Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

ЗВІТ
З ВИРОБНИЧОЇ ПРАКТИКИ

Виконала: студентка групи ПМІ-21
спеціальності 122 – комп'ютерні науки
Ткачова Марія Володимирівна

Львів 2023

https://github.com/mashatkk/lagger

1. Побудувати функцію для обчислення значення функції Лаґерра за формулою (1.2) для довільних t і n, а параметри задавати за замовчуванням $\beta = 2, \sigma = 4$.

Спочатку імпортуємо бібліотеки NumPy та Pandas.

[1]:
```
import numpy as np
import pandas as pd
```

Функція обчислює значення функції Лагерра для заданих t і n, використовуючи параметри за замовчуванням $\beta = 2$, та $\sigma = 4$.

Перевірка параметрів β і σ: Перевіряємо, чи значення параметра β знаходиться в діапазоні від 0 до σ. Якщо β менше 0 або більше, ніж σ, генерується виняток ValueError з повідомленням про помилку "Wrong parameters". Обчислення початкових значень lpp та lp: Тут обчислюються початкові значення для lpp та lp на основі переданих параметрів t, βта σ. Перевірка значень n для випадків $n = 0$ та $n = 1$: Якщо n дорівнює 0 або 1, функція повертає відповідно значення lpp або lp. Обчислення значень для $n > 1$ за допомогою циклу for: У цьому циклі виконуються обчислення для lp та lpp на основі попередніх значень lp та lpp для значень $n > 1$. Кожна ітерація циклу оновлює lp та lpp, використовуючи формулу для функції Лагерра. Повернення результату: У кінці функції повертається значення lp, обчислене в останній ітерації циклу for.

[2]:
```
def lagger(t, n, beta=2, omega=4):
    if beta < 0 or beta > omega:
        raise ValueError("Wrong parameters")


    lpp = np.sqrt(omega)*np.exp(-beta*t/2)

    lp = np.sqrt(omega)*(1 - omega*t)*np.exp(-beta*t/2)

    if n == 0:
        return lpp
    if n == 1:
        return lp

    for i in range(2, n+1):
        temp = lp
        lp = (2*i -1 -omega*t)*lp/i - (i-1)*lpp / i
        lpp = temp

    return lp
```

Виклик функції lagger, яка обчислює значення функції Лагерра для заданих параметрів t, n, β та σ. Це обчислить значення функції Лагерра для t=1 та n=3 з параметрами за замовчуванням β =2 та σ=4.

[3]:
```
lagger(1, 3)
```

[3]:
```
1.7167707254667308
```

2. Побудувати функцію для табулювання при заданих n, β, σ функції Лаґерра на відрізку [0,T] із заданим $T \in \mathbb{R}_+$.

Функція tabulate_lagger табулює значення функції Лагерра на відрізку[0,T] для заданого T з параметрами n, β і σ.

Функція np.linspace створює 100 рівномірно розподілених значень від 0 до T. Це формує вектор t, що містить значення від 0 до T у 100 точках.

Функція lagger викликається з вектором t, n, β, і σ. lagger обчислює значення функції Лагерра для кожної точки t на відрізку [0, T] з заданими параметрами.

З результатів обчислень створюється об'єкт DataFrame з бібліотеки Pandas, де стовпець 't' містить значення t, а стовпець 'l' містить обчислені значення функції Лагерра.

Повертається об'єкт DataFrame, в якому всі значення округлені до 5 знаків після коми.

Отже, ця функція використовує функцію lagger для обчислення значень функції Лагерра на відрізку [0, T] з вказаними параметрами n, βі σ, після чого результати утворюють DataFrame, який потім повертається як вихід з функції.

[4]:
```
def tabulate_lagger(T, n, beta, gamma):#оголошення функції tabulate_lagger з␣
↪параметрами T, n, beta, і gamma.
    t = np.linspace(0, T, 100)
    results = lagger(t, n, beta, gamma)
    df = pd.DataFrame({'t': t, 'l': results})
    return df.round(5)
```

Функція tabulate_lagger(10, 2, 2, 4) обчислить значення функції Лагерра для відрізка від 0 до 10 з параметрами n=2, β=2 і σ=4.

Це створить DataFrame з двома стовпцями: 't', що містить 100 значень рівномірно розподілених на відрізку [0, 10], та 'l', який міститиме обчислені значення функції Лагерра для кожної відповідної точки 't'.

Результатом виклику функції буде об'єкт DataFrame, що містить пари значень 't' та відповідних значень функції Лагерра 'l' для заданих параметрів.

[5]:
```
tabulate_lagger(10, 2, 2, 4)
```

[5]:

	t	l
0	0.00000	2.00000
1	0.10101	0.49453
2	0.20202	-0.47336
3	0.30303	-1.01868
4	0.40404	-1.23687
..	...	...
95	9.59596	0.08989
96	9.69697	0.08307
97	9.79798	0.07675
98	9.89899	0.07089
99	10.00000	0.06547

[100 rows x 2 columns]

3. Провести обчислювальний експеримент: для N = 20 на основі графіків з п.2 знайти точку T>0, щоб |ln(T)|<ε=10^{-3}

для усіх n ∈ $[0, N]$

Побудувати табличку для |ln(T)| для усіх n ∈ $[0, N]$.

Генерація точок t: t = np.linspace(0, T, 1000) генерує 1000 значень на відрізку від 0 до T. Ці значення будуть використані для подальшого обчислення значень функції Лагерра. Створення діапазону n: n = range(1, N+1) створює послідовність чисел від 1 до N. Ці числа представлятимуть показники n для функції Лагерра.

Перевірка значень функції Лагерра для кожної точки t та n: Вкладений цикл перевіряє значення функції Лагерра для кожного n від 1 до N для кожної точки t. Якщо будь-яке з цих значень більше за epsilon, змінна flag встановлюється на False, і цикл обривається. Це робиться для кожної точки t, поки не знайдеться така, де всі значення функції Лагерра для n менші за epsilon.

Знаходження точки T: Якщо всі значення функції Лагерра для всіх n менші за epsilon для певної точки t і result ще не був знайдений, то result встановлюється як значення цієї точки t.

Створення DataFrame: Формується DataFrame, де стовпці відповідають кожному n і містять значення функції Лагерра для відповідних точок t.

Повернення результату: Повертається знайдене значення T та DataFrame зі значеннями функції Лагерра для кожного n на всьому відрізку t.

Отже, ця функція шукає таку точку T, для якої всі значення функції Лагерра для n від 0 до N менші за epsilon.

[6]:
```
def experiment(T, beta, gamma, epsilon=1e-3, N=20):
    t = np.linspace(0, T, 1000)
    n = range(1, N+1)
    result = None
    for i in t:
        flag = True
        for j in n:
            if abs(lagger(i, j, beta, gamma)) > epsilon:
                flag = False
                break
        if flag and result is None:
            result = i


    cols = {"t" : t}
    for j in n:
        cols[f"n={j}"] = lagger(t, j, beta, gamma)

    df = pd.DataFrame(cols)

    return result, df.round(5)
```

Виклик функції experiment з аргументами (100, 2, 4), тобто T = 100, β= 2, і σ = 4. Функція повертає два значення: r та df.

r - це знайдена точка T, де всі значення функції Лагерра для n від 0 до N менші за epsilon. Це значення представлене як результат змінної result у функції experiment.

df - це DataFrame, що містить значення функції Лагерра для кожного n на відрізку від 0 до T, що було знайдено. Кожен стовпець у цьому DataFrame представляє значення функції Лагерра для певного n.

Отже, результати обчислювального експерименту зберігаються в змінних r та df і можуть бути використані для подальшого аналізу або візуалізації.

[7]:
```
r, df = experiment(100, 2, 4)
```

представляє собою знайдену точку T, де всі значення функції Лагерра для n від 0 до N менші за ε.

[8]:
```
r
```

[8]:
```
79.07907907907908
```

При виклику df можна побачити табличку з числами, які представляють значення функції Лагерра для кожного n на відрізку від 0 до знайденої точки T. Кожен стовпець буде містить числа для певного n, а рядки відповідають кожній точці t на цьому відрізку. Така таблична форма дозволяє аналізувати та використовувати ці значення для подальших обчислень чи візуалізації.

[9]:
```
df
```

[9]:

	t	n=1	n=2	n=3	n=4	n=5	n=6	n=7	\
0	0.0000	2.00000	2.00000	2.00000	2.00000	2.00000	2.00000	2.00000	
1	0.1001	1.08497	0.50550	0.05172	-0.29380	-0.54668	-0.72094	-0.82906	
2	0.2002	0.32612	-0.45997	-0.86125	-0.98978	-0.93409	-0.76300	-0.52907	
3	0.3003	-0.29802	-1.00863	-1.07852	-0.80705	-0.39600	0.02583	0.38297	
4	0.4004	-0.80621	-1.23375	-0.86012	-0.23551	0.33962	0.72824	0.89472	
..	...	...	...	...	...	...	...	...	
995	99.5996	-0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
996	99.6997	-0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
997	99.7998	-0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
998	99.8999	-0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
999	100.0000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	0.00000	-0.00000	

	n=8	n=9	...	n=11	n=12	n=13	n=14	n=15	\
0	2.00000	2.00000	...	2.00000	2.00000	2.00000	2.00000	2.00000	
1	-0.88217	-0.89014	...	-0.80442	-0.72510	-0.62955	-0.52282	-0.40925	
2	-0.27143	-0.01825	...	0.40417	0.55421	0.65857	0.71780	0.73477	
3	0.63795	0.77947	...	0.75506	0.62619	0.44937	0.24662	0.03764	
4	0.86127	0.67826	...	0.09748	-0.19735	-0.44519	-0.62440	-0.72499	
..	...	...	...	...	...	...	...	...	
995	0.00000	-0.00000	...	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
996	0.00000	-0.00000	...	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
997	0.00000	-0.00000	...	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
998	0.00000	-0.00000	...	-0.00000	0.00000	-0.00000	0.00000	-0.00000	
999	0.00000	-0.00000	...	-0.00000	0.00000	-0.00000	0.00000	-0.00000	

	n=16	n=17	n=18	n=19	n=20
0	2.00000	2.00000	2.00000	2.00000	2.00000
1	-0.29253	-0.17579	-0.06163	0.04782	0.15085
2	0.71390	0.66063	0.58092	0.48093	0.36668
3	-0.16110	-0.33677	-0.48021	-0.58574	-0.65081
4	-0.74672	-0.69683	-0.58770	-0.43478	-0.25469
..	...	...	...	...	...
995	0.00000	-0.00000	0.00000	-0.00000	0.00000

```
996  0.00000 -0.00000  0.00000 -0.00000  0.00000
997  0.00000 -0.00000  0.00000 -0.00000  0.00000
998  0.00000 -0.00000  0.00000 -0.00000  0.00000
999  0.00000 -0.00000  0.00000 -0.00000  0.00000

[1000 rows x 21 columns]
```

4. Побудувати функцію для обчислення значень інтегралів (1.3) наближено за формулою

$$f_k = \int_0^T f(t) l_k(t) e^{-\alpha t}\, dt, k \in [0, N],$$

використовуючи метод прямокутників із заданою точністю ε> 0.

Функція quad реалізує метод прямокутників для наближеного обчислення інтегралу від функції f на відрізку [a,b] з використанням N прямокутників.

Основні кроки функції:

Генерується послідовність значень x на відрізку [a,b] з N рівновіддаленими точками, які використовуються для обчислення функції f в цих точках.

Обчислюється сума значень функції f в усіх точках x.

Повертається наближене значення інтегралу, яке обчислюється як сума значень функції, помножених на відношення довжини відрізка [a,b] до кількості прямокутників N.

[10]:
```
def quad(f, a, b, N=10000):

    x = np.linspace(a, b, N)
    s = sum([f(i) for i in x])
    return s*abs(b-a)/N
```

Функція lagger_transformation використовує метод прямокутників (за допомогою функції quad) для наближеного обчислення інтегралу, визначеного за формулою (1.6).

Основні кроки функції: 1.Визначається функція integrand(використовується як інтегрант для обчислення інтегралу), яка представляє собою добуток f(t), lagger(t, n, β, σ), та (exp($-t(\sigma - \beta)$.

2.Викликається функція experiment для знаходження точки b (верхньої межі інтегрування) для заданих параметрів βі σ.

3.Викликається функція quad для обчислення наближеного значення інтегралу від integrand на відрізку від 0 до b.

Отже, ця функція lagger_transformation взаємодіє з іншими функціями (такими як lagger та experiment) для обчислення інтегралу, використовуючи метод прямокутників, з точністю, яка визначається параметром N у функції quad.

[11]:
```
def lagger_transformation(f, n, beta=2, gamma=4):
    def integrand(t):
        return f(t)*lagger(t, n, beta, gamma)*np.exp(-t*(gamma-beta))
    b = experiment(100, beta, gamma)[0]
```

```
    return quad(integrand, 0, b)
```

Це простий тест для функції 'quad', який обчислює чисельний інтеграл $\int_0^{100} e^{-t^2/2}$

Отримані числові результати можуть дати приблизне уявлення про те, як змінюється результат інтегрування при різних значеннях 'N' (кількість прямокутників). Чим більше значення 'N', тим точніше буде наближення до точного значення інтегралу.

Перший виклик 'quad(func, 0, 100, 1000)' обчислює інтеграл, розбиваючи відрізок [0, 100] на 1000 прямокутників, другий - на 10000, а третій - на 100000.

Якщо ви хочете порівняти точність апроксимації інтегралу для різних значень 'N', ви можете порівняти отримані числові результати цих викликів. Чим більше 'N', тим більша точність може бути досягнута, але це також може вимагати більшого обчислювального часу.

[12]:
```
func = lambda t: np.exp(-t**2/2)
print(quad(func, 0, 100,1000))
print(quad(func, 0, 100,10000))
print(quad(func, 0, 100,100000))
```

```
1.3020608231781858
1.258188805901769
1.2538016041741151
```

5. Для функції

$$f(t) = \begin{cases} \frac{sin(t-\pi/2)+1}{,}, & t \in [0, 2\pi]; \\ 0, \ t \geq 2\pi \end{cases}$$

виконати ПЛ, а саме знайти коефіцієнти $\mathbf{f}^N := (f0, f1, ..., f^N)^T$ **при N = 20.**

Функція tabulate_transformation призначена для обчислення інтегралів з використанням методу прямокутників для кожного значення n у діапазоні від 1 до N. Основні кроки функції:

1.Створюється послідовність чисел від 1 до N для представлення значень n.

2.Для кожного n викликається функція lagger_transformation з використанням функції f, параметрів beta і gamma, обчислюючи інтеграл за допомогою методу прямокутників.

3.Результати цих інтегралів для кожного n зберігаються в results.

4.Повертається список results, який містить обчислені значення інтегралів для кожного n.

Ця функція дозволяє обчислити та зберегти значення інтегралів для функції f та параметрів beta і gamma з використанням методу прямокутників для різних значень n.

[13]:
```
def tabulate_transformation(f, N, beta, gamma):
    t = range(1, N+1)
    results = [lagger_transformation(f, n, beta, gamma) for n in t]

    return results
```

Функція f(t) приймає значення t і перевіряє, чи вони знаходяться в межах від 0 до 2π. Якщо t знаходиться у цьому діапазоні, функція обчислює значення (sin(t - π/2) + 1, відповідне для формули (1.7). Якщо t не знаходиться у діапазоні від 0 до 2π, функція повертає 0.

Це дозволяє використовувати функцію f(t) для обчислення значень sin(t - π/2) + 1 у межах від 0 до 2π, і при цьому обнуляє результат для всіх інших значень t.

[14]:
```
def f(t):
    if t >= 0 and t <= 2*np.pi:
        return np.sin(t-np.pi/2) + 1
    else:
        return 0
```

Виклик функції tabulate_transformation(f, 20, 2, 4) (для обчислення інтегралів за допомогою методу прямокутників для вказаних параметрів beta = 2,gamma = 4 та функції f(t)) поверне список значень інтегралів для кожного значення k від 1 до 20, де кожне значення представляє результат виклику функції lagger_transformation для вказаних параметрів і методу прямокутників.

[15]:
```
tabulate_transformation(f, 20, 2, 4)
```

[15]:
```
[-0.1822039881310192,
 0.17805610913078898,
 -0.0742826695000306,
 0.007262784325811699,
 0.007587430478937864,
 -0.0030964949450654838,
 -0.0006148703444646362,
 0.0007994250066752776,
 -2.585015381201144e-05,
 -0.00023592602139502894,
 5.2569600955046235e-05,
 9.38139027783229e-05,
 -3.067668357426933e-05,
 -5.2658350956100556e-05,
 1.097528013689054e-05,
 3.6161726975336044e-05,
 4.5476548832641705e-06,
 -2.3361494137766213e-05,
 -1.4823620942386276e-05,
 8.910978104645316e-06]
```

6. Побудувати функцію, яка для заданої послідовності $\mathbf{h}^N = (h0, h1, ..., h_k, ...h_N, 0, 0, ...)^T, N \in N$

(яка має скінчене число відмінних від нуля елементів) обчислює значення функції

$$\widetilde{h^N}(t) \tag{1}$$

у точці $\mathbf{t} \in \mathbb{R}_+$за формулою (1.4).

Функція reversed_lagger_transformation призначена для обчислення значення оберненого перетворення Лагерра для вказаної послідовності коефіцієнтів, яка має скінченну кількість ненульових елементів, у точці t за використання методу Лагерра.

Основні кроки функції:

1.Створюється новий список h_list_new, який містить тільки ненульові коефіцієнти з вихідного списку h_list.

2.Для кожного ненульового коефіцієнта hk обчислюється значення hk* Lk(t), де Lk(t) - поліном Лагерра з параметрами beta і gamma для кожного k.

3.Результати цих добутків для кожного ненульового hk сумуються разом у result_sum.

4.Повертається значення суми, яке представляє обчислене значення оберненого перетворення Лагерра для заданої у точці t послідовності коефіцієнтів h_N

Ця функція дозволяє обчислити значення оберненого перетворення Лагерра для заданої послідовності коефіцієнтів у точці t, використовуючи метод Лагерра.

[16]:
```
def reversed_lagger_transformation(h_list, t, beta=2, gamma=4):
    result_sum = 0

    h_list_new = list(filter(lambda x: x != 0, h_list))

    for i in range(len(h_list_new)):
        result_sum += h_list_new[i]*lagger(t, i, beta, gamma)

    return result_sum
```

Функція f_test(t) просто повертає вхідний аргумент t. Це означає, що значення, яке ви передаєте в цю функцію, повертається без будь-яких змін чи обробки. Ця функція може використовуватися для тестування інших функцій, де потрібно передати аргумент, але не потрібно нічого з ним робити або обробляти.

[17]:
```
def f_test(t):
    return t
```

1.Виклик функції tabulate_transformation для функції f_test з параметрами N=20, beta=2 і gamma=4. Це обчислює послідовність інтегралів методом прямокутників для функції f_test та заданих параметрів.

2.Виклик функції reversed_lagger_transformation з обчисленими значеннями інтегралів transfomormed_temp (отриманими після перетворення функції f_test) та іншими параметрами t=1, beta=2 і gamma=4. Це обчислює обернене перетворення Лагерра для отриманих значень інтегралів у точці t=1 з вказаними параметрами beta і gamma.

Отже, код виконує послідовність обчислень значень інтегралів методом прямокутників для функції f_test та їх подальше обернене перетворення Лагерра у точці t=1 за заданими параметрами.

[18]:
```
transfomormed_temp = tabulate_transformation(f_test, 20, 2, 4)
reversed_lagger_transformation(transfomormed_temp, 1, 2, 4)
```

[18]:
```
-0.7776771623336753
```

7. За даними завдання 2 побудувати графіки функцій Лаґерра ln(t), t $\in [0, T], n \in [0, N]$.

Цей код малює графіки поліномів Лагерра для заданих значень N, T, beta і gamma. Ось кроки, які він виконує:

1.Створення фігури та осей: ‘fig = plt.figure(figsize=(10, 10))‘ створює фігуру для малювання графіків, ‘ax = fig.gca()‘ створює осі.

2.Будування графіків поліномів Лагерра: Цикл ‘for n in range(N+1):‘ обходить значення n від 0 до N. Для кожного значення n обчислюється набір значень полінома Лагерра за допомогою функції ‘tabulate_lagger(T, n, beta, gamma)‘. Після цього графік кожного полінома Лагерра будується за допомогою ‘ax.plot(...)‘, де ‘lagger_values[’t’]‘ - значення t, а ‘lagger_values[’l’]‘ - відповідні значення полінома Лагерра для кожного t.

3.Оформлення графіку: Задається підпис осей, заголовок графіку та легенда (частина графіка, яка пояснює, що саме відображається на графіку) для різних ліній поліномів Лагерра .У даному випадку підписи легенди надають інформацію про значення n, щоб можна було відокремити і ідентифікувати кожен поліном Лагерра на графіку за його номером n. Це робить зручним порівняння різних кривих, що представляють поліноми Лагерра на графіку.

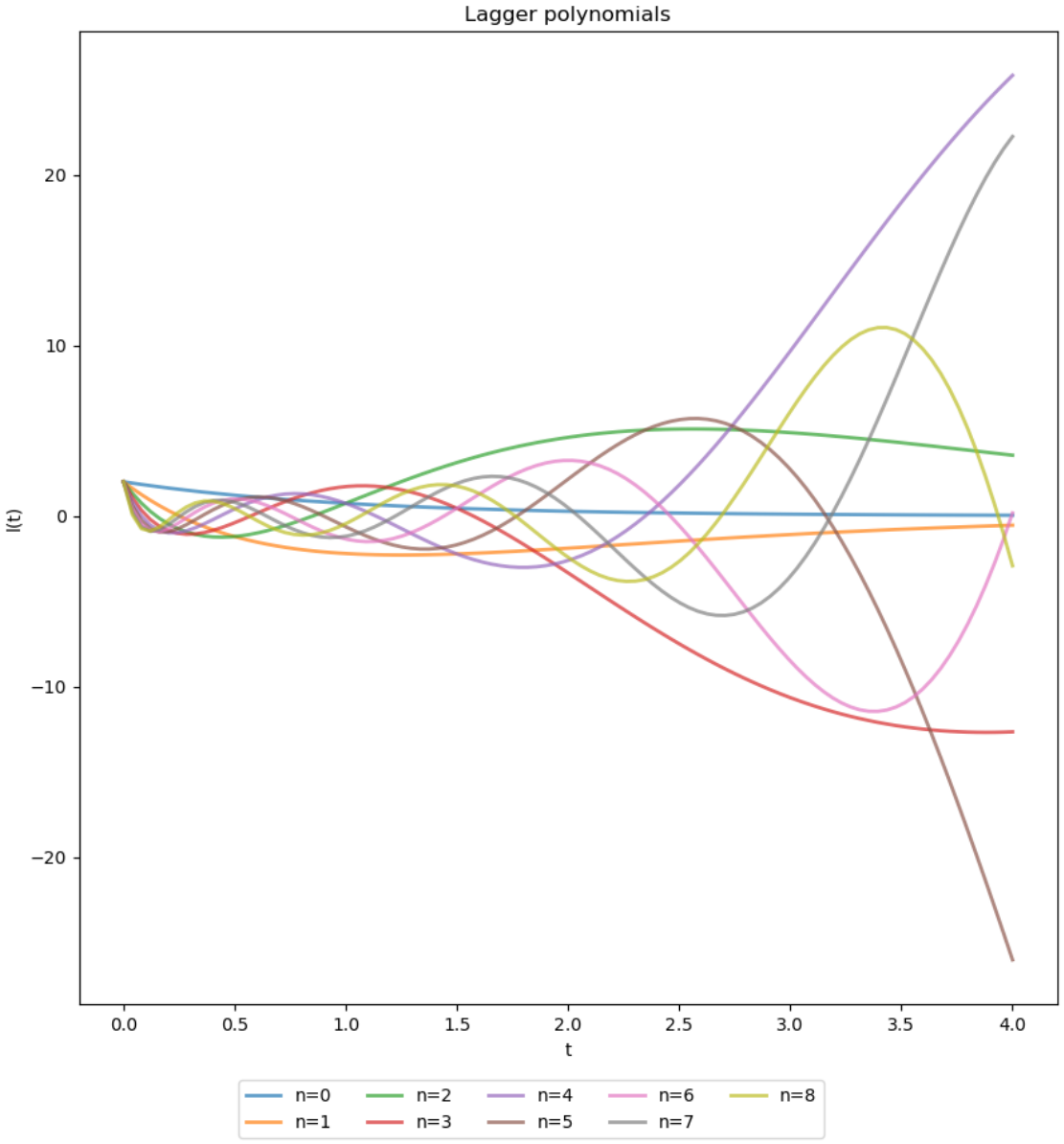
4.Показ графіка: ‘plt.show()‘ відображає побудований графік.

Отже, цей код малює графіки поліномів Лагерра для значень n від 0 до N на інтервалі [0, T]з використанням даних параметрів beta і gamma. Кожен графік позначений відповідним значенням n на легенді, що дозволяє візуально порівняти різні поліноми Лагерра на одному графіку.

[19]:
```
import matplotlib.pyplot as plt
def plot_lagger(T, N, beta=2, gamma=4):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.gca()
    for n in range(N+1):
        lagger_values = tabulate_lagger(T, n, beta, gamma)
        ax.plot(lagger_values['t'], lagger_values['l'], label=f"n={n}", linewidth=2.0,␣
 ↪alpha=0.7)

    ax.set_xlabel("t")
    ax.set_ylabel("l(t)")
    ax.set_title("Lagger polynomials")
    fig.legend(loc='lower center', ncol=5)
    plt.show()
```

Функція plot_lagger побудує графіки поліномів Лагерра для значень N = 8 на інтервалі [0, 4] за допомогою функції tabulate_lagger. Це дозволить візуально оцінити форму і зміну цих поліномів зі збільшенням їхніх індексів n.

[20]:
```
plot_lagger(4,8)
```


Lagger polynomials
20
10
0
−10
−20
l(t)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
t
n=0
n=1
n=2
n=3
n=4
n=5
n=6
n=7
n=8

8. Для функції (1.7) виконати пряме і обернене ПЛ при деяких значеннях N . Побудувати графік функції $\widetilde{f^N}(t)$, $t \in [0, 2\pi]$.

Функція plot_transformation створює стовпчикову діаграму для візуалізації прямого перетворення Лагерра для заданої функції f(t) та параметрів n, beta, gamma. Розглянемо, як працює цей код:

fig = plt.figure(figsize=(10, 10)): Створення фігури для малювання графіку розміром 10x10 дюймів.

ax = fig.gca(): Отримання поточних осей для подальшого малювання на них.

transform_values = tabulate_transformation(f, n, beta, gamma): Виклик функції tabulate_transformation для обчислення значень прямого перетворення Лагерра для функції f та вказаних параметрів n, beta, gamma.

ax.bar(range(1, n+1), transform_values, alpha=0.7, edgecolor='black'): Побудова стовпчикової діаграми, де вісь x відповідає значенням n, а вісь y відповідає значенням прямого перетворення Лагерра. Кожен стовпчик представляється значенням прямого перетворення для конкретного n.

ax.set_xlabel("n"), ax.set_ylabel("f_n"), ax.set_title("Transformation"): Задання підписів для осей та заголовку графіку.

ax.set_xticks(range(1, n+1)): Задання міток для вісі x для кожного значення n.

fig.tight_layout(): Автоматичне вирівнювання розміщення графіку.

plt.axhline(0, color='black'): Додавання горизонтальної лінії на рівні 0 для кращого розуміння графіку.

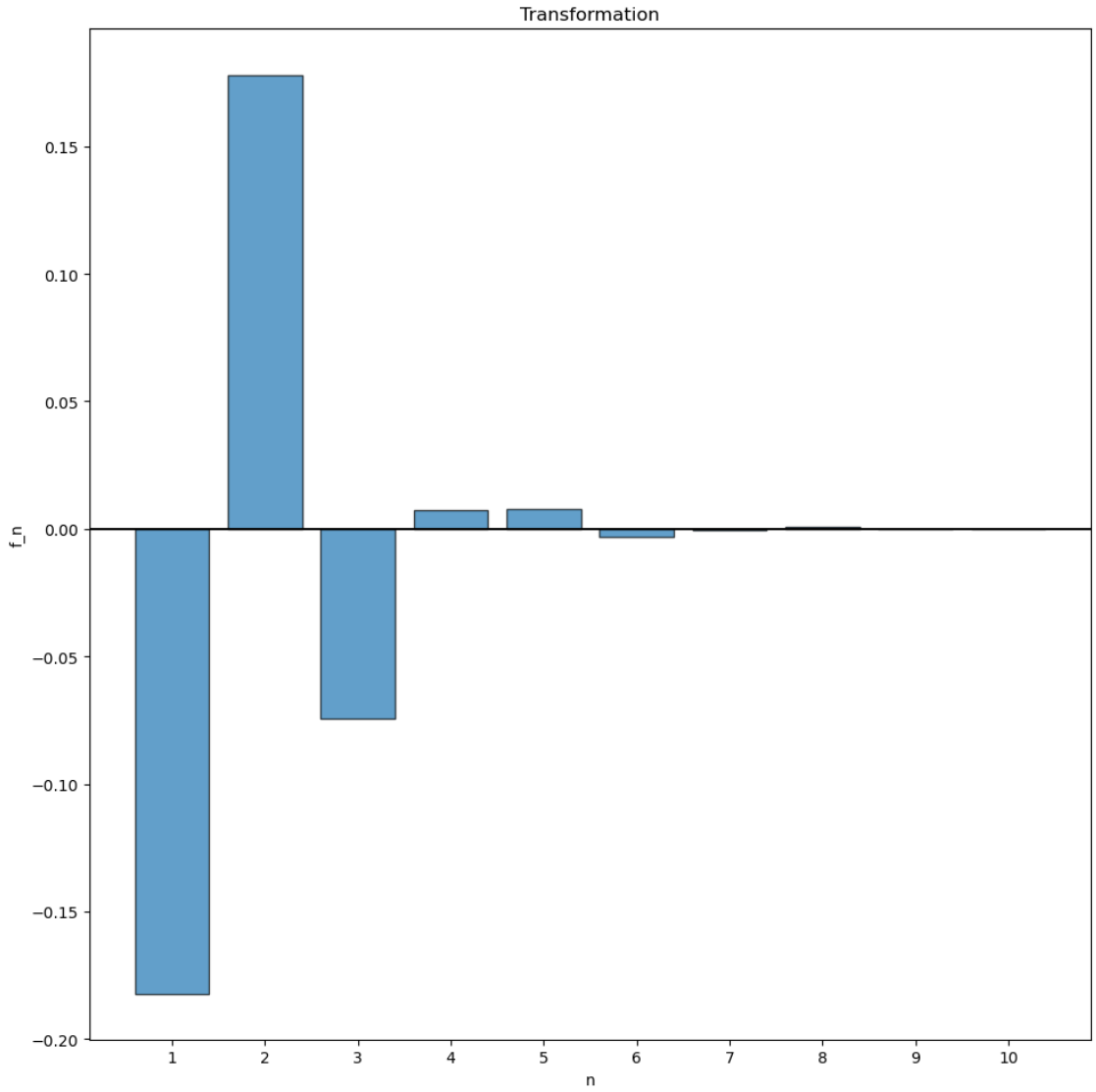
plt.show(): Відображення побудованого графіку.

Ця функція важлива для візуалізації результатів прямого перетворення Лагерра та допомагає зрозуміти, як змінюються коефіцієнти fn зі збільшенням n.

[21]:
```
def plot_transformation(f, n, beta=2, gamma=4):
    fig = plt.figure(figsize=(10, 10))
    ax = fig.gca()
    transform_values = tabulate_transformation(f, n, beta, gamma)
    ax.bar(range(1, n+1), transform_values, alpha=0.7, edgecolor='black')

    ax.set_xlabel("n")
    ax.set_ylabel("f_n")
    ax.set_title("Transformation")
    ax.set_xticks(range(1, n+1))
    fig.tight_layout()
    plt.axhline(0, color='black')
    plt.show()
```

Цей код викликає функцію plot_transformation з функцією f (sin(t - π/2) + 1) та параметром n=10. Функція plot_transformation будує стовпчикову діаграму для прямого перетворення Лагерра для вказаної функції та значення параметра n.

[22]:
```
plot_transformation(f, 10)
```



Функція plot_transformations призначена для візуалізації прямого та оберненого перетворення Лагерра для заданої функції (f(t)).

Основні кроки функції:

Пряме перетворення Лагерра: Обчислюються коефіцієнти прямого перетворення Лагерра за допомогою tabulate_transformation. Створюється стовпчикова діаграма, де по горизонтальній вісі відображаються значення (n), а по вертикальній - відповідні значення коефіцієнтів (f_n).

Обернене перетворення Лагерра та порівняння з вихідною функцією (f(t)): Використовуються значення прямого перетворення Лагерра для обчислення оберненого перетворення Лагерра через функцію reversed_lagger_transformation. Далі побудований графік цього оберненого перетворення порівнюється з вихідною функцією (f(t)), щоб визначити, наскільки точно обернене перетворення відтворює вихідну функцію.

Оформлення графіків: Задаються підписи осей та заголовки для обох графіків для зручності розуміння

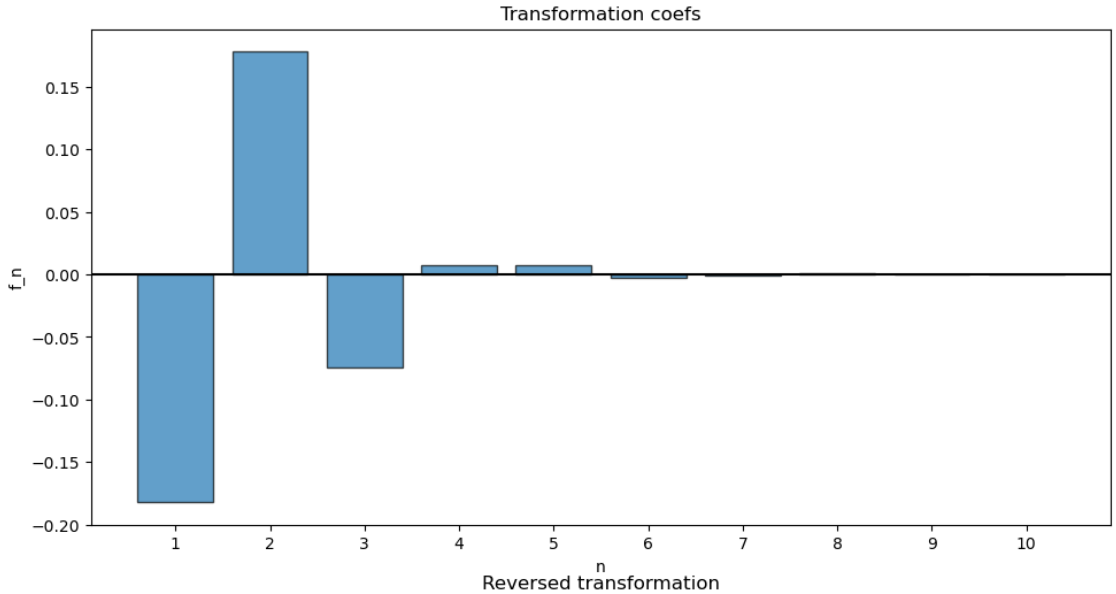
представленої інформації.

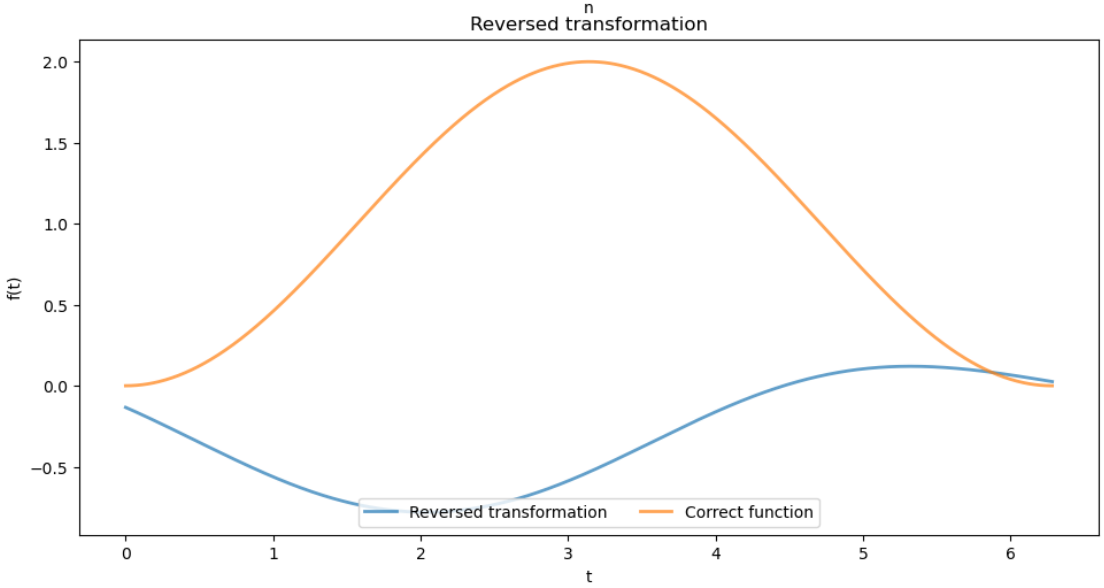
функція дозволяє одночасно порівняти коефіцієнти прямого перетворення Лагерра та відтворення вихідної функції з її оберненим перетворенням.

[23]:
```
def plot_tranformations(f, n, beta=2, gamma=4, t1=0, t2=2*np.pi):

    transform_values = tabulate_transformation(f, n, beta, gamma)
    reversed_transform_values = [reversed_lagger_transformation(transform_values, t,␣
↪beta, gamma) for t in np.linspace(t1, t2, 1000)]
    correct_values = [f(t) for t in np.linspace(t1, t2, 1000)]

    fig = plt.figure(figsize=(10, 10))
    ax = fig.subplots(2, 1)
    ax[0].bar(range(1, n+1), transform_values, alpha=0.7, edgecolor='black')

    ax[0].set_xlabel("n")
    ax[0].set_ylabel("f_n")
    ax[0].set_title("Transformation coefs")
    ax[0].set_xticks(range(1, n+1))
    fig.tight_layout()
    ax[0].axhline(0, color='black')

    ax[1].plot(np.linspace(t1, t2, 1000), reversed_transform_values, alpha=0.7,␣
↪linewidth=2.0, label="Reversed transformation")
    ax[1].plot(np.linspace(t1, t2, 1000), correct_values, alpha=0.7, linewidth=2.0,␣
↪label="Correct function")
    ax[1].set_xlabel("t")
    ax[1].set_ylabel("f(t)")
    ax[1].set_title("Reversed transformation")
    ax[1].legend(loc='lower center', ncol=2)

    plt.show()
```

Виклик функції plot_tranformations візуалізує пряме та обернене перетворення Лагерра для функції (sin(t - π/2) + 1) з використанням 10 коефіцієнтів n.

[24]:
```
plot_tranformations(f, 10)
```



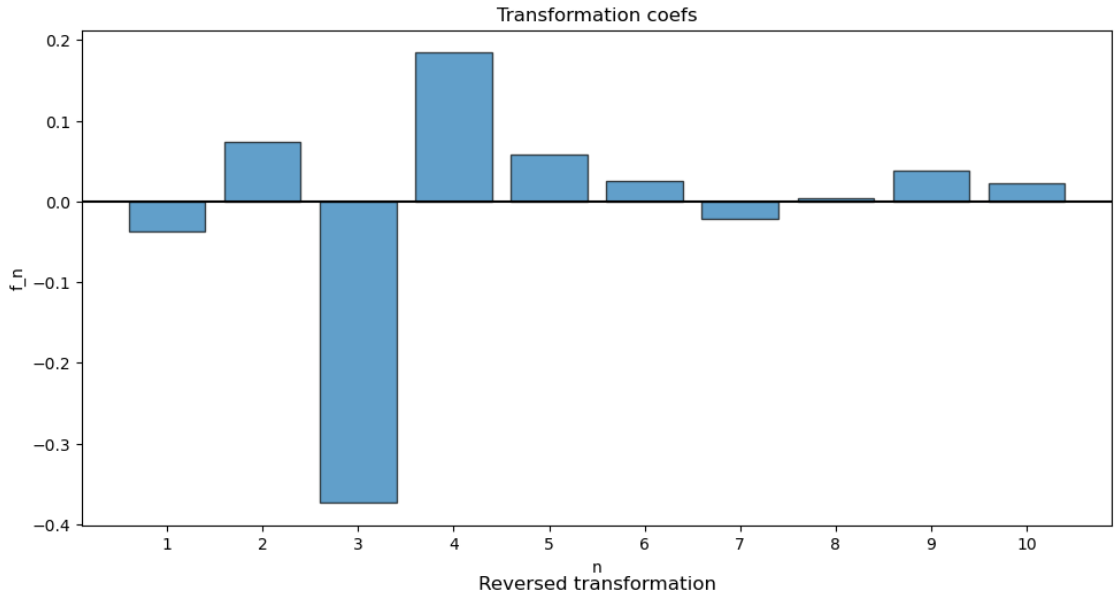


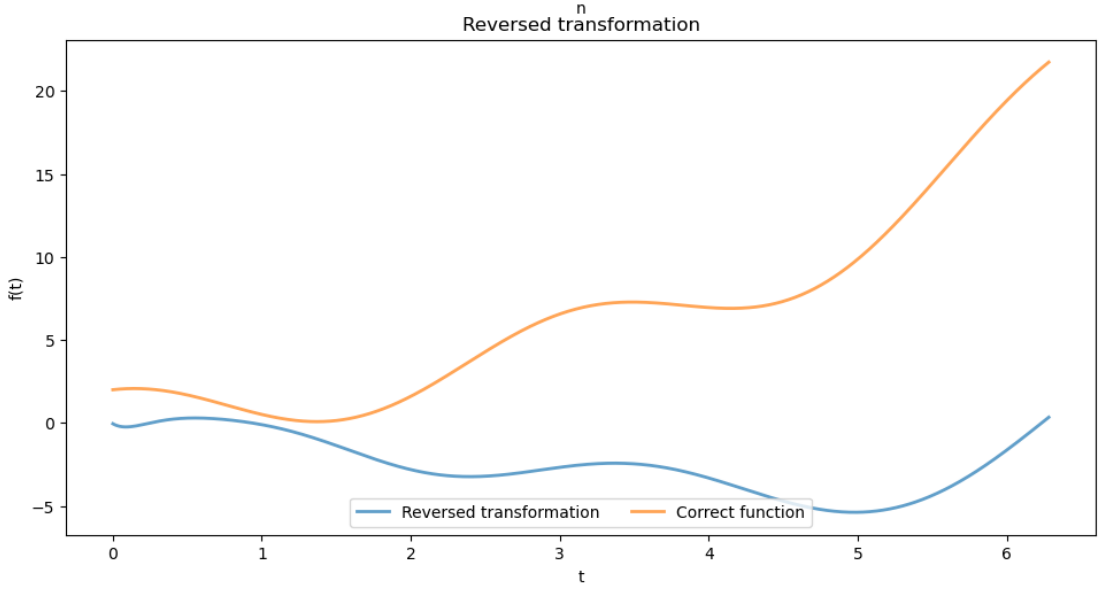
[25]:
```
def my_f(t):
    if t >= 0 and t <= 2*np.pi:
        return np.sin(t) + 2 * np.cos(2*t) + 0.5 * t**2
    else:
        return 0
```

Виклик функції plot_tranformations візуалізує пряме та обернене перетворення Лагерра для моєї функції $(\sin(t)+2\cos(2t)+0.5*t^2)$

з використанням 10 коефіцієнтів n.

[26]:
```
plot_tranformations(my_f, 10)
```

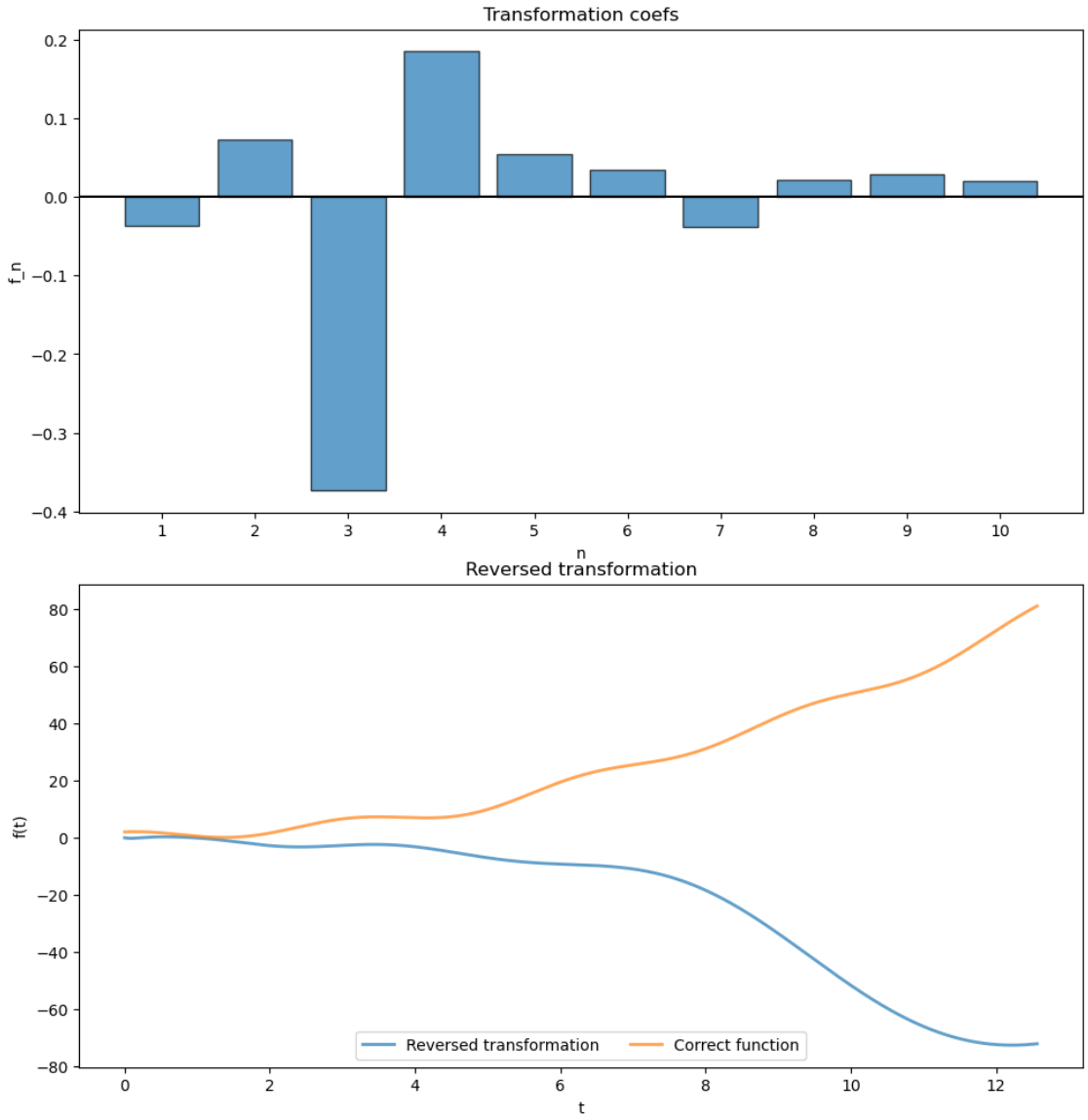

Transformation coefs
f_n
n


Reversed transformation
f(t)
t
Reversed transformation
Correct function

[ ]:

[27]:
```
def my_f(t):
    if t >= 0 and t <= 4*np.pi:
        return np.sin(t) + 2 * np.cos(2*t) + 0.5 * t**2
    else:
        return 0
```

[28]:
```
plot_tranformations(my_f, 10,t1=0,t2=4*np.pi)
```



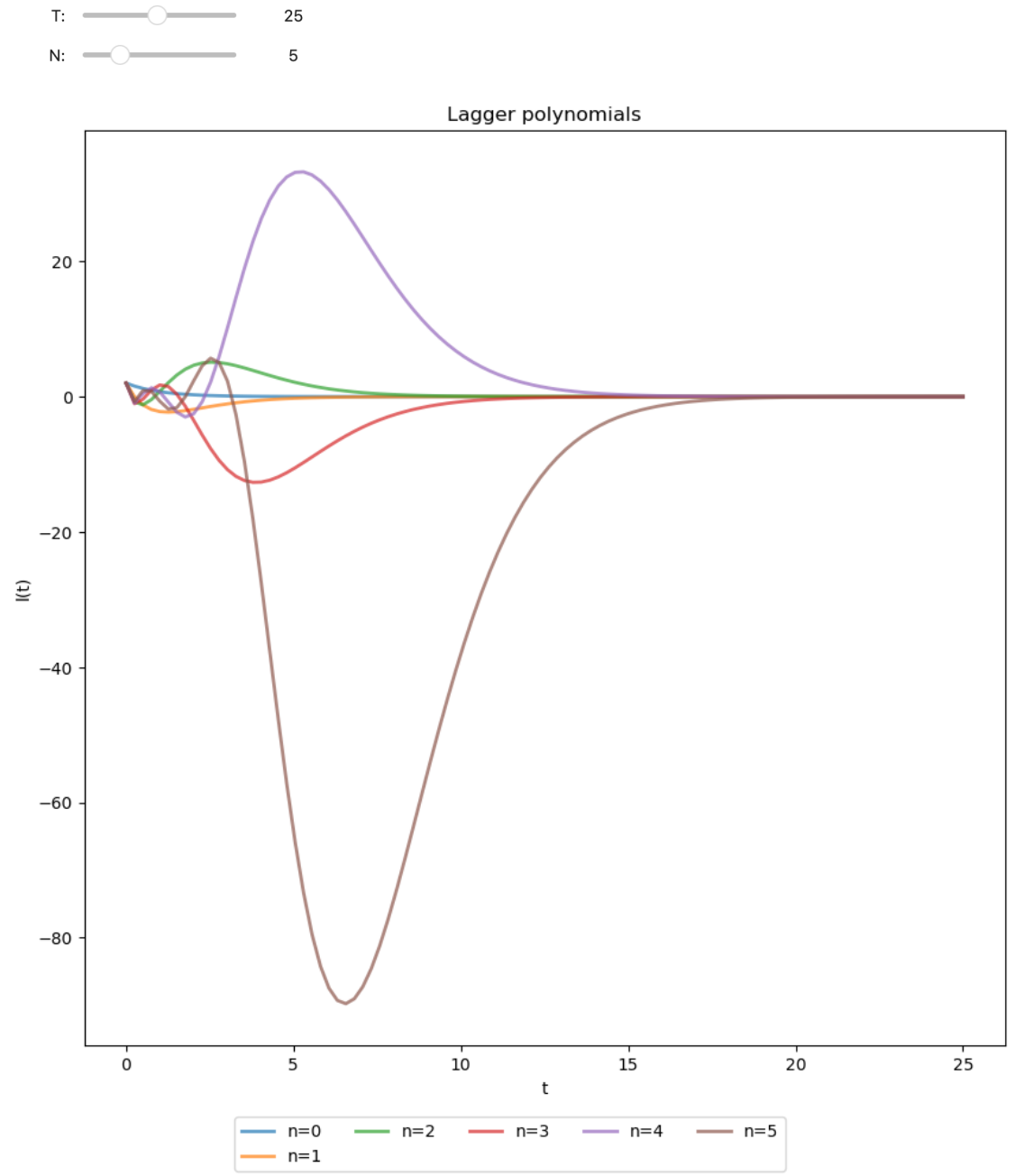
Створення віджетів для завдання №7

Цей код створює інтерактивну панель з двома віджетами: T і N. Віджет T - це повзунок, який дозволяє користувачеві вибирати значення для параметру T в межах від 1 до 50. Віджет N - це також повзунок, але він дозволяє користувачеві вибирати значення для параметру N в межах від 0 до 20.

Функція update_plot викликається при зміні значень віджетів. Вона приймає два аргументи: T і N. У цьому прикладі вона викликає функцію plot_lagger зі значеннями T та N. Ці значення будуть використані для побудови графіка відповідно до логіки, яка реалізована всередині функції plot_lagger.

Завдяки віджетам та функції update_plot, користувач може вибирати значення T та N за допомогою ползунків, і графік буде автоматично оновлюватися в залежності від обраних значень.

[29]:
```
import ipywidgets as widgets
from IPython.display import display

def update_plot(T, N):
    plot_lagger(T, N)

T_widget = widgets.IntSlider(value=10, min=1, max=50, step=1, description='T:')
N_widget = widgets.IntSlider(value=5, min=0, max=20, step=1, description='N:')

interactive_plot = widgets.interactive(update_plot, T=T_widget, N=N_widget)

display(interactive_plot)
```


T:
25
N:
5
Lagger polynomials
20
0
−20
−40
−60
−80
l(t)
0
5
10
15
20
25
t
n=0
n=1
n=2
n=3
n=4
n=5

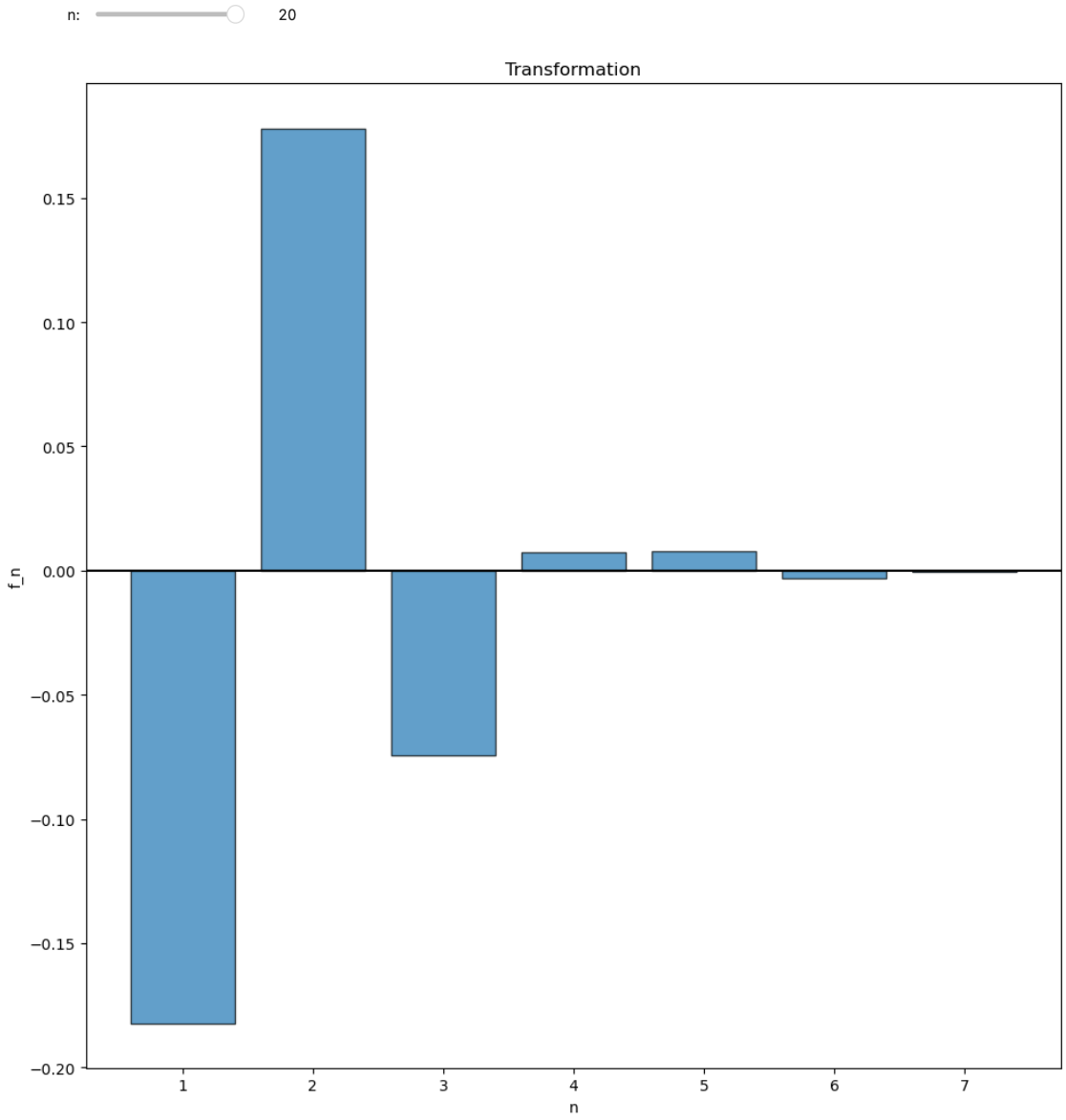
Створення віджетів до завдання №8

Цей код створює інтерактивний віджет у вигляді повзунка (IntSlider), який дозволяє користувачеві вибирати значення для параметра n. Віджет має значення за замовчуванням 5 і може приймати значення від 1 до 20 з кроком 1.

Функція update_plot викликається при зміні значень віджету. У цьому прикладі, коли користувач змінює значення віджету n, викликається функція plot_transformation з новим значенням n. Це означає, що графік буде оновлюватися автоматично при зміні значення n за допомогою ползунка.

Коли ви виконуєте display(interactive_plot), відображається сам віджет (IntSlider) та він стає доступним для вибору значень користувачем.

[30]:
```
def update_plot(n_value):
    plot_transformation(f, n_value)

n_widget = widgets.IntSlider(value=5, min=1, max=20, step=1, description='n:')
interactive_plot = widgets.interactive(update_plot, n_value=n_widget)
display(interactive_plot)
```



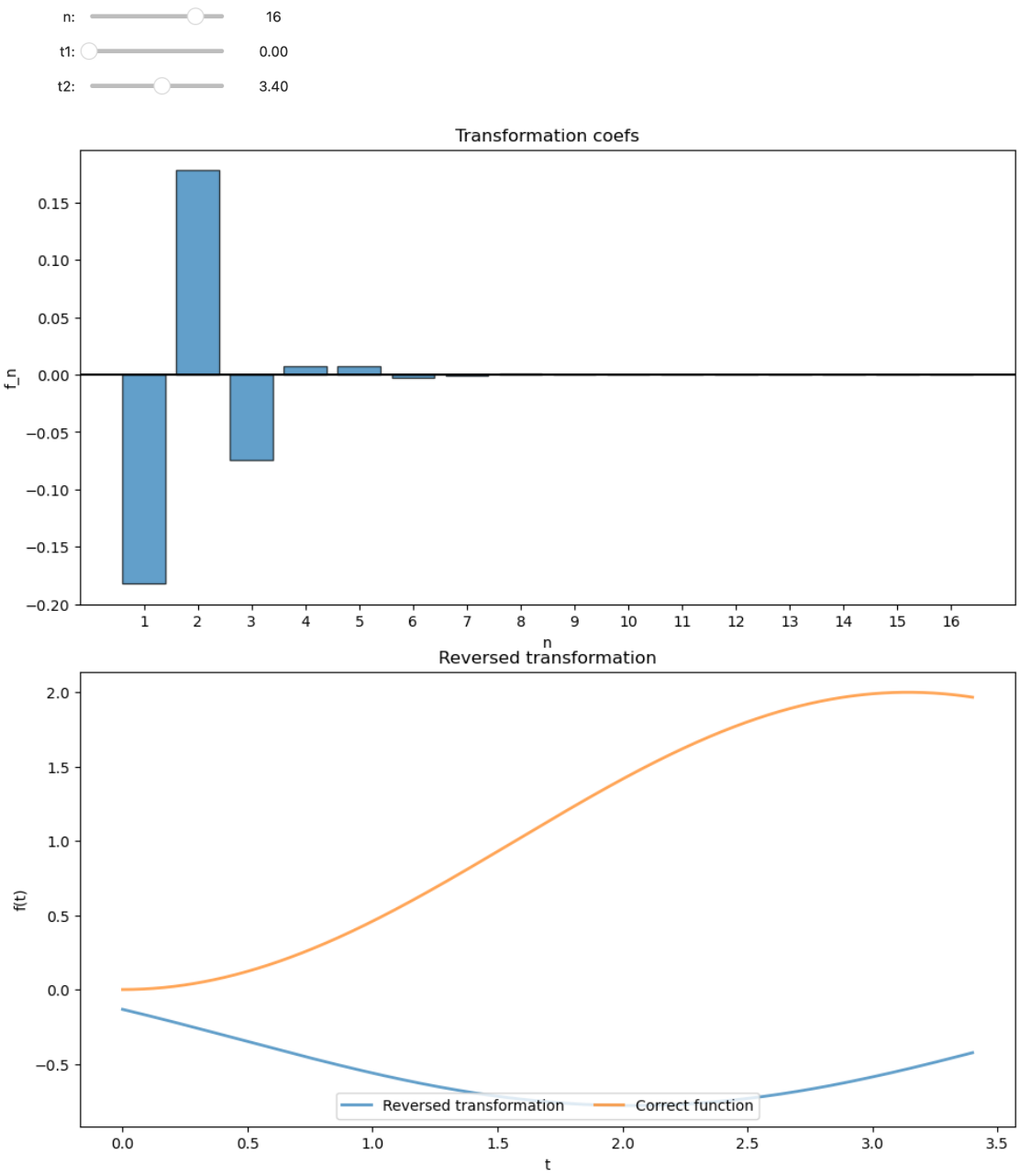
Цей код створює інтерактивну панель з трьома віджетами: n, t1 та t2.

n_widget - повзунок, який дозволяє користувачеві вибирати значення для параметру n в межах від 1 до 20 з кроком 1. t1_widget - також повзунок, але для значення параметру t1. Він дозволяє користувачеві вибирати значення від 0 до 2 * π. t2_widget - ще один повлзунок для параметра t2, також від 0 до 2 * π. Функція update_plot викликається при зміні значень цих віджетів. У даному випадку, коли користувач змінює будь-яке значення віджету (n, t1 або t2), викликається функція plot_tranformations з новими значеннями параметрів. Це дає можливість користувачеві вибирати значення для цих параметрів і бачити графічне представлення результатів.

Коли ви виконуєте display(interactive_plot), відображаються всі три віджети (IntSlider для n та FloatSlider для t1 та t2) і панель стає доступною для вибору значень користувачем.

[31]:
```
def update_plot(n, t1, t2):
    plot_tranformations(f, n, t1=t1, t2=t2)
n_widget = widgets.IntSlider(value=5, min=1, max=20, step=1, description='n:')
```

```
t1_widget = widgets.FloatSlider(value=0, min=0, max=2*np.pi, step=0.1, description='t1:
 ↪')
t2_widget = widgets.FloatSlider(value=2*np.pi, min=0, max=2*np.pi, step=0.1,␣
 ↪description='t2:')

interactive_plot = widgets.interactive(update_plot, n=n_widget, t1=t1_widget,␣
 ↪t2=t2_widget)
display(interactive_plot)
```


n: 16
t1: 0.00
t2: 3.40
Transformation coefs
f_n
n
Reversed transformation
f(t)
t
Reversed transformation
Correct function

Цей код створює інтерактивну панель з трьома віджетами: n, t1 та t2.

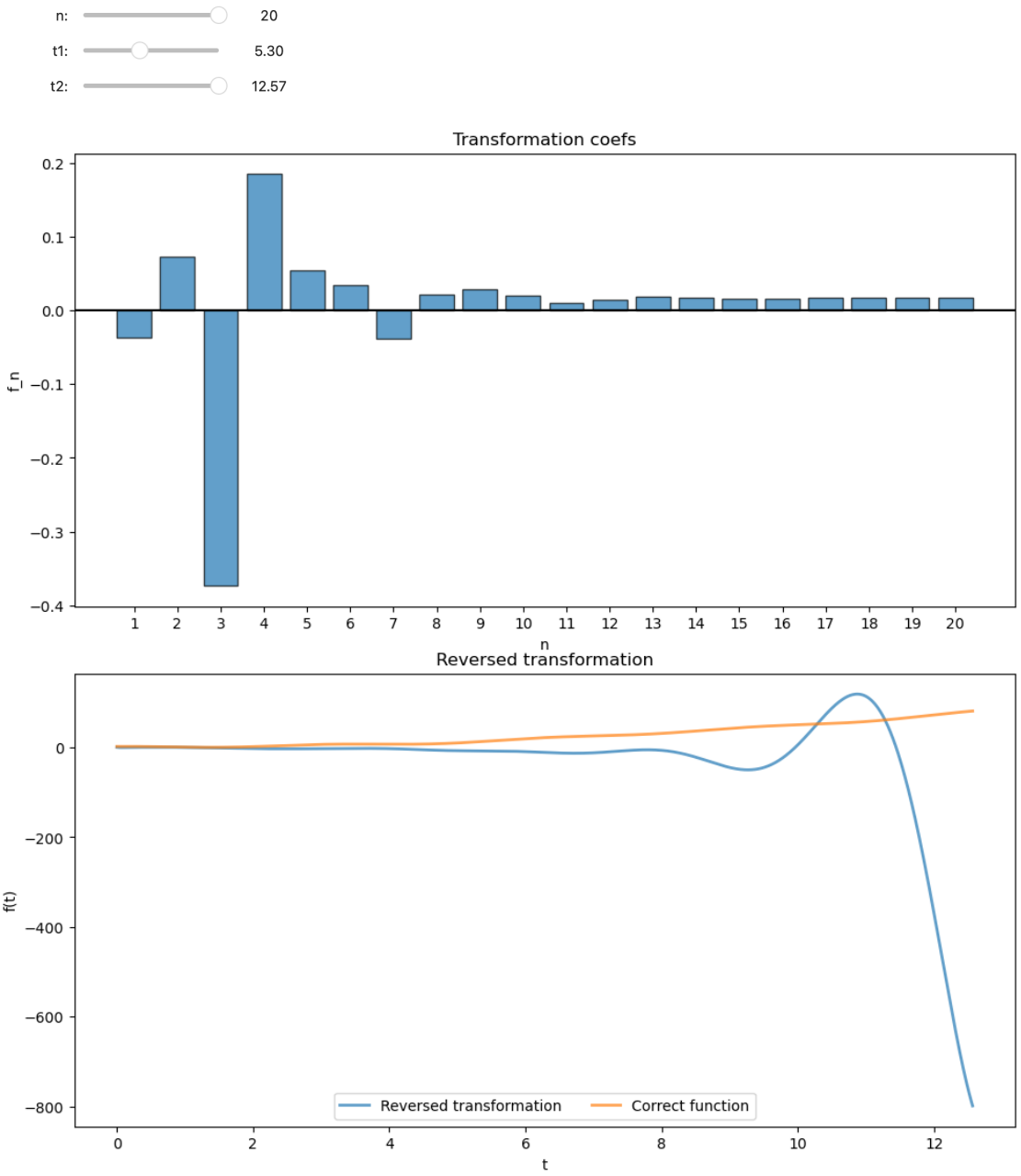
Функція my_function містить внутрішню функцію plot_tranformations_wrapper, яка обгортає функцію plot_tranformations з фіксованими значеннями t1=0 та t2=4*np.pi для параметрів t1 та t2.

Віджет n_widget є повзунком, який дозволяє користувачеві вибирати значення для параметра n в межах від 1 до 20 з кроком 1. t1_widget та t2_widget - це повзунки для параметрів t1 та t2 відповідно. Вони дозволяють користувачу вибирати значення від 0 до 4 * π.

Після створення віджетів і функції-обгортки, створюється інтерактивна панель interactive_plot, яка викликає функцію plot_tranformations_wrapper при зміні значень віджетів n, t1 та t2.

Коли ви виконуєте my_function(), відображаються всі три віджети (IntSlider для n та FloatSlider для t1 та t2) і панель стає доступною для вибору значень користувачем.

[33]:
```
def my_function():
    def plot_tranformations_wrapper(n, t1, t2):
        plot_tranformations(my_f, n, t1=0, t2=4*np.pi)
    n_widget = widgets.IntSlider(value=5, min=1, max=20, step=1, description='n:')
    t1_widget = widgets.FloatSlider(value=0, min=0, max=4*np.pi, step=0.1,␣
↪description='t1:')
    t2_widget = widgets.FloatSlider(value=4*np.pi, min=0, max=4*np.pi, step=0.1,␣
↪description='t2:')

    interactive_plot = widgets.interactive(plot_tranformations_wrapper, n=n_widget,␣
↪t1=t1_widget, t2=t2_widget)

    display(interactive_plot)

my_function()
```


n: 20
t1: 5.30
t2: 12.57
Transformation coefs
f_n
n
Reversed transformation
f(t)
t
Reversed transformation
Correct function

ВИСНОВОК

Під час виконання цих завдань я навчилася:

- Створювати функції в Python: Розроблення функцій, які виконують різноманітні обчислення та використовуються для моделювання різних математичних виразів та функцій.
- Використовувати бібліотеки numpy та matplotlib: Використання numpy для математичних обчислень та matplotlib для візуалізації результатів. Ці бібліотеки стають корисними інструментами для роботи з числовими даними та створення графіків
- Реалізувати чисельне інтегрування та перетворення Лагерра: Розуміння чисельних методів обчислення інтегралів та використання перетворень Лагерра для апроксимації та оберненого відтворення функцій.
- Реалізувати чисельне інтегрування та перетворення Лагерра: Розуміння чисельних методів обчислення інтегралів та використання перетворень Лагерра для апроксимації та оберненого відтворення функцій.
- Створювати візуалізації результатів: Відображення графіків функцій та їхніх перетворень, що допомагає краще розуміти їхню поведінку та властивості.
- Розуміння математичних концепцій через код: Використання програмування для дослідження та візуалізації математичних концепцій, таких як перетворення Лагерра та інтегрування функцій.
- Створювати проекти в Overleaf: Створення проекту в Overleaf дає можливість вивчити LaTeX, мову для створення наукових документів, спільно працювати над проектами з колегами та зосередитися на написанні наукових матеріалів в зручному середовищі для досліджень.
- Створювати віджети:

  Використання бібліотеки ipywidgets у Jupyter Notebook: вивчила, як створювати інтерактивні віджети, такі як повзунки (IntSlider, FloatSlider), які дають можливість користувачеві змінювати параметри та спостерігати за змінами в реальному часі.

  Створення інтерактивних графіків: опанувала процес створення графіків, які реагують на зміни параметрів. Це дозволяє аналізувати різні аспекти даних, вносячи зміни в параметри та спостерігаючи, як це впливає на графік.

  Створення функцій для обробки та візуалізації даних: розроблила функції для обчислення значень та побудови графіків на основі цих значень. Це важливий аспект аналізу даних та візуалізації.

  Розуміння взаємодії між віджетами та функціями: опанувала механізм взаємодії між віджетами та функціями: як віджети можуть викликати функції та оновлювати відображення на основі нових параметрів.

https://github.com/mashatkk/lagger