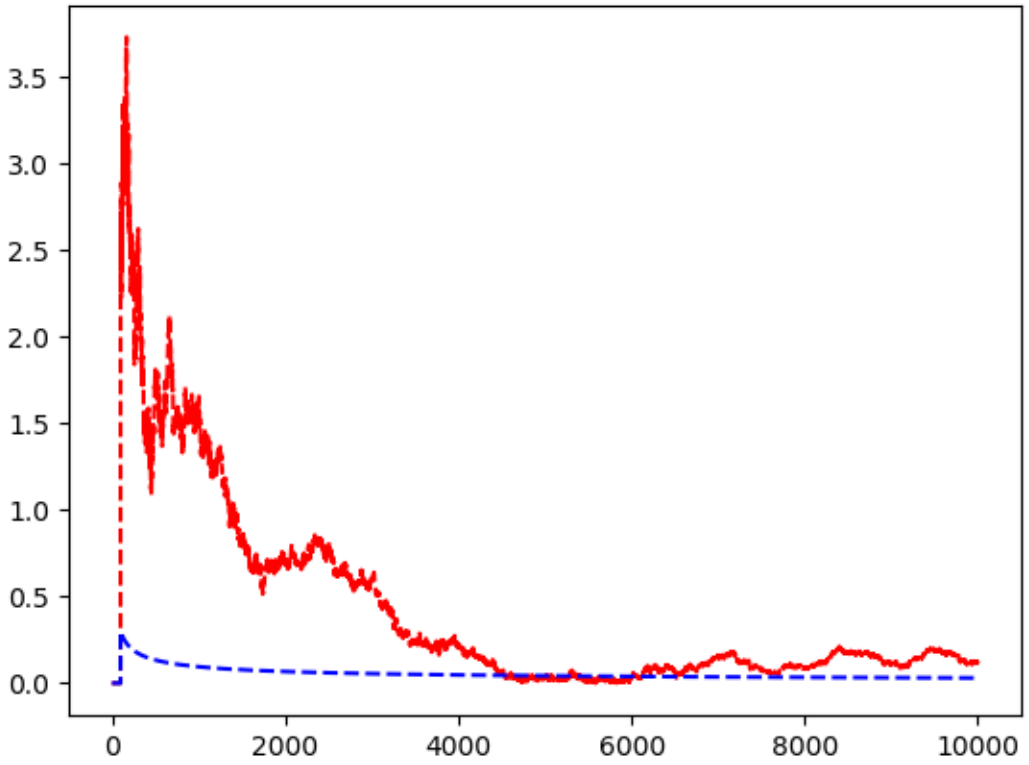
- The blue line is $\frac{\sigma}{\sqrt{N}}$
- I've set the first 100 points to 0, so we can see the graphs more precisely.

Proof of convergence of the power spectrum:

For $N = 10,000, \sigma = 10$



For $N = 10,000, \sigma = 1$



A few notes:

- The blue line is $L\frac{\sigma^2}{\sqrt{N}}$
- The first 100 points are set to zero so we can see more precisely.
- The error did not converge nice enough, I'm note sure why.

Exercise 4

The code:

```
def signal_observation_error(signal, observation):
    L = signal.shape[0]
    assert L == observation.shape[0]
    error = np.inf
    for i in range(L):
        new_error = np.linalg.norm(signal - observation.take(INDEXES[i]))
        if new_error < error:
            error = new_error

    return error
```

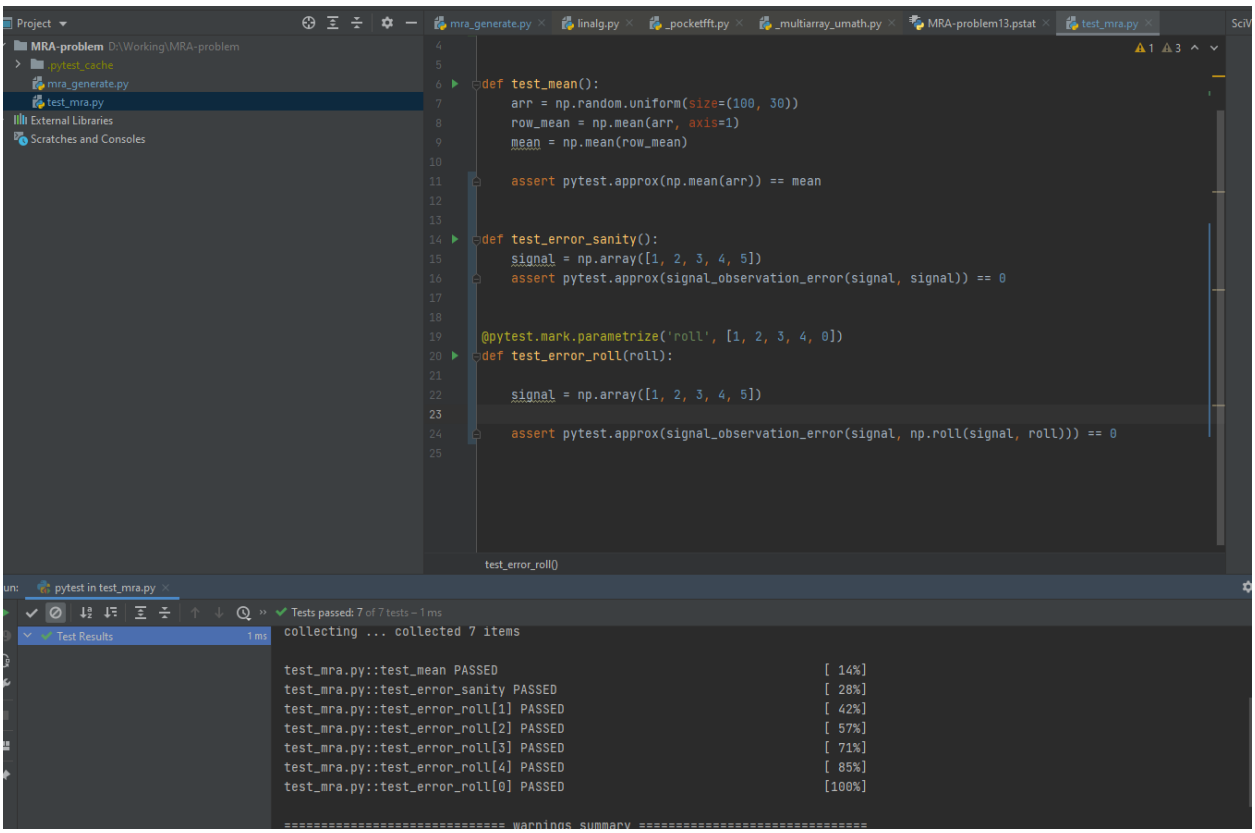
Proof it works:

```
import pytest
```

```python
def test_error_sanity():
    signal = np.array([1, 2, 3, 4, 5])
    assert pytest.approx(signal_observation_error(signal, signal)) == 0


@pytest.mark.parametrize('roll', [1, 2, 3, 4, 0])
def test_error_roll(roll):
    signal = np.array([1, 2, 3, 4, 5])
    assert pytest.approx(signal_observation_error(signal, np.roll(signal, roll))) == 0
```

This passed on my machine:



For this to pass on yours, the `INDEXES` variable needs to be defined in the `mra_generator.py` file, with the correct L(which is 5 in the tests).

Exercise 5

Well, the rest of the code is responsible for that.

The main part of it:

```python
def estimate_signal_from_data(data, sigma):
    mu_1 = signal_first_moment(data)
    mu_2 = signal_mu_2_from_data(data)
    P_x = signal_power_spectrum_from_data(data, sigma)
    return estimate_signal(mu_1, mu_2, sigma, P_x)


def estimate_signal(mu_1, mu_2, sigma, P_x):
```

```
    """
    :param mu_1: Estimation of the average of the signal
    :param mu_2: Estimation of the outer product
    :param sigma: standard deviation of our data
    :param P_x: power spectrum vector
    """
    L = len(P_x)
    # Abs not necessary, just to prevent weird output when sigma is high
    P_x = np.abs(P_x)
    v_p = np.sqrt(np.diag(1 / P_x))
    W = (1 / np.sqrt(L)) * scipy.linalg.dft(L)
    Q = W @ v_p @ W.conjugate()

    # Abs not necessary, just to prevent weird output when sigma is high
    mu_2_corrected = np.abs(mu_2)
    mu_2_tilde = Q @ mu_2_corrected @ Q.conjugate()
    eigvals, eigvectors = np.linalg.eig(mu_2_tilde)
    u = eigvectors[:, np.argmax(eigvals)]
    v_tilde = np.fft.ifft(np.fft.fft(u) * np.sqrt(P_x))
    x = ((np.sum(mu_1) / np.sum(v_tilde)) * v_tilde).real
    C_x = np.zeros((L, L))
    for i in range(L):
        C_x[:, i] = x.take(INDEXES[i])
    rho = np.linalg.inv(C_x) @ mu_1
    return x, rho
```

Here calculating a graph would take a really high amount of time, so I will just write some raw data samples:

```
Experiment: N=10000, x=[1 2 3], sigma=1
Error: 0.09836125468862339
Rho: [0.5003038  0.25578641 0.24390979]

Experiment: N=100000, x=[1 2 3], sigma=1
Error: 0.014442829134278811
Rho: [0.49918545 0.25155871 0.24925584]

Experiment: N=1000000, x=[1 2 3], sigma=1
Error: 0.0018508167186262682
Rho: [0.49933984 0.25044228 0.25021788]

Experiment: N=10000, x=[1 2 3], sigma=10
Error: 3.729763083689563
Rho: [0.33762201 0.30188918 0.3604888 ]

Experiment: N=100000, x=[1 2 3], sigma=10
Error: 1.4523607092228141
Rho: [0.43866596 0.28207512 0.27925892]

Experiment: N=1000000, x=[1 2 3], sigma=10
Error: 0.4725805269123462
Rho: [0.45999879 0.27758554 0.26241567]

Experiment: N=10000, x=[ 1  2  3  4  5  6  7  8  9 10], sigma=1
Error: 0.06704925194887679
```

```
Rho: [0.49797579 0.05806266 0.05342327 0.05218292 0.05692324 0.06015525
 0.05385862 0.05588676 0.05592168 0.0556098 ]

Experiment: N=100000, x=[ 1  2  3  4  5  6  7  8  9 10], sigma=1
Error: 0.025724940654790404
Rho: [0.49888513 0.05539118 0.05486342 0.05629742 0.05624215 0.05593763
 0.05664767 0.05603033 0.05349392 0.05621114]

Experiment: N=1000000, x=[ 1  2  3  4  5  6  7  8  9 10], sigma=1
Error: 0.0052796817802200705
Rho: [0.50030414 0.05551767 0.05568073 0.05534556 0.05539625 0.05559898
 0.05548094 0.05519003 0.05560272 0.05588298]
```

If you want me to execute more samples, please let me know.