MTH 602 Scientific Machine Learning

Homework 2

10/1/2025

S. M. Mahfuzul Hasan

02181922

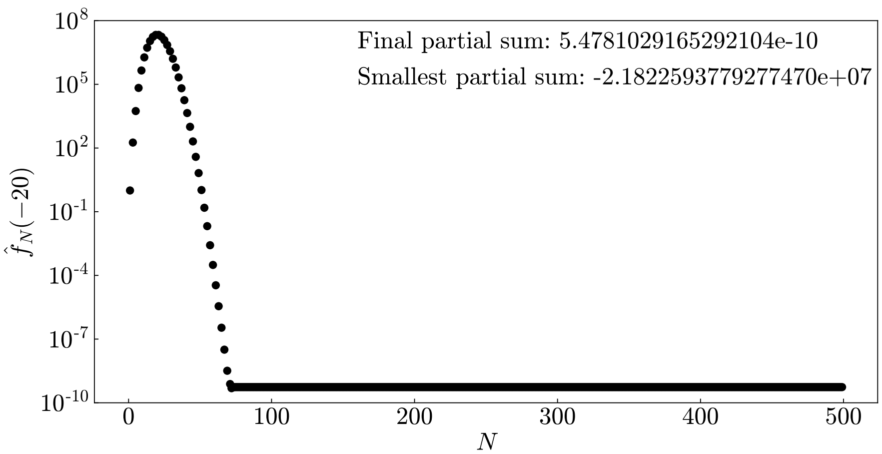
I. FLOATING-POINT NUMBERS

1. N -term Taylor series expansion of the exponential function:

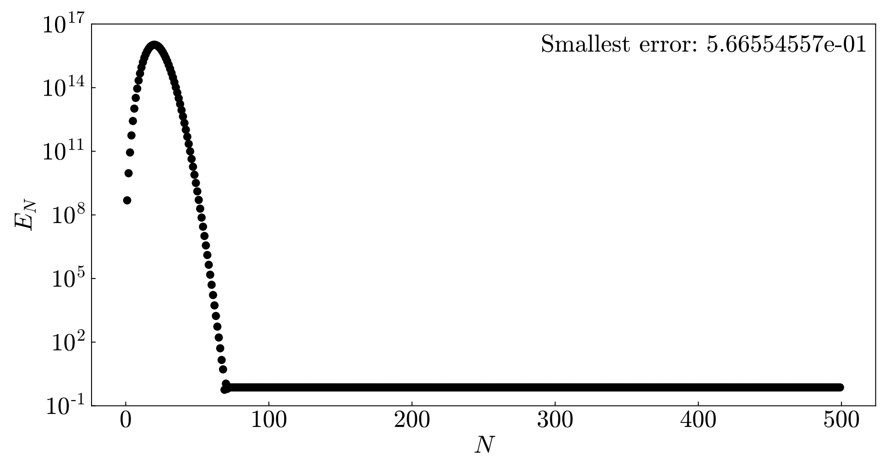
$$e^{-x} \approx \sum_{n=0}^{N-1} \frac{x^n}{n!} = \hat{f}_N(x)$$

(a) The relative error is caluclued by:

$$E_N = \frac{\left|\hat{f}_N(-20) - e^{-20}\right|}{\left|e^{-20}\right|}$$



(a)



(b)

Figure 1: (a) Taylor series expansion value and (b) corresponding relative error of $\hat{f}_N(-20)$ as a function of number of terms.

Smallest relative error: 0.5665545568629835

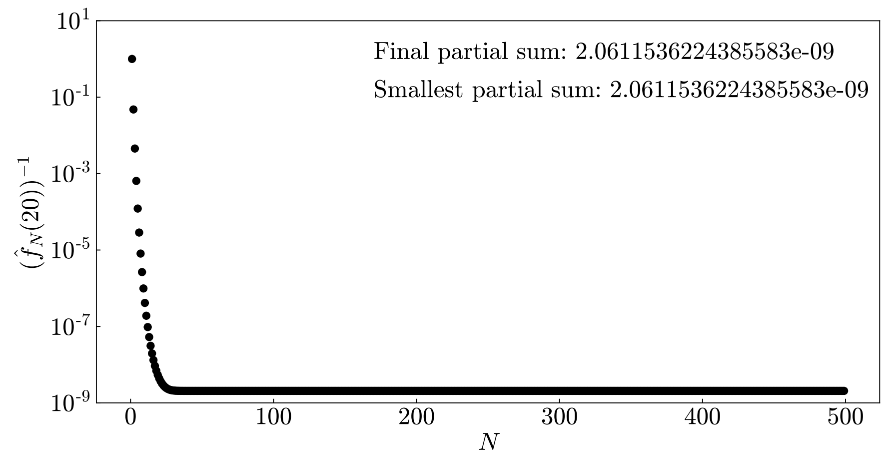
(b) The *Taylor* series expansion for e^{-20} is

$$e^{-20} \approx \frac{(-20)^0}{0!} + \frac{(-20)^1}{1!} + \frac{(-20)^2}{2!} + \frac{(-20)^3}{3!} + \frac{(-20)^4}{4!} + ... = \hat{f}_N(-20)$$

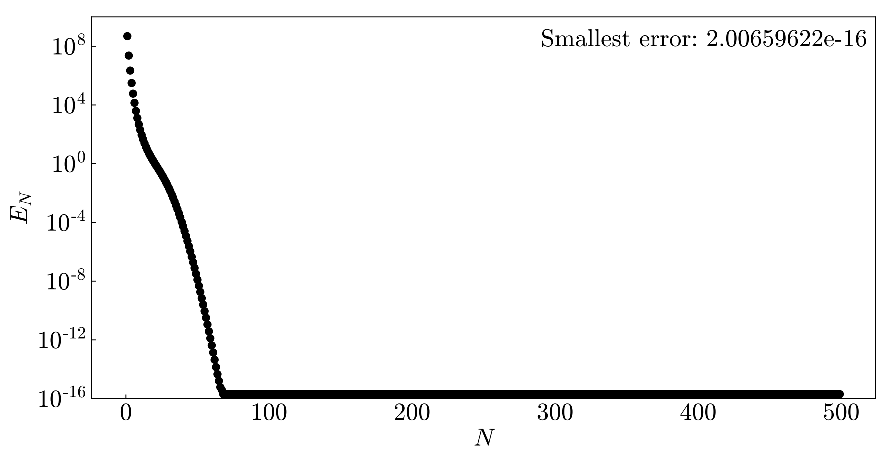
21st and 22nd term of the *Taylor*'s series expansion of e^{-20} are

$$\frac{(-20)^{20}}{20!} \approx 4.31 \times 10^{7} \qquad \text{and} \qquad \frac{(-20)^{21}}{21!} \approx -4.10 \times 10^{7}$$

respectively. They both have the magnitude of order $O(8)$ but their signs are opposite. When such high order of magnitudes cancels each other, disastrous round-off error can show up in the partial sum as it oscillates intensely between large positive and negative values. e^{-20} has a very small exact value of $\sim 2.06 \times 10^{-9}$. CPU uses double precision (accurate till 16 digits). The best round off error can be 10^{-16}, but since the largest term is $\sim 10^{8}$, any difference smaller than 10^{-8} ($10^{-16} \times 10^{8} = 10^{-8}$) is not traceable by the machine. As the exact value of e^{-20} is $\sim 10^{-9}$, it is 10 times smaller than the best possible precision. As a result, the relative error becomes large and rounding artifact gives an incorrect final partial sum as shown in fig. 1a.



(a)



(b)

Figure 2: (a) Taylor series expansion value and (b) corresponding relative error of $\left(\hat{f}_N(20)\right)^{-1}$ as a function of number of terms.

(c) Instead of direct calculation of partial sum using the *Taylor* series expansion of e^{-20}, we can expand e^{20} and then take the inverse of the partial sum. This way all the terms in the series are positive valued, and there is no such cancellations of extremely large numbers involved that happened in previous case.

So, the *Taylor* series expansion becomes

$$e^{-20} = \frac{1}{e^{20}} \approx \frac{1}{\frac{20^0}{0!} + \frac{20^1}{1!} + \frac{20^2}{2!} + \frac{20^3}{3!} + \frac{20^4}{4!} + ...} = \hat{f}_N(-20)$$

As a result, round off error (despite present) can not cause any issue in the convergence of $\hat{f}_N(-20)$ to the exact value because successive positive terms are added in partial sum of e^{20} which first, peaks and then, flattens with higher number of terms, imparting the same behavior to e^{-20}. The final and smallest partial sum points to the same value as can be seen from fig. 2a with machine accuracy $\sim 10^{-16}$ as fig. 2b indicates.

```python
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.ticker import MultipleLocator

def taylor_exp(N, x):
    """
    Compute the N-term Taylor series expansion of e^x about 0, evaluated at x.
    Parameters
    ----------
    N : int
    Number of terms in the Taylor expansion.
    x : float
    Point at which to evaluate the expansion.
    Returns
    -------
    float
    Approximation of e^x using the first N terms of the series.
    """
    import math
    total = 0.0
    for n in range(N):
        total += (x**n) / math.factorial(n)
    return total

# finding $e^{-20}$
N = 500 # no. of terms
x = -20 # evaluation point

N_arr1 = [] # array for terms
taylor_arr1 = [] # array for partial sum
error_arr1 = [] # array for error

# taylor expansion partial sum and error
for i in range(1, N):
    value = taylor_exp(i, x) # partial sum
    error = np.abs(value - math.exp(-20))/np.abs(math.exp(-20)) # error
    N_arr1.append(i) # update term
    taylor_arr1.append(value) # update partial sum
    error_arr1.append(error) # update error
```

```


print(f'Smallest error: ', min(error_arr1))

#parameters for plotting
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'cmr10'
plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.size'] = 21
mpl.rcParams['axes.unicode_minus'] = False

# plotting the partial sum and error
fig, ax = plt.subplots(figsize=(12, 6))
ax.semilogy(N_arr1, taylor_arr1, 'o', color='black')
plt.xlabel('$N$')
plt.ylabel(r'$\hat f_N(-20)$')
plt.ylim(1e-10, 1e8)
plt.text(160, 5e6, f"Final partial sum: {taylor_arr1[-1]:.16e}")
plt.text(160, 1e5, f"Smallest partial sum: {min(taylor_arr1):.16e}")
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('exp-20.pdf', dpi=1080)
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
ax.semilogy(N_arr1, error_arr1, 'o', color='black')
plt.xlabel('$N$')
plt.ylabel(r'$E_N$')
plt.ylim(1e-1, 1e17)
plt.text(290, 5e15, f"Smallest error: {min(error_arr1):.8e}")
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('exp-20_error.pdf', dpi=1080)
plt.show()

# finding $\frac{1}{e^20}$
x = 20

N_arr2 = [] # array for terms
taylor_arr2 = [] # array for partial sum
error_arr2 = [] # array for error

# taylor expansion partial sum and error
for i in range(1, N):
    value = 1/(taylor_exp(i, x)) # partial sum
    error = np.abs(value - math.exp(-20))/np.abs(math.exp(-20)) # error
    N_arr2.append(i) # update term
    taylor_arr2.append(value) # update partial sum
    error_arr2.append(error) # update error

print(f'Smallest error: ', min(error_arr2))

# plotting the partial sum and error
fig, ax = plt.subplots(figsize=(12, 6))
ax.semilogy(N_arr2, taylor_arr2, 'o', color='black')
plt.xlabel('$N$')
plt.ylabel(r'${(\hat f_N(20))}^{-1}$')
plt.ylim(1e-9, 1e1)
plt.text(170, 1e0, f"Final partial sum: {taylor_arr2[-1]:.16e}")
plt.text(170, 1e-1, f"Smallest partial sum: {min(taylor_arr2):.16e}")
plt.tick_params(axis="both", which="both", direction="in")
```

```python
plt.savefig('exp20.pdf', dpi=1080)
plt.show()

fig, ax = plt.subplots(figsize=(12, 6))
plt.semilogy(N_arr2, error_arr2, 'o', color='black')
plt.xlabel('$N$')
plt.ylabel(r'$E_N$')
plt.ylim(1e-16, 1e10)
plt.text(290, 1e8, f"Smallest error: {min(error_arr2):.8e}")
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig('exp20_error.pdf', dpi=1080)
plt.show()
```

Listing 1: *problem1.py*

(d) (Optional)

2. Given,

$$A = \begin{bmatrix} 10^3 & 0 \\ 0 & 10^{-2} \end{bmatrix}$$

(a) For each component of $\vec{x}$, the error bound is

$$\begin{aligned} \frac{|\delta x_i|}{|x_i|} &\leq N\epsilon_m \\ \Rightarrow |\delta x_i| &\leq N\epsilon_m |x_i| \end{aligned} \tag{1}$$

Now using inequality (1),

$$\begin{aligned} &\|\delta x\|_2^2 = \sum_{i=1}^{2} |x_i|^2 \leq \sum_{i=1}^{2} (N\epsilon_m |x_i|)^2 = (N\epsilon_m)^2 \sum_{i=1}^{2} |x_i|^2 = (N\epsilon_m)^2 \|x\|_2^2 \\ \Rightarrow &\frac{\|\delta x\|_2^2}{\|x\|_2^2} \leq (N\epsilon_m)^2 \\ \Rightarrow &\frac{\|\delta x\|_2}{\|x\|_2} \leq N\epsilon_m \quad \text{(Ans.)} \end{aligned} \tag{2}$$

(b)

$$\frac{\|\delta y\|_2}{\|y\|_2} = \frac{\|A\delta x\|_2}{\|Ax\|_2} \leq \kappa(A) \frac{\|\delta x\|_2}{\|x\|_2} = \|A\|_2 \|A^{-1}\|_2 \frac{\|\delta x\|_2}{\|x\|_2} \tag{3}$$

Now,

$$A^{-1} = \frac{1}{10^3 \cdot 10^{-2}} \begin{bmatrix} 10^{-2} & 0 \\ 0 & 10^3 \end{bmatrix} = \begin{bmatrix} 10^{-3} & 0 \\ 0 & 10^2 \end{bmatrix}$$

If a_{ij} and a'_{ij} are the diagonal elements of A and A^{-1} respectively, then

$$\|A\|_2 \|A^{-1}\|_2 = \max_{i=j}(|a_{ij}|) \cdot \max_{i=j}(|a'_{ij}|) = 10^3 \cdot 10^2 = 10^5 \tag{4}$$

Using inequality (2) and equation (4), we get from inequality (3)

$$\frac{\|\delta y\|_2}{\|y\|_2} \leq 10^5 N\epsilon_m \quad \text{(Ans.)} \tag{5}$$

(c) For $N = 10^3$ and $\epsilon_m = 10^{-8}$ (single precision), using equation (5), we get

$$\frac{\|\delta y\|_2}{\|y\|_2} \leq 10^5 \cdot 10^3 \cdot 10^{-8} = 1 \quad \text{(Ans.)}$$

3. (Optional)

II. PROBABILITY (CHAPTER 2)

1. Given,

$$p(x) = \frac{1}{d-c}, \qquad x \in [c, d]$$

For normalization, $\int\limits_{-\infty}^{\infty} p(x)dx$ should be 1.

$$\int\limits_{-\infty}^{\infty} p(x)dx = \int\limits_{c}^{d} p(x)dx = \int\limits_{c}^{d} \frac{1}{d-c}dx = \frac{1}{d-c}\left[x\right]_c^d = \frac{1}{d-c}(d-c) = 1$$

So, $p(x)$ is correctly normalized.

Expected value:

$$\begin{aligned}\mathbb{E}\left[f(x)\right] = \int\limits_{c}^{d} xp(x)dx = \frac{1}{d-c}\int\limits_{c}^{d} xdx = \frac{1}{d-c}\left[\frac{x^2}{2}\right]_c^d &= \frac{1}{d-c}\frac{\left(d^2-c^2\right)}{2} \\ &= \frac{1}{2}(c+d) \quad \text{(Ans.)}\end{aligned}$$

Variance:

$$\begin{aligned}\mathbb{E}\left[(f(x))^2\right] = \int\limits_{c}^{d} x^2p(x)dx = \frac{1}{d-c}\int\limits_{c}^{d} x^2dx = \frac{1}{d-c}\left[\frac{x^3}{3}\right]_c^d &= \frac{1}{d-c}\frac{\left(d^3-c^3\right)}{2} \\ &= \frac{1}{3}(c^2+cd+d^2)\end{aligned}$$

$$\begin{aligned}Var(X) &= \mathbb{E}\left[(f(x))^2\right] - \left(\mathbb{E}\left[f(x)\right]\right)^2 = \frac{1}{3}(c^2+cd+d^2) - \left(\frac{1}{2}(c+d)\right)^2 \\ &= \frac{1}{3}(c^2+cd+d^2) - \frac{1}{4}(c^2+2cd+d^2) = \frac{4c^2+4cd+4d^2-3c^2-6cd-3d^2}{12} \\ &= \frac{c^2-2cd+d^2}{12} = \frac{1}{12}(d-c)^2 \quad \text{(Ans.)}\end{aligned}$$

2.

$$\begin{aligned}
L.H.S = \mathbb{E}\left[\alpha f(x) + \beta g(x)\right] &= \int\limits_{-\infty}^{\infty} \left(\alpha f(x) + \beta g(x)\right) p(x) dx \\
&= \int\limits_{-\infty}^{\infty} \alpha f(x) p(x) dx + \int\limits_{-\infty}^{\infty} \beta g(x) p(x) dx = \alpha \int\limits_{-\infty}^{\infty} f(x) p(x) dx + \beta \int\limits_{-\infty}^{\infty} g(x) p(x) dx \\
&= \mathbb{E}\left[\alpha f(x)\right] + \mathbb{E}\left[\beta g(x)\right] = R.H.S \quad \text{(Showed)}
\end{aligned}$$

3. **(Optional)**

4. Problems from book:
Problem 2.16
Given,

$$\mathbb{E}[x] = \int\limits_{-\infty}^{\infty} \mathcal{N}\left(x|\mu, \sigma^2\right) x dx = \mu \tag{6}$$

$$\mathbb{E}\left[x^2\right] = \int\limits_{-\infty}^{\infty} \mathcal{N}\left(x|\mu, \sigma^2\right) x^2 dx = \mu^2 + \sigma^2 \tag{7}$$

To show and prove,

$$\mathbb{E}\left[x_n x_m\right] = \mu^2 + I_{nm}\sigma^2$$
$$\mathbb{E}\left[\mu_{ML}\right] = \mu$$
$$\mathbb{E}\left[\sigma^2_{ML}\right] = \left(\frac{N-1}{N}\right)\sigma^2$$

where x_n and x_m denote data points sampled from a Gaussian distribution with mean μ and variance σ^2 and I_{nm} satisfies $I_{nm} = 1$ if $n = m$ and $I_{nm} = 0$ otherwise.

If $n = m$, then using equation (7)

$$\mathbb{E}[x_n^2] = \mu^2 + \sigma^2 = \mu^2 + 1 \cdot \sigma^2 \tag{8}$$

If $n \neq m$, then the data points are independent. So, using equation (6)

$$\mathbb{E}[x_n x_m] = \mathbb{E}[x_n]\mathbb{E}[x_m] = \mu \cdot \mu = \mu^2 = \mu^2 + 0 \cdot \sigma^2 \tag{9}$$

From equation (8) and (9), we can conclude that

$$\mathbb{E}\left[x_n x_m\right] = \mu^2 + I_{nm}\sigma^2 \quad \text{(Showed)}$$

Next, we know the maximum likelihood solution for mean

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

So, the expectation of μ_{ML} is

$$\mathbb{E}\left[\mu_{ML}\right] = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} x_n\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[x_n\right] = \frac{1}{N}\sum_{n=1}^{N}\mu \quad \text{[using linearity and equation (6)]}$$

$$= \frac{\mu}{N}\sum_{n=1}^{N} 1 = \frac{\mu}{N}\cdot N = \mu \quad \text{(Proved)}$$

We also know the maximum likelihood solution for variance

$$\sigma_{ML}^2 = \frac{1}{N}\sum_{n=1}^{N}\left(x_n - \mu_{ML}\right)^2$$

So, the expectation of σ_{ML}^2 is

$$\begin{aligned}
\mathbb{E}\left[\sigma_{ML}^2\right] &= \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n - \mu_{ML}\right)^2\right] = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2 - 2x_n\mu_{ML} + \mu_{ML}^2\right)\right] \\
&= \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\left(x_n^2 - 2x_n\frac{1}{N}\sum_{m=1}^{N}x_m + \left(\frac{1}{N}\sum_{m=1}^{N}x_m\right)^2\right)\right] \\
&\qquad \text{[using equation (11)]} \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[x_n^2\right] - \frac{2}{N^2}\sum_{n=1}^{N}\sum_{m=1}^{N}\mathbb{E}\left[x_n x_m\right] + \frac{1}{N^3}\sum_{n=1}^{N}\left(\sum_{m=1}^{N}\sum_{k=1}^{N}\mathbb{E}\left[x_m x_k\right]\right) \\
&\qquad \text{[using linearity]}
\end{aligned} \tag{10}$$

Now

$$\sum_{n=1}^{N}\mathbb{E}\left[x_n^2\right] = N\left(\mu^2 + \sigma^2\right) \quad \text{[using equation (8)]}$$

$$\begin{aligned}
\sum_{n=1}^{N}\sum_{m=1}^{N}\mathbb{E}\left[x_n x_m\right] &= N\mathbb{E}\left[x_n^2\right] + \left(N^2 - N\right)\mathbb{E}\left[x_n x_m\right] \\
&= N\left(\mu^2 + \sigma^2\right) + \left(N^2 - N\right)\mu^2 \quad \text{[using equation (8) and (9)]} \\
&= N^2\mu^2 + N\sigma^2
\end{aligned}$$

$$\begin{aligned}
\sum_{n=1}^{N}\left(\sum_{m=1}^{N}\sum_{k=1}^{N}\mathbb{E}\left[x_m x_k\right]\right) &= N\left(N\mathbb{E}\left[x_m^2\right] + \left(N^2 - N\right)\mathbb{E}\left[x_m x_k\right]\right) \\
&= N^2\left(\mu^2 + \sigma^2\right) + \left(N^3 - N^2\right)\mu^2 \quad \text{[using equation (8) and (9)]} \\
&= N^3\mu^2 + N^2\sigma^2
\end{aligned}$$

Replacing the evaluated terms in equation (10)

$$\begin{aligned}
\mathbb{E}\left[\sigma_{ML}^2\right] &= \frac{1}{N}\cdot N\left(\mu^2+\sigma^2\right) - \frac{2}{N^2}\left(N^2\mu^2+N\sigma^2\right) + \frac{1}{N^3}\left(N^3\mu^2+N^2\sigma^2\right) \\
&= \mu^2+\sigma^2-2\mu^2-\frac{2}{N}\sigma^2+\mu^2+\frac{1}{N}\sigma^2 \\
&= \frac{N-1}{N}\sigma^2 \quad \text{(Proved)}
\end{aligned}$$

Problem 2.18

Given,

$$\ln p\left(\mathbf{t}|\mathbf{x},\mathbf{w},\sigma^2\right) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi) \tag{11}$$

To show

$$\sigma_{ML}^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}_{ML}\right)-t_n\right\}^2$$

Differentiating equation (11) with respet to σ^2

$$\begin{aligned}
\frac{\partial}{\partial\sigma^2}\left(\ln p\left(\mathbf{t}|\mathbf{x},\mathbf{w},\sigma^2\right)\right) &= \frac{\partial}{\partial\sigma^2}\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi)\right) \\
&= \frac{1}{2(\sigma^2)^2}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 - \frac{N}{2}\frac{1}{\sigma^2} - 0 \qquad (12) \\
&= \frac{1}{2\sigma^4}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 - \frac{N}{2\sigma^2}
\end{aligned}$$

Maximizing likelihood in $\mathbf{w}$ means minimizing the residual, hence setting equation (12) to zero.

$$\begin{aligned}
&\frac{1}{2\sigma^4}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 - \frac{N}{2\sigma^2} = 0 \\
\Rightarrow &\frac{N}{2\sigma^2} = \frac{1}{2\sigma^4}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2 \qquad (13) \\
\Rightarrow &\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}\right)-t_n\right\}^2
\end{aligned}$$

Setting $\mathbf{w}=\mathbf{w}_{ML}$ and consequently, $\sigma^2=\sigma_{ML}^2$ in equation (13), we get

$$\sigma_{ML}^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{y\left(x_n,\mathbf{w}_{ML}\right)-t_n\right\}^2 \quad \text{(Showed)}$$

III. IS THERE A SIGNAL IN MY TIME-SERIES DATA? APPLICATION OF PROBABILITY THEORY

- Given,

$$p\left(\mathbf{n}|0,1\right) = \prod_{i=1}^{N} \mathcal{N}\left(n_i|0,1\right)$$

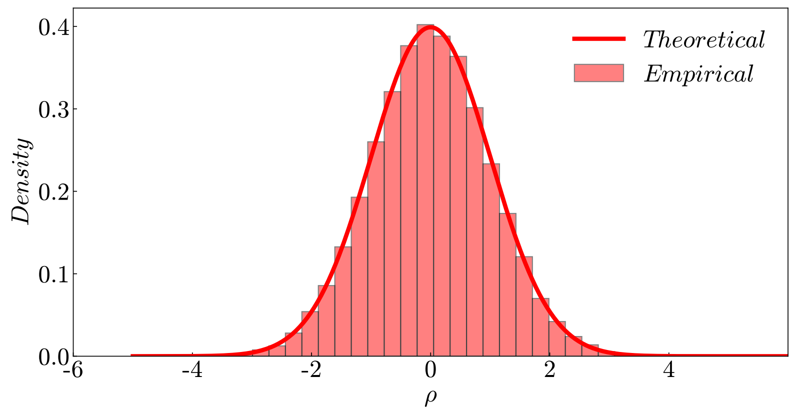
$$\rho(A) = \langle \mathbf{y}, \hat{\mathbf{s}} \rangle = \sum_{i=1}^{N} y_i \hat{s}_i$$

where, $y_i = s_i + n_i$, $\hat{s}_i = A\hat{s}_i$ and $\sum_{i=1}^{N} {s_i}^2 = 1$

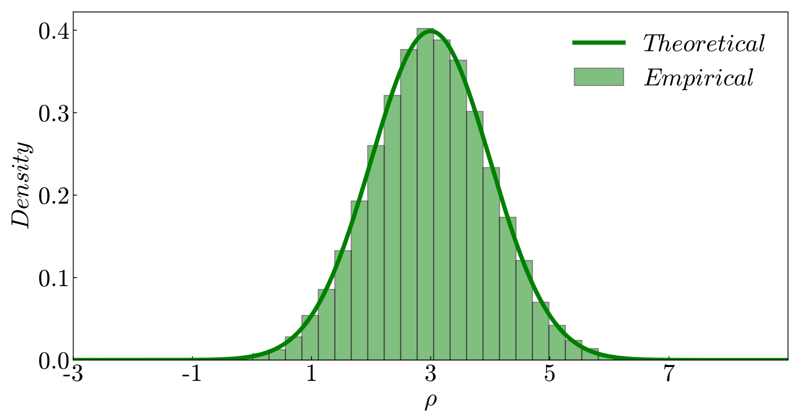
$$\begin{aligned}
\mathbb{E}\left[\rho(A)\right] &= \mathbb{E}\left[\sum_{i=1}^{N} y_i \hat{s}_i\right] = \mathbb{E}\left[\sum_{i=1}^{N} \left(s_i + n_i\right)\hat{s}_i\right] = \mathbb{E}\left[\sum_{i=1}^{N} \left(A\hat{s}_i + n_i\right)\hat{s}_i\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{N} A\hat{s}_i^2 + \sum_{i=1}^{N} n_i \hat{s}_i\right] = \mathbb{E}\left[A\sum_{i=1}^{N} \hat{s}_i^2 + \sum_{i=1}^{N} n_i \hat{s}_i\right] \\
&= \mathbb{E}\left[A \cdot 1 + \sum_{i=1}^{N} n_i \hat{s}_i\right] = A + \sum_{i=1}^{N} \mathbb{E}\left[n_i \hat{s}_i\right] \\
&\qquad\qquad [\mathbb{E}[A] = A \text{ since, } A \text{ is a scalar constant}] \\
&= A + \sum_{i=1}^{N} \hat{s}_i \mathbb{E}\left[n_i\right] \quad [\hat{s}_i \text{ is just a vector coefficient}] \\
&= A + \sum_{i=1}^{N} \hat{s}_i \cdot 0 = A \quad \text{(Ans.)}
\end{aligned} \tag{14}$$

$$
\begin{aligned}
Var\left(\rho(A)\right) &= \mathbb{E}\left[\left(\rho(A)\right)^2\right] - \left(\mathbb{E}\left[\rho(A)\right]\right)^2 = \mathbb{E}\left[\left(\sum_{i=1}^{N} y_i \hat{s}_i\right)^2\right] - A^2 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{[using equation (14)]} \\
&= \mathbb{E}\left[\left(A + \sum_{i=1}^{N} n_i \hat{s}_i\right)^2\right] - A^2 \quad \text{[using equation (14)]} \\
&= \mathbb{E}\left[A^2 + 2A\sum_{i=1}^{N} n_i \hat{s}_i + \left(\sum_{i=1}^{N} n_i \hat{s}_i\right)^2\right] - A^2 \\
&= \mathbb{E}\left[A^2\right] + \mathbb{E}\left[2A\sum_{i=1}^{N} n_i \hat{s}_i\right] + \mathbb{E}\left[\sum_{i=1}^{N}\sum_{j=1}^{N} n_i s_i n_j s_j\right] - A^2 \\
&= A^2 + 2A \cdot 0 + \sum_{i=1}^{N}\sum_{j=1}^{N} s_i s_j \mathbb{E}\left[n_i n_j\right] - A^2 \quad \text{[again, using equation (14)]} \\
&= \sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N} s_i s_j \mathbb{E}\left[n_i\right]\mathbb{E}\left[n_j\right] + \sum_{i=1}^{N} s_i^2 \mathbb{E}\left[n_i^2\right] \\
&\qquad\qquad\qquad\qquad\qquad \text{[n_i and n_j are independent of one another]} \\
&= \sum_{i=1}^{N}\sum_{\substack{j=1 \\ j\neq i}}^{N} s_i s_j \cdot 0 \cdot 0 + \sum_{i=1}^{N} s_i^2\left(0^2 + 1^2\right) \quad \text{[using equation (6) and (7)]} \\
&= \sum_{i=1}^{N} s_i^2 = 1 \quad \text{(Ans.)}
\end{aligned}
$$

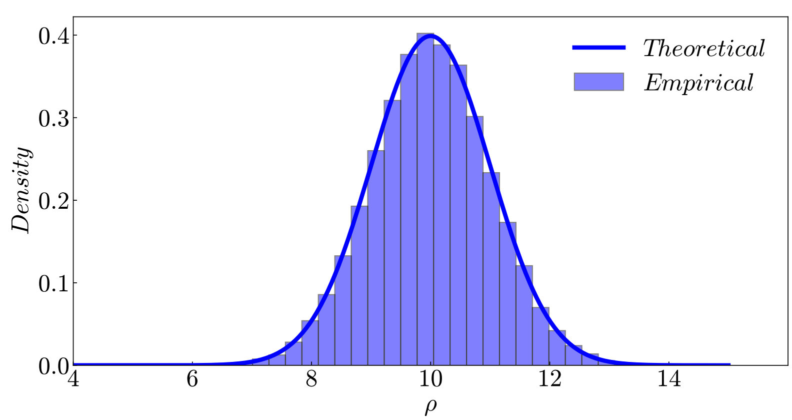
- Set-up for the estimation of the distribution:
  - No. of data points in each realization, $N = 1000$
  - No. of realization $= 50000$



(a)



(b)



(c)

Figure 3: Empirical distribution of $\rho(A)$ for (a) $A = 0$, (b) $A = 3$, and (c) $A = 10$.

Empirical estimation:

For $A = 0$,

$$\rho \sim \mathcal{N}(0.00603666, 1.00015936)$$

For $A = 3$,

$$\rho \sim \mathcal{N}(3.00603666, 1.00015936)$$

For $A = 10$,

$$\rho \sim \mathcal{N}(10.00603666, 1.00015936)$$

The distributions mimic the theoretically obtained Gaussian distribution of $\rho(A) \sim \mathcal{N}(A, 1)$ pretty well as can be seen from the computed mean and variance values, and from the fig. 3 and 4, which suggests that the theoretical formulae for mean and variance obtained in previous part are correct.

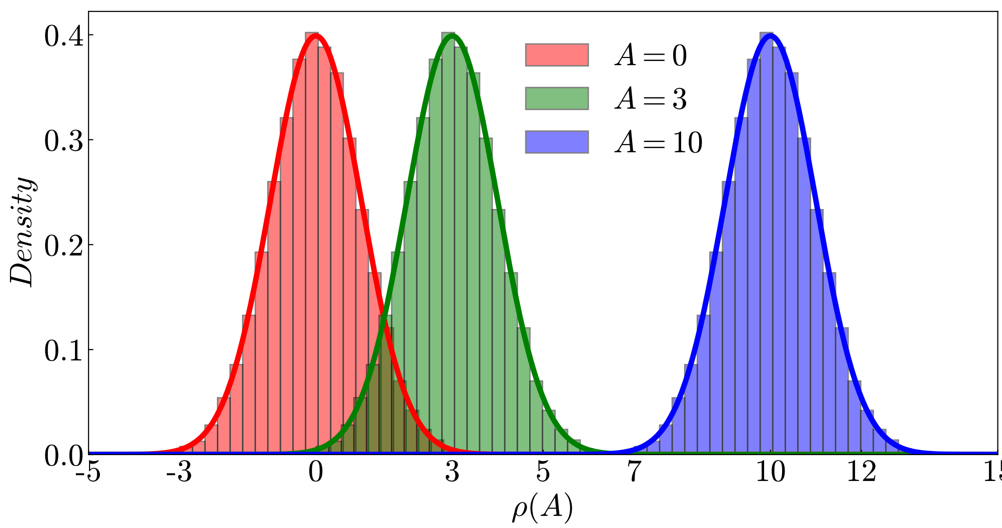


Figure 4: Relative position of the empirical distribution of $\rho(A)$ for $A = \{0, 3, 10\}$.

- Set-up for the estimation of the distribution:
  - No. of data points in each mock dataset, $N = 1000$
  - No. of mock dataset $= 1000$
  - Threshold limit: $[-3, 8]$
  - No. of threshold points evaluated $= 2000$

  For $\rho_0 = 1.561781$, the number of false negatives and false positives are 31 and 31 respectively. So, they balance out each other for 1000 mock datasets.

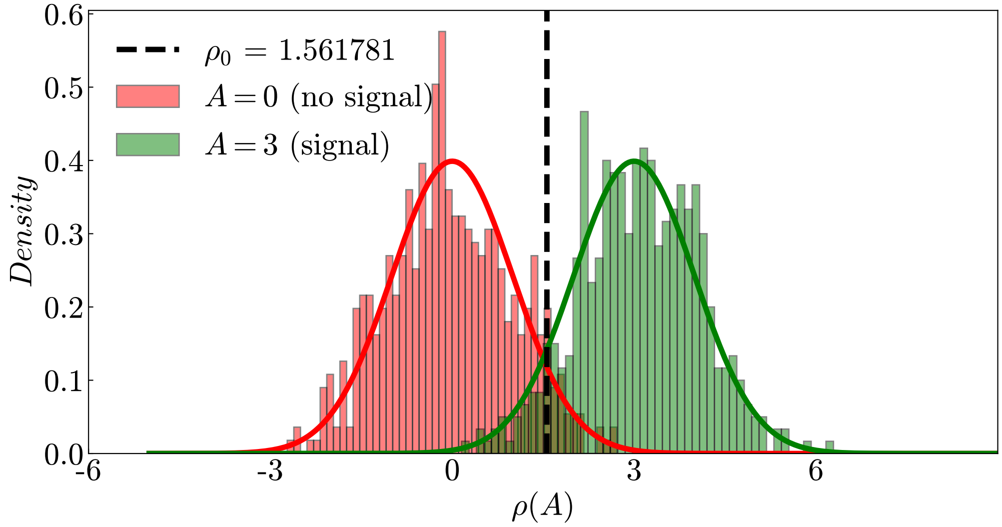


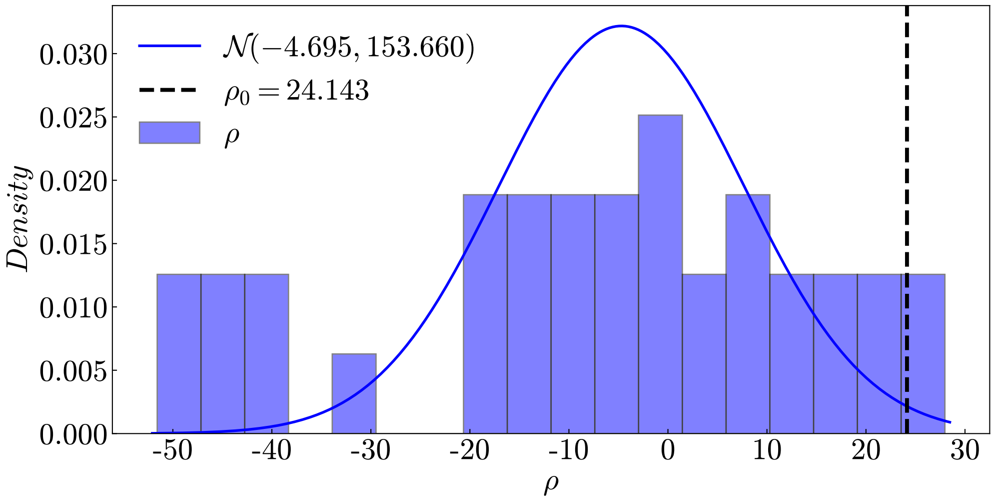
Figure 5: Empirical distribution of $\rho(A)$ for $A = \{0, 3\}$ to classify signal-or-no signal.

The confusion matrix as it stands:

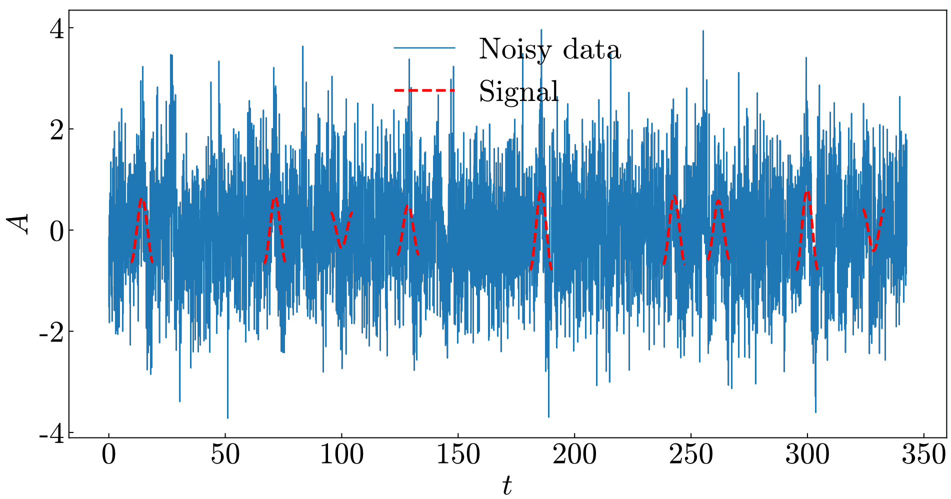
	Predicted: No Signal	Predicted: Signal
Actual: No Signal	True Negative 480	False Positive 31
Actual: Signal	False Negative 31	True Positive 458

A. Class Competition

No. of signals: 9



(a)



(b)

Figure 6: (a) Empirical distribution of ρ for noise. (b) Constructed underlying signal.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.stats import norm

def generate_data(A, N, seed=None):
    """
    Generate a noisy time series dataset {t_i, y_i}_{i=1}^N.
    Parameters
    ----------
    A : float
    Amplitude of the signal (A >= 0)
```

```python
    N : int
    Number of data points. Time points uniformly sampled from [0, 2pi]
    seed : int or None, optional
    Random seed for reproducibility
    """
    rng = np.random.default_rng(seed)
    t = np.linspace(0.0, 2*np.pi, N)
    dt = (2*np.pi) / (N - 1)
    s_hat = np.sqrt(dt / np.pi) * np.sin(t) # ||s_hat||_2 == 1 exactly on this grid
    s = A * s_hat
    n = rng.normal(loc=0.0, scale=1.0, size=N)
    y = s + n
    return t, y, s, n, s_hat, dt

# ===== Emperical and theoretical Gaussian distribution =====

A = [0, 3, 10] # input amplitudes
N = 1000 # no. of data points
seed = 32 # for reproducibility
n_realization = 50000 # no. of realizations

rho_list = {} # $\rho(A)$ list

for i in A:
    rho_arr = [] # $\rho(A)$ array for realizations
    for j in range(n_realization):
        t, y, s, n, s_hat, dt = generate_data(i, N, seed + j)
        rho = np.dot(y, s_hat)
        rho_arr.append(rho) # update new realization
    rho_list[i] = np.array(rho_arr) # update list

for i in A:
    print(f"A={i}: rho_mean={rho_list[i].mean():.8f}, rho_var={rho_list[i].var():.8f}")

# parameters for plotting
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.serif'] = 'cmr10'
plt.rcParams['mathtext.fontset'] = 'cm'
plt.rcParams['font.size'] = 23
mpl.rcParams['axes.unicode_minus'] = False

# individual plots of distributions for different $A$ values
x = np.linspace(-5, 15, 1000)
clr = ['red', 'green', 'blue']
lbl = [r"$A=0$", r"$A=3$", r"$A=10$"]

for i, color in zip(A, clr):
    fig, ax = plt.subplots(figsize=(12, 6))

    # empirical histogram
    ax.hist(rho_list[i], bins=30, density=True, alpha=0.5,
            label=r"$Empirical$", color = color, edgecolor='black')

    # theoretical gaussian line plot
    ax.plot(x, norm.pdf(x, loc=i, scale=1), color=color, ls = '-', lw=4,
            label=r"$Theoretical$")

```

```
    plt.xlim(i-6, i+6)
    ax.set_xticks(np.arange(i-6, i+6, 2))
    plt.xlabel(fr"$\rho$")
    plt.ylabel(fr"$Density$")
    plt.legend(frameon=False)
    plt.tick_params(axis="both", which="both", direction="in")
    plt.savefig(f'hist{i}.pdf', dpi=1080)
    plt.show()

# single plot of the three distributions
fig, ax = plt.subplots(figsize=(12, 6))

for i, color, label in zip(A, clr, lbl):
    # empirical histogram
    ax.hist(rho_list[i], bins=30, alpha=0.5, density=True,
              color=color, label=label, edgecolor='black')

    # theoretical gaussian line plot
    ax.plot(x, norm.pdf(x, loc=i, scale=1), linestyle='-', color=color, linewidth = 4)

plt.xlim(-5, 15)
ax.set_xticks([-5, -3, 0, 3, 5, 7, 10, 12, 15])
plt.xlabel(r'$\rho(A)$')
plt.ylabel('$Density$')
plt.legend(loc='upper left', bbox_to_anchor=(0.45, 0.99), frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'hist.pdf', dpi=1080)
plt.show()

# ===== Construction of confusion matrix =====

N_c = 1000 # no. of data points
seed = 32 # for reproducibility
n_dataset = 1000 # no. of mock datasets

rng = np.random.default_rng(seed)
signal_label = rng.choice([0, 3], size=n_dataset) # randomly assignment of $A=0$ or $A=3$

rho_arr = [] # $rho$ array

# generating each mock dataset with amplitude
for i, A in enumerate(signal_label):
    t, y, s, n, s_hat, dt = generate_data(A, N, seed + i)
    rho_arr.append(np.dot(y, s_hat)) # update $rho$ array

rho_ = np.array(rho_arr)

actual = (signal_label == 3).astype(int) # 1 if signal, 0 if no signal

thres = np.linspace(-3, 8, 2000) # 2000 threshold points evaluated
min_diff = 10 # initialized difference between false negative and false positive

for rho0 in thres:
    predict = (rho_ >= rho0).astype(int) # 1 if the condition is true, 0 if false

    # condiions of four states of confusion matrix
    false_negative = np.sum((predict==0) & (actual==1))
```

```python
    true_negative = np.sum((predict==0) & (actual==0))
    true_positive = np.sum((predict==1) & (actual==1))
    false_positive = np.sum((predict==1) & (actual==0))

    # difference between false negative and false positive
    diff = abs(false_negative - false_positive)

    # minimizing the difference
    if diff < min_diff:
        min_diff = diff
        best_thres = rho0
        best_confus = [[true_negative, false_positive], [false_negative,
            true_positive]]

print(fr'Best threshold = {best_thres:.6f}')
print('Confusion matrix: ')
print(np.array(best_confus))

# plots
rho0 = rho_[signal_label == 0]
rho3 = rho_[signal_label == 3]

fig, ax = plt.subplots(figsize=(12, 6))

# histograms
ax.hist(rho0, bins=50, density=True, alpha=0.5, color='red',
         edgecolor='black', label=r'$A=0$ (no signal)')
ax.hist(rho3, bins=50, density=True, alpha=0.5, color='green',
         edgecolor='black', label=r'$A=3$ (signal)')

# theoretical gaussian line plot
ax.plot(x, norm.pdf(x, loc=0, scale=1), linestyle='-', color='red', linewidth = 4)
ax.plot(x, norm.pdf(x, loc=3, scale=1), linestyle='-', color='green', linewidth =
    4)

# threshold line
plt.axvline(best_thres, color='k', linestyle='--', linewidth=4,
            label=fr'$\rho_0$ = {best_thres:.6f}')

plt.xlim(-6, 9)
ax.set_xticks(np.arange(-6, 9, 3))
plt.xlabel(r'$\rho(A)$')
plt.ylabel('$Density$')
plt.legend(loc='upper left', frameon=False)
plt.tick_params(axis="both", which="both", direction="in")
plt.savefig(f'confusion.pdf', dpi=1080)
plt.show()

# ===== Class competition =====

# read csv
data = np.loadtxt("hw2.csv", delimiter=",")
t = data[:,0]
y = data[:,1]
print('Length of y:', len(y))

# scaling function
def scaling(x):
    med = np.median(x)
```

```
    mad = np.median(np.abs(x - med)) # mean absolute deviation to determine the
        sparsity of data
    sigma = 1.4826 * mad # standart deviation

    # centering and rescaling (if possible)
    if sigma > 0:
        return (x - med) / sigma
    else:
        return x - med

# loop over different size of bins
for N in range(50, 1001, 50):
    dt = t[1] - t[0] # timestep
    t_ = t[:N]
    omega = 2*np.pi / (t_[-1] - t_[0])
    s_hat = np.cos(omega * (t_ - t_[0]))  # discrete template as cosine function

    rho_g = []
    segments = []  # for optional template refinement

    # $rho$ computation
    for i in range(0, len(y), N):
        y_ = y[i:i+N]
        if len(y_) < N:
            continue
        y_scaled = scaling(y_)
        rho = np.dot(y_scaled, s_hat)
        rho_g.append(rho)
        segments.append((i, y_))

    rho_g = np.array(rho_g)

    # computing noise mean, spread and threshold by taking 60% center data
    rho_lo, rho_hi = 20, 80
    lo, hi = np.percentile(rho_g, [rho_lo, rho_hi])
    noise_chunk = rho_g[(rho_g >= lo) & (rho_g <= hi)]
    mu_ = np.median(noise_chunk)
    mad = np.median(np.abs(noise_chunk - mu_))
    if mad > 0:
        sigma_ = 1.4826 * mad
    else:
        sigma_ = np.std(noise_chunk, ddof=1)

    alpha = 0.01
    rho0 = mu_ + sigma_ * norm.ppf(1 - alpha)

    detect = np.where(np.abs(rho_g) >= rho0)[0] # condition for detection
    sig_count = len(detect) # counting signals
    print(f'N={N}: Estimated number of signals = {sig_count} / {len(rho_g)}')

    # refine template from detected bins
    refined_template = s_hat  # fallback default
    if len(detect) > 0:
        detected_segs = []
        for idx in detect:
            i, y_seg = segments[idx]
            if len(y_seg) == N:
                y_seg = y[i:i+N]
                a = np.dot(y_seg, s_hat) / np.dot(s_hat, s_hat)
```

```
                signal_estimate = a * s_hat
                detected_segs.append(signal_estimate)
        refined_template = np.mean(detected_segs, axis=0)

    # histogram for each N
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.hist(rho_g, bins=max(10, min(40, len(rho_g)//2)), density=True,
            alpha=0.5, color='blue', edgecolor='black', label=r"$\rho$")
    x = np.linspace(min(rho_g)-0.5, max(rho_g)+0.5, 500)
    ax.plot(x, norm.pdf(x, loc=mu_, scale=sigma_), 'b-', lw=2,
            label=fr"$\mathcal{{N}}({mu_:.3f},{sigma_**2:.3f})$")
    ax.axvline(rho0, color='black', ls='--', lw=3,
               label=fr'$\rho_0={rho0:.3f}$')
    plt.xlabel(r'$\rho$')
    plt.ylabel(r'$Density$')
    plt.legend(frameon=False)
    plt.tick_params(axis="both", which="both", direction="in")
    plt.savefig(f'hist_sig{N}.pdf', dpi = 1080)
    plt.show()

    # signal line plot for each N
    fig, ax = plt.subplots(figsize=(12, 6))
    ax.plot(t, y, linewidth=1, label="Noisy data")

    for i, j in enumerate(detect):
        seg_start = j * N
        seg_t = t[seg_start:seg_start+N]
        if len(seg_t) < N:
            continue

        y_seg = y[seg_start:seg_start+N]
        a = np.dot(y_seg, refined_template) / np.dot(refined_template,
            refined_template)
        ax.plot(seg_t, a * refined_template, 'r--', lw=2,
                label='Signal' if i == 0 else "")

    plt.xlabel('$t$')
    plt.ylabel('$A$')
    plt.legend(frameon=False)
    plt.tick_params(axis="both", which="both", direction="in")
    plt.savefig(f'signal{N}.pdf', dpi = 1080)
    plt.show()
```

Listing 2: *signal.py*