Засоби підготовки та аналізу даних

Лабораторна робота 5

ФБ-24 Воровська Єва

Посилання на репозиторій (GitHub repository):
https://github.com/yevavorov/ad_labs.git

Мета роботи: отримати поглиблені навички з візуалізації даних; ознайомитись з matplotlib.widgets, scipy.signal.filters, а також з Plotly, Bokeh, Altair; отримати навички зі створення інтерактивних застосунків для швидкого підбору параметрів і аналізу отриманих результатів

Постановка задачі

Створіть програму, яка дозволить користувачам малювати графік функції гармоніки (функція виду `y(t) = A * sin(ω * t + ϕ)`) з накладеним шумом та надавати можливість змінювати параметри гармоніки та шуму за допомогою інтерактивного інтерфейсу, що включає в себе слайдери, кнопки та чекбокси. Зашумлену гармоніку відфільтруйте за допомогою фільтру на вибір, порівняйте результат.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button, CheckButtons
from bokeh.layouts import column, row
from bokeh.models import ColumnDataSource, Slider, Button, CheckboxGroup,
Dropdown
from bokeh.plotting import figure
from bokeh.server.server import Server
from scipy import signal
# ^ Завантаження бібліотек ^
```

Завдання 1

1. Створіть програму, яка використовує бібліотеки Matplotlib для створення графічного інтерфейсу.

2. Реалізуйте функцію *harmonic_with_noise*, яка приймає наступні параметри:

- *amplitude* – амплітуда гармоніки.
- *frequency* – частота гармоніки.
- *phase* – фазовий зсув гармоніки.
- *noise_mean* – амплітуда шуму.
- *noise_covariance* – дисперсія шуму
- *show_noise* – флаг, який вказує, чи слід показувати шум на графіку.

3. У програмі має бути створено головне вікно з такими елементами інтерфейсу:

- Поле для графіку функції (plot).
- Слайдери (sliders), які відповідають за амплітуду, частоту гармоніки, а також слайдери для параметрів шуму.
- Чекбокс для перемикання відображення шуму на гармоніці.
- Кнопка «Reset», яка відновлює початкові параметри.

4. Програма повинна мати початкові значення кожного параметру, а також передавати параметри для відображення оновленого графіку.

5. Через чекбокс користувач може вмикати або вимикати відображення шуму на графіку. Якщо прапорець прибрано – відображати «чисту гармоніку», якщо ні – зашумлену.

6. Після оновлення параметрів програма повинна одразу оновлювати графік функції гармоніки з накладеним шумом згідно з виставленими параметрами.

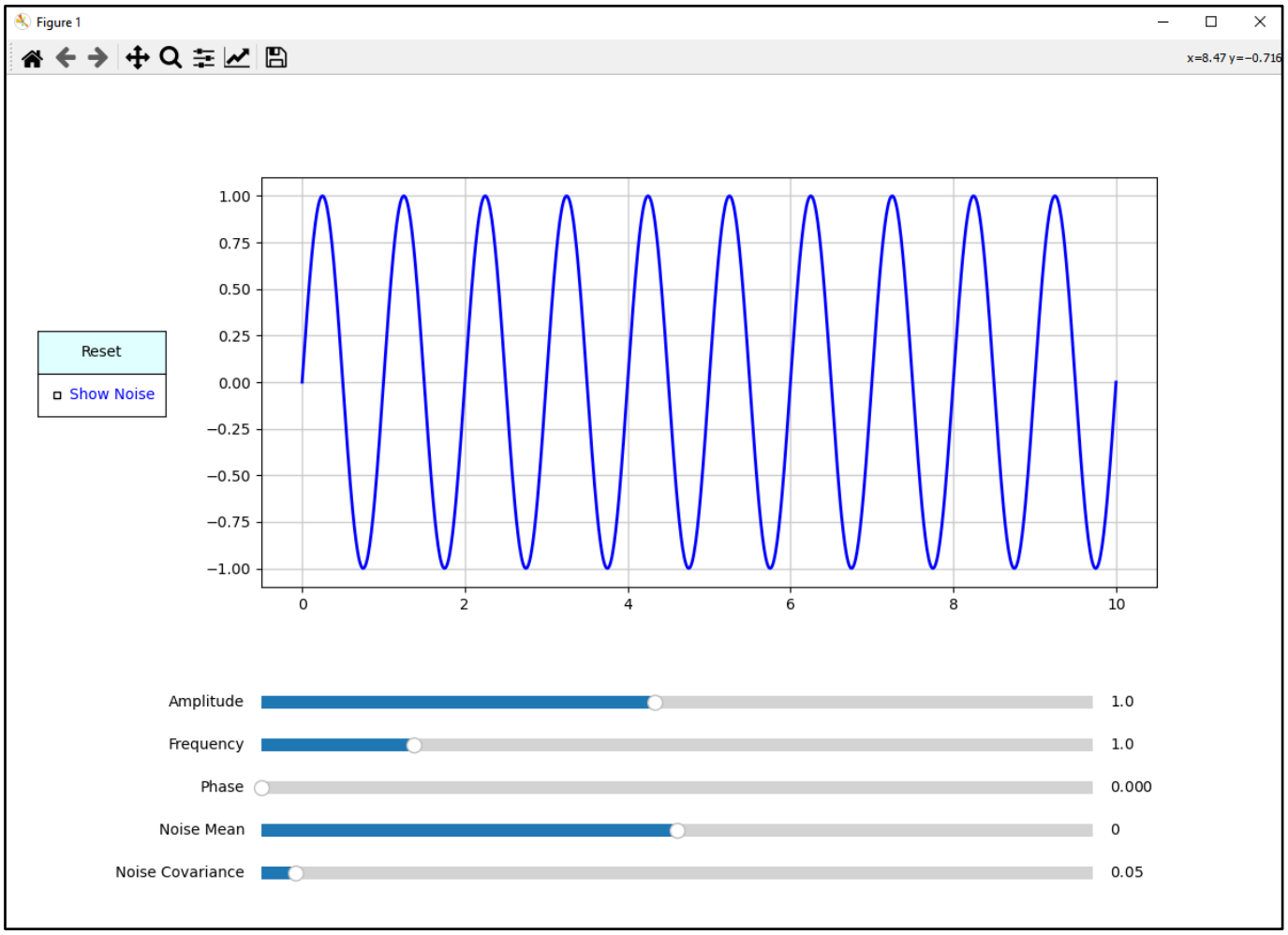
- Зауваження: якщо ви змінили параметри гармоніки, але не змінювали параметри шуму, то шум має залишитись таким як і був, а не генеруватись наново. Якщо ви змінили параметри шуму, змінюватись має лише шум – параметри гармоніки мають залишатись незмінними.

7. Після натискання кнопки «Reset», мають відновитись початкові параметри.

8. Залиште коментарі та інструкції для користувача, які пояснюють, як користуватися програмою.

9. Завантажте файл зі скриптом до вашого репозиторію на GitHub.

10. Надайте короткий звіт про ваш досвід та вивчені навички.



[laba5_1]

Код:

```
t = np.linspace(0, 10, 1000)

# Дефолтні параметри
default_amplitude = 1
default_frequency = 1
default_phase = 0
default_noise_mean = 0
default_noise_covariance = 0.05
default_show_noise = False

last_noise_mean = default_noise_mean
last_noise_covariance = default_noise_covariance

# Шум
noise = np.random.multivariate_normal([default_noise_mean],
[[default_noise_covariance]], len(t))[:,0]

def generate_noise(mean, covariance, length):
    global noise
    noise = np.random.multivariate_normal(mean, covariance, length)[:,0]
```

```
# Функція
def harmonic_with_noise(t, amplitude, frequency, phase, noise_mean,
noise_covariance, show_noise):
    graph = amplitude * np.sin(2 * np.pi * frequency * t + phase)
    global last_noise_mean
    global last_noise_covariance
    if show_noise:
        if (noise_mean != last_noise_mean) or (noise_covariance !=
last_noise_covariance):
            generate_noise(noise_mean, noise_covariance, len(t))
        last_noise_mean = noise_mean
        last_noise_covariance = noise_covariance
        return graph + noise
    else:
        last_noise_mean = noise_mean
        last_noise_covariance = noise_covariance
        return graph

fig, ax = plt.subplots(figsize=(12, 8))
line, = ax.plot(t, harmonic_with_noise(t, default_amplitude,
default_frequency, default_phase, default_noise_mean,
default_noise_covariance, default_show_noise), linewidth=2, color="blue")
fig.subplots_adjust(left=0.2, bottom=0.4)


# Слайдери
ax_amplitude = plt.axes([0.2, 0.25, 0.65, 0.03])
ax_frequency = plt.axes([0.2, 0.2, 0.65, 0.03])
ax_phase = plt.axes([0.2, 0.15, 0.65, 0.03])
ax_noise_mean = plt.axes([0.2, 0.1, 0.65, 0.03])
ax_noise_covariance = plt.axes([0.2, 0.05, 0.65, 0.03])

s_amplitude = Slider(ax_amplitude, "Amplitude", 0.1, 2.0,
valinit=default_amplitude)
s_frequency = Slider(ax_frequency, "Frequency", 0.1, 5.0,
valinit=default_frequency)
s_phase = Slider(ax_phase, "Phase", 0, 2*np.pi, valinit=default_phase)
s_noise_mean = Slider(ax_noise_mean, "Noise Mean", -1, 1,
valinit=default_noise_mean)
s_noise_covariance = Slider(ax_noise_covariance, "Noise Covariance",
0.01, 1, valinit=default_noise_covariance)


# Чекбокс
ax_show_noise = plt.axes([0.025, 0.6, 0.1, 0.05])
check_show_noise = CheckButtons(ax_show_noise, ["Show Noise"],
[default_show_noise], label_props={"color": "blue"})


# Update (оновлення параметрів)
def update(val):
    amplitude = s_amplitude.val
```

```
    frequency = s_frequency.val
    phase = s_phase.val
    noise_mean = [s_noise_mean.val]
    noise_covariance = [[s_noise_covariance.val]]
    show_noise = check_show_noise.get_status()[0]
    line.set_ydata(harmonic_with_noise(t, amplitude, frequency, phase,
noise_mean, noise_covariance, show_noise))
    fig.canvas.draw_idle()

s_amplitude.on_changed(update)
s_frequency.on_changed(update)
s_phase.on_changed(update)
s_noise_mean.on_changed(update)
s_noise_covariance.on_changed(update)
check_show_noise.on_clicked(update)


# Reset (скидання параметрів)
def reset(event):
    s_amplitude.reset()
    s_frequency.reset()
    s_phase.reset()
    s_noise_mean.reset()
    s_noise_covariance.reset()

button_reset_ax = plt.axes([0.025, 0.65, 0.1, 0.05])
button_reset = Button(button_reset_ax, "Reset", color="lightcyan",
hovercolor="0.95")
button_reset.on_clicked(reset)


# Графік
ax.grid(color="0.8", linewidth=1)
plt.show()
```

Завдання 2

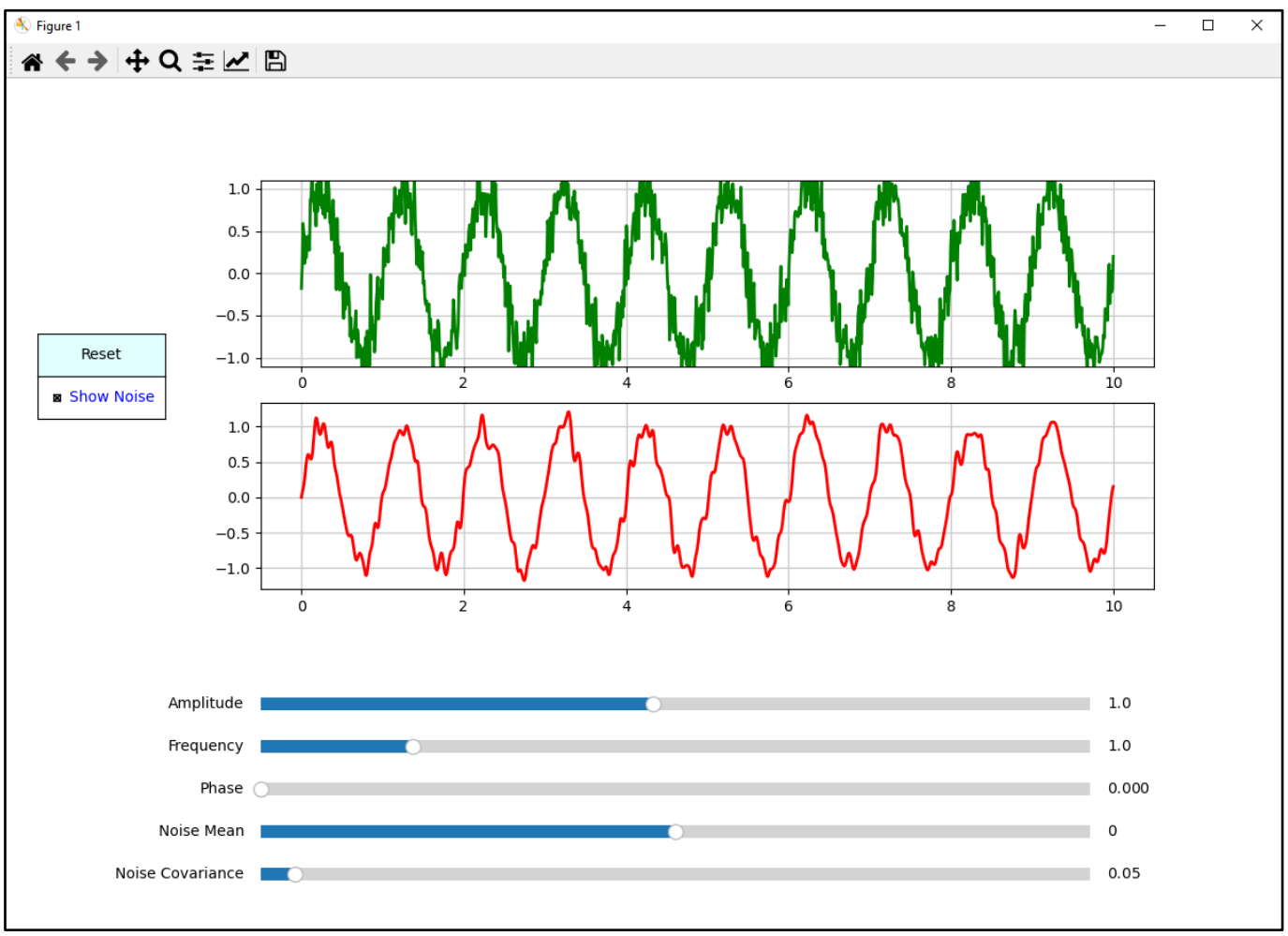
1. Отриману гармоніку з накладеним на неї шумом відфільтруйте за допомогою фільтру на ваш вибір (наприклад scipy.signal.iirfilter, повний список за посиланням:

https://docs.scipy.org/doc/scipy/reference/signal.html.

Відфільтрована гармоніка має бути максимально близька до «чистої».

2. Відобразіть відфільтровану «чисту» гармоніку поряд з початковою.

3. Додайте відповідні інтерактивні елементи (чекбокс показу, параметри фільтру тощо) та оновіть існуючі: відфільтрована гармоніка має оновлюватись разом з початковою.



[laba5_2]

↑ *Змінені фрагменти:*

[…]

```
# Відфільтрований шум
def harmonic_with_noise_filtered(t, amplitude, frequency, phase):
    graph = amplitude * np.sin(2 * np.pi * frequency * t + phase)
    b, a = signal.iirfilter(4, frequency / 5, btype="lowpass")
    filtered_noise = signal.lfilter(b, a, noise)
```

```
    return graph + filtered_noise

fig, ax = plt.subplots(2, figsize=(12, 8))
line, = ax[0].plot(t, harmonic_with_noise(t, default_amplitude,
default_frequency, default_phase, default_noise_mean,
default_noise_covariance, default_show_noise), linewidth=2,
color="green")
line2, = ax[1].plot(t, harmonic_with_noise_filtered(t, default_amplitude,
default_frequency, default_phase), linewidth=2, color="red")
fig.subplots_adjust(left=0.2, bottom=0.4)
```

[...]

```
    line2.set_ydata(harmonic_with_noise_filtered(t, amplitude, frequency,
phase))
```

[...]

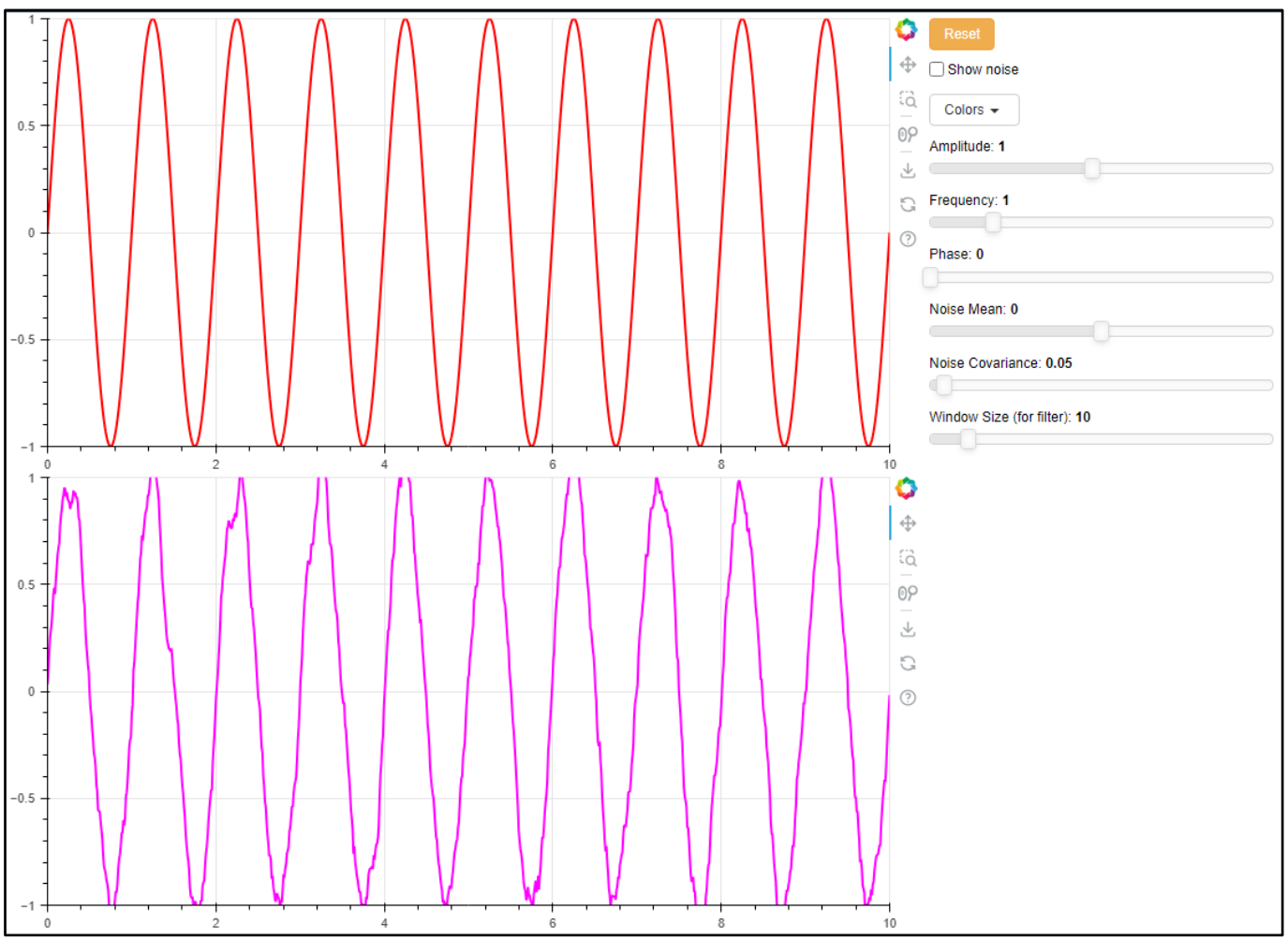
```
ax[0].grid(color="0.8", linewidth=1)
ax[1].grid(color="0.8", linewidth=1)
```

[...]

Завдання 3

1. Реалізуйте завдання 1 за допомогою сучасних графічних бібліотек на ваш вибір: Plotly, Bokeh, Altair тощо. Додайте декілька вікон для візуалізації замість одного, спадне меню (drop-down menu) та інші інтерактивні елементи на власний розсуд.

2. Реалізуйте ваш власний фільтр, використовуючи виключно Python (а також numpy, але виключно для операцій з масивами numpy.ndarray). Застосуйте фільтр



[laba5_3]

Код:

```
t = np.linspace(0, 10, 1000)

# Дефолтні параметри
default_amplitude = 1
default_frequency = 1
default_phase = 0
default_noise_mean = 0
default_noise_covariance = 0.05
default_show_noise = False
default_color = "red"
default_window_size = 10
```

```
last_noise_mean = default_noise_mean
last_noise_covariance = default_noise_covariance

# Шум
noise = np.random.multivariate_normal([default_noise_mean],
[[default_noise_covariance]], len(t))[:,0]


def all_code(doc):
    # Функція
    def generate_noise(mean, covariance, length):
        global noise
        noise = np.random.multivariate_normal([mean], [[covariance]],
length)[:,0]

    def harmonic_with_noise(t, amplitude, frequency, s_phase, noise_mean,
noise_covariance, show_noise):
        graph = amplitude * np.sin(2 * np.pi * frequency * t + s_phase)
        global last_noise_mean
        global last_noise_covariance
        if show_noise:
            if (noise_mean != last_noise_mean) or (noise_covariance !=
last_noise_covariance):
                generate_noise(noise_mean, noise_covariance, len(t))
            last_noise_mean = noise_mean
            last_noise_covariance = noise_covariance
            return graph + noise
        else:
            last_noise_mean = noise_mean
            last_noise_covariance = noise_covariance
            return graph

    # Відфільтрований шум (свій)
    def harmonic_with_noise_filtered(t, amplitude, frequency, s_phase,
w):
        graph = amplitude * np.sin(2 * np.pi * frequency * t + s_phase)
        filtered_noise = np.zeros_like(noise)
        window_size = w
        for i in range(len(noise)):
            start_index = max(0, i - window_size // 2)
            end_index = min(len(noise), i + window_size // 2 + 1)
            filtered_noise[i] = np.mean(noise[start_index:end_index])
        return graph + filtered_noise


    source = ColumnDataSource(data=dict(x=t, y=harmonic_with_noise(t,
default_amplitude, default_frequency, default_phase, default_noise_mean,
default_noise_covariance, default_show_noise)))
```

```
    source2 = ColumnDataSource(data=dict(x=t,
y=harmonic_with_noise_filtered(t, default_amplitude, default_frequency,
default_phase, default_window_size)))

    # Графік
    p = figure(width=800, height=400, y_range=(-1, 1), x_range=(0, 10))
    line = p.line(source=source, color=default_color, line_width=2)
    p2 = figure(width=800, height=400, y_range=(-1, 1), x_range=(0, 10))
    line2 = p2.line(source=source2, color="magenta", line_width=2)


    # Слайдери
    s_amplitude = Slider(start=0.1, end=2.0, value=default_amplitude,
step=0.1, title="Amplitude")
    s_frequency = Slider(start=0.1, end=5.0, value=default_frequency,
step=0.1, title="Frequency")
    s_phase = Slider(start=0, end=2*np.pi, value=default_phase, step=0.1,
title="Phase")
    s_noise_mean = Slider(start=-1, end=1, value=default_noise_mean,
step=0.1, title="Noise Mean")
    s_noise_covariance = Slider(start=0.01, end=1,
value=default_noise_covariance, step=0.01, title="Noise Covariance")
    s_window_size = Slider(start=5, end=50, value=10, step=1,
title="Window Size (for filter)")

    # Чекбокс
    checkbox_show_noise = CheckboxGroup(labels=["Show noise"])

    # Dropdown
    menu = [("Red", "red"), ("Green", "green"), ("Blue", "blue")]
    dropdown_button = Dropdown(label="Colors", button_type="default",
menu=menu)


    # Update (оновлення параметрів)
    def update(attrname, old, new):
        amplitude = s_amplitude.value
        frequency = s_frequency.value
        phase = s_phase.value
        noise_mean = s_noise_mean.value
        noise_covariance = s_noise_covariance.value
        w = s_window_size.value
        if 0 in checkbox_show_noise.active:
            show_noise = True
        else:
            show_noise = False
        source.data = dict(x=t, y=harmonic_with_noise(t, amplitude,
frequency, phase, noise_mean, noise_covariance, show_noise))
        source2.data = dict(x=t, y=harmonic_with_noise_filtered(t,
amplitude, frequency, phase, w))
```

```
    def color_update(event):
        new_color = event.item
        line.glyph.line_color = new_color

    s_amplitude.on_change("value", update)
    s_frequency.on_change("value", update)
    s_phase.on_change("value", update)
    s_noise_mean.on_change("value", update)
    s_noise_covariance.on_change("value", update)
    s_window_size.on_change("value", update)
    checkbox_show_noise.on_change("active", update)
    dropdown_button.on_click(color_update)


    # Reset (скидання параметрів)
    def reset(event):
        s_amplitude.value = default_amplitude
        s_frequency.value = default_frequency
        s_phase.value = default_phase
        s_noise_mean.value = default_noise_mean
        s_noise_covariance.value = default_noise_covariance
        s_window_size.value = default_window_size
        checkbox_show_noise.active = []
        line.glyph.line_color = default_color
        source.data = dict(x=t, y=harmonic_with_noise(t,
default_amplitude, default_frequency, default_phase, default_noise_mean,
default_noise_covariance, default_show_noise))
        source2.data = dict(x=t, y=harmonic_with_noise_filtered(t,
default_amplitude, default_frequency, default_phase,
default_window_size))

    button_reset = Button(label="Reset", button_type="warning")
    button_reset.on_event("button_click", reset)


    doc.add_root(row(column(p, p2), column(button_reset,
checkbox_show_noise, dropdown_button, s_amplitude, s_frequency, s_phase,
s_noise_mean, s_noise_covariance, s_window_size)))


server = Server({"/": all_code})
server.start()

if __name__ == "__main__":
    server.io_loop.add_callback(server.show, "/")
    server.io_loop.start()
```