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

Лабораторна робота
Функції Лаґерра
з курсу "Виробнича практика"

Виконав:
студент групи ПМІ-21
Дудинець Олександр Іванович

Львів – 2023

Мета роботи: здобути практичні навички використання многочленів Лаґерра, їхніх прямих та обернених перетворень.

GitHub репозиторій: https://github.com/dudynets/Laguerre-Polymonials

Перетворення Лагера

[1]:
```
import numpy as np
import pandas as pd
%matplotlib widget
import matplotlib.pyplot as plt
import ipywidgets as widgets
from typing import Callable
import unittest
```

Визначення власної функції для перетворення Лагера та оберненого перетворення Лагера.

[2]:
```
def f_1(t):
    return np.cos(t + np.exp(t) / 2)
```

Реалізація класу, що обраховує значення інтегралу від функції на проміжку [a, b]

[3]:
```
class IntegralSolver:
    """
    Клас для обчислення наближеного значення інтегралу методом прямокутників
    """

    def __init__(self, f: Callable[[float], float]):
        """
        Конструктор класу IntegralSolver

        :param f: Функція, яку потрібно інтегрувати
        """

        self._f = f


    @property
    def f(self) -> Callable[[float], float]:
        return self._f


    def solve(self, a: float, b: float, int_points: int = 10000) -> float:
```

```
        """
        Метод для обчислення наближеного значення інтегралу методом прямокутників

        :param a:           Початок інтервалу
        :param b:           Кінець інтервалу
        :param int_points:  Кількість точок для інтегрування

        :return: Значення інтегралу
        """

        x = np.linspace(a, b, int_points)
        s = sum([self.f(i) for i in x])
        result = s * abs(b - a) / int_points

        return result
```

[4]:
```
class TestIntegralSolver(unittest.TestCase):
    def setUp(self) -> None:
        self.solver = IntegralSolver(lambda x: x**2)


    def test_solve(self):
        # Test solve method
        result = self.solver.solve(0, 1, 10000)
        self.assertAlmostEqual(result, 1/3, places=4)


    def tearDown(self) -> None:
        del self.solver
```

Реалізація класу калькулятора многочленів Лагера

[5]:
```
class LaguerreSolver:
    """
    Клас для роботи з многочленами Лаґерра
    """

    def __init__(self, beta: float = 2.0, sigma: float = 4.0):
        if beta < 0:
            raise ValueError('Value "beta" must be positive')

        if sigma < beta:
            raise ValueError('Value "sigma" must be greater than beta')

        self._beta = beta
        self._sigma = sigma
```

```
    @property
    def beta(self) -> float:
        return self._beta


    @property
    def sigma(self) -> float:
        return self._sigma


    def solve_polynomial(
        self,
        t: float,
        n: int,
    ) -> float:
        """
        Метод для обчислення многочлену Лаґерра

        :param t:       Значення аргументу
        :param n:       Степінь многочлена Лаґерра

        :return:        Значення многочлена Лаґерра
        """

        # Валідація вхідних даних
        if n < 0:
            raise ValueError('Value "n" must be positive')

        # Найкращі випадки
        l_prev_prev = np.sqrt(self.sigma) * np.exp(-self.beta * t / 2)
        l_prev = np.sqrt(self.sigma) * (1 - self.sigma * t) * np.exp(-self.beta
↪* t / 2)
        if n == 0:
            return l_prev_prev
        if n == 1:
            return l_prev

        # Обчислення
        for i in range(2, n + 1):
            temp = l_prev
            l_prev = (2 * i - 1 - self.sigma * t) * l_prev / i - (i - 1) *
↪l_prev_prev / i
            l_prev_prev = temp

        return l_prev
```

```python
def tabulate_polynomial(
        self,
        n: int,
        t_max: float,
        t_step: float = 0.1,
) -> pd.DataFrame:
    """
    Функція для табуляції многочленів Лаґерра

    :param n:       Степінь многочлена Лаґерра
    :param t_max:   Максимальне значення аргументу
    :param t_step:  Крок аргументу

    :return:        DataFrame з табульованими значеннями
    """

    # Валідація вхідних даних
    if n < 0:
        raise ValueError('Value "n" must be positive')

    if t_max < 0:
        raise ValueError('Value "t_max" must be positive')

    if t_step < 0:
        raise ValueError('Value "t_step" must be positive')

    # Табуляція
    t = np.arange(0, t_max, t_step)
    return pd.DataFrame(
        data={
            't': t,
            f'L_{n}': [self.solve_polynomial(t=i, n=n) for i in t]
        }
    ).set_index('t')


def find_optimal_t(
        self,
        n_max: int = 20,
        epsilon: float = 1e-3,
        t_max: float = 100,
        t_points: int = 1000,
) -> tuple[float, pd.DataFrame]:
    """
```

```python
        Функція для проведення обчислювального експерименту. Пошук такого t, що␣
↪|laguerre_polymonials(n, t)| < epsilon для усіх n Є [0, N]

        :param n_max:       Верхня межа степеня многочлена Лаґерра
        :param epsilon:     Точність
        :param t_max:       Максимальне значення аргументу
        :param t_points:    Кількість точок для від 0 до t_max

        :return:            Кортеж з t та DataFrame з табульованими значеннями
        """

        # Валідація вхідних даних
        if n_max < 0:
            raise ValueError('Value "N" must be positive')

        if epsilon < 0:
            raise ValueError('Value "epsilon" must be positive')

        if t_max < 0:
            raise ValueError('Value "t_max" must be positive')

        if t_points < 0:
            raise ValueError('Value "t_points" must be positive')

        # Пошук t
        T = np.linspace(0, t_max, t_points)
        N = range(0, n_max + 1)
        suitable_t = None
        for t in T:
            is_t_suitable = True
            for n in N:
                if abs(self.solve_polynomial(t=t, n=n)) > epsilon:
                    is_t_suitable = False
                    break
            if is_t_suitable and suitable_t is None:
                suitable_t = t
                break

        # Табуляція
        return suitable_t, pd.DataFrame(
            data={
                'n': N,
                'L_n': [self.solve_polynomial(t=suitable_t, n=n) for n in N]
            }
        ).set_index('n')
```

```
    def solve_laguerre_transform(
            self,
            f: Callable[[float], float],
            n_max: int,
            int_points: int = 10000
    ) -> float:
        """
        Функція для обчислення перетворення Лаґерра

        :param f:           Функція, яку перетворюємо
        :param n_max:       Верхня межа степеня многочлена Лаґерра (N)
        :param int_points:  Кількість точок для інтегрування

        :return:            Значення перетворення Лаґерра
        """

        if n_max < 0:
            raise ValueError('Value "n_max" must be positive')

        if int_points < 0:
            raise ValueError('Value "int_points" must be positive')

        # Функція для інтегрування
        def integrand(t):
            alpha = self.sigma - self.beta
            return f(t) * self.solve_polynomial(t=t, n=n_max) * np.exp(-alpha *␣
↪t)

        # Верхня межа інтегрування
        t_max, _ = self.find_optimal_t(n_max=n_max)

        integral_solver = IntegralSolver(integrand)
        result = integral_solver.solve(0, t_max, int_points)
        return result


    def tabulate_laguerre_transform(
            self,
            f: Callable[[float], float],
            n_max: int,
            int_points: int = 10000,
    ) -> pd.DataFrame:
        """
        Функція для табулювання перетворення Лаґерра

        :param f:           Функція, яку перетворюємо
        :param n_max:       Верхня межа степеня многочлена Лаґерра (N)
```

```
        :param int_points:  Кількість точок для інтегрування

        :return:            DataFrame з табульованими значеннями
        """

        # Валідація вхідних даних
        if n_max < 0:
            raise ValueError('Value "n_max" must be positive')

        if int_points < 0:
            raise ValueError('Value "t_step" must be positive')

        # Табулювання
        N = range(0, n_max)
        return pd.DataFrame(
            data={
                'n': N,
                f'L_n': [self.solve_laguerre_transform(f=f, n_max=n,␣
↪int_points=int_points) for n in N]
            }
        ).set_index('n')


    def solve_inverse_laguerre_transform(
            self,
            h: list[float],
            t: float,
    ) -> float:
        """
        Функція для обчислення оберненого перетворення Лаґерра

        :param h:       Довільна послідовність дійсних чисел
        :param t:       Значення аргументу

        :return:        Значення оберненого перетворення Лаґерра
        """

        # Обчислення
        return sum([h[k] * self.solve_polynomial(t=t, n=k) for k in range(0,␣
↪len(h))])
```

[6]:
```
class TestLaguerreSolver(unittest.TestCase):
    def setUp(self) -> None:
        self.solver = LaguerreSolver()


    def test_constructor(self):
```

```
        # Test beta validator
        with self.assertRaises(ValueError):
            LaguerreSolver(beta=-1)

        # Test sigma validator
        with self.assertRaises(ValueError):
            LaguerreSolver(sigma=1, beta=2)


    def test_solve_polynomial(self):
        # Test solve_polynomial method
        result = self.solver.solve_polynomial(2, 2)
        self.assertAlmostEqual(result, 4.601399630044832, places=4)

        # Test n < 0
        with self.assertRaises(ValueError):
            self.solver.solve_polynomial(2, -1)


    def test_tabulate_polynomial(self):
        # Test tabulate_polynomial method
        result = self.solver.tabulate_polynomial(2, 10, 1)
        self.assertEqual(result.shape, (10, 1))
        self.assertAlmostEqual(result.iloc[0, 0], 2.0, places=4)
        self.assertAlmostEqual(result.iloc[9, 0], 0.1424149139160282, places=4)

        # Test n < 0
        with self.assertRaises(ValueError):
            self.solver.tabulate_polynomial(-1, 10, 1)

        # Test t_max < 0
        with self.assertRaises(ValueError):
            self.solver.tabulate_polynomial(2, -1, 1)

        # Test t_step < 0
        with self.assertRaises(ValueError):
            self.solver.tabulate_polynomial(2, 10, -1)


    def test_find_optimal_t(self):
        # Test find_optimal_t method
        result, df = self.solver.find_optimal_t(20, 1e-3, 100, 1000)
        self.assertAlmostEqual(result, 79.07907907907908, places=4)
        self.assertEqual(df.shape, (21, 1))
        self.assertAlmostEqual(df.iloc[0, 0], 9.066137838279844e-35, places=4)
        self.assertAlmostEqual(df.iloc[20, 0], 0.0009699020960047245, places=4)
```

```python
        # Test n_max < 0
        with self.assertRaises(ValueError):
            self.solver.find_optimal_t(-1, 1e-3, 100, 1000)

        # Test epsilon < 0
        with self.assertRaises(ValueError):
            self.solver.find_optimal_t(20, -1e-3, 100, 1000)

        # Test t_max < 0
        with self.assertRaises(ValueError):
            self.solver.find_optimal_t(20, 1e-3, -100, 1000)

        # Test t_points < 0
        with self.assertRaises(ValueError):
            self.solver.find_optimal_t(20, 1e-3, 100, -1000)


    def test_solve_laguerre_transform(self):
        # Test solve_laguerre_transform method
        result = self.solver.solve_laguerre_transform(lambda x: x**2, 2)
        self.assertAlmostEqual(result, 0.5431555555545893, places=4)

        # Test n_max < 0
        with self.assertRaises(ValueError):
            self.solver.solve_laguerre_transform(lambda x: x**2, -1)

        # Test int_points < 0
        with self.assertRaises(ValueError):
            self.solver.solve_laguerre_transform(lambda x: x**2, 2, -10000)


    def test_tabulate_laguerre_transform(self):
        # Test tabulate_laguerre_transform method
        result = self.solver.tabulate_laguerre_transform(lambda x: x**2, 2, 10000)
        self.assertEqual(result.shape, (2, 1))
        self.assertAlmostEqual(result.iloc[0, 0], 0.14813332817992056, places=4)
        self.assertAlmostEqual(result.iloc[1, 0], -0.44439999999445956, places=4)

        # Test n_max < 0
        with self.assertRaises(ValueError):
            self.solver.tabulate_laguerre_transform(lambda x: x**2, -1, 10000)

        # Test int_points < 0
        with self.assertRaises(ValueError):
            self.solver.tabulate_laguerre_transform(lambda x: x**2, 2, -10000)
```

```
    def test_solve_inverse_laguerre_transform(self):
        # Test solve_inverse_laguerre_transform method
        result = self.solver.solve_inverse_laguerre_transform([1, 2, 3], 2)
        self.assertAlmostEqual(result, 10.285481525982567, places=4)


    def tearDown(self) -> None:
        del self.solver
```

Реалізація класу, що створює графіки до многочленів Лагера

[7]:
```
class LaguerrePlotter(LaguerreSolver):
    """
    Клас для побудови графіків многочленів Лаґерра та їх перетворень
    """

    def plot_laguerre_polynomials(
            self,
            t_max: float,
            n_max: int,
            t_step: float = 0.01,
    ) -> None:
        """
        Функція для побудови графіку многочленів Лаґерра

        :param t_max:   Максимальне значення аргументу
        :param n_max:   Верхня межа степеня многочлена Лаґерра
        :param t_step:  Крок аргументу

        :return:        None
        """

        plt.figure(figsize=(12, 8))

        for n in range(0, n_max + 1):
            l_n_tabulation = self.tabulate_polynomial(
                n=n,
                t_max=t_max,
                t_step=t_step
            )
            plt.plot(l_n_tabulation.index, l_n_tabulation[f'L_{n}'],␣
↪label='$L_{' + str(n) + '}$')

        plt.title('Многочлени Лаґерра')
        plt.xlabel('t')
```

```
        plt.ylabel('$L_n(t)$')
        plt.grid()
        plt.legend(
            loc='upper left',
            bbox_to_anchor=(-0.2, 1)
        )

        plt.show()


    def plot_laguerre_transform(
            self,
            f: Callable[[float], float],
            n_max: int,
            t_max: float = np.pi * 2,
            t_step: float = 0.01,
            int_points: int = 10000,
    ) -> None:
        """
        Функція для побудови графіку перетворення Лаґерра

        :param f:           Функція, яку перетворюємо
        :param n_max:       Верхня межа степеня многочлена Лаґерра (N)
        :param t_max:       Максимальне значення аргументу
        :param t_step:      Крок аргументу
        :param int_points:  Кількість точок для інтегрування

        :return:            None
        """

        # Обчислення послідовності h
        laguerre_transform_tabulation_values = self.tabulate_laguerre_transform(
            f=f,
            n_max=n_max,
            int_points=int_points,
        )
        h = laguerre_transform_tabulation_values['L_n'].tolist()

        # Табулювання
        T = np.arange(0, t_max, t_step)
        inverse_laguerre_transform_tabulation = pd.DataFrame(
            data={
                't': T,
                'h': [self.solve_inverse_laguerre_transform(h=h, t=t) for t in T]
            }
        ).set_index('t')
```

```python
    # Побудова графіків
    plt.figure(figsize=(12, 8))
    plt.subplots_adjust(hspace=0.5)
    plt.suptitle(f'Графік $\\widetilde{{f}}^N(t), t\\in[0, {t_max}]$')

    # Перетворення Лаґерра
    plt.subplot(2, 1, 1)
    plt.bar(
        laguerre_transform_tabulation_values.index,
        laguerre_transform_tabulation_values['L_n'],
    )
    plt.title(f'Перетворення Лагерра, N = {n_max}')
    plt.xlabel('n')
    plt.ylabel(r'$L_{N}$')
    plt.grid()

    # Обернене перетворення Лаґерра
    plt.subplot(2, 1, 2)
    initial_function_tabulation = pd.DataFrame(
        data={
            't': T,
            'f': [f(t) for t in T]
        }
    ).set_index('t')
    plt.plot(
        initial_function_tabulation.index,
        initial_function_tabulation['f'],
        label='Початкова функція',
        linewidth=6,
        color='black',
        alpha=0.25
    )

    plt.plot(
        inverse_laguerre_transform_tabulation.index,
        inverse_laguerre_transform_tabulation['h'],
        label='Обернене перетворення Лагерра',
        linewidth=1,
        color='green'
    )

    plt.title('Обернене перетворення Лагерра')
    plt.xlabel('t')
    plt.ylabel(r'$\widetilde{f}^N(t)$')
    plt.legend()
    plt.grid()
```

```
        plt.show()
```

Реалізація класу графічного інтерфейсу користувача

[8]:
```
class LaguerreUI(LaguerrePlotter):
    def interactive_polynomial_solve(self):
        return widgets.interact(
            self.solve_polynomial,
            t=widgets.FloatSlider(
                value=2,
                min=0,
                max=10,
                step=0.1,
                description='t'
            ),
            n=widgets.IntSlider(
                value=2,
                min=0,
                max=20,
                step=1,
                description='n'
            )
        )


    def interactive_tabulate_polynomial(self):
        return widgets.interact(
            self.tabulate_polynomial,
            n=widgets.IntSlider(
                value=2,
                min=0,
                max=20,
                step=1,
                description='n'
            ),
            t_max=widgets.FloatSlider(
                value=5,
                min=0,
                max=10,
                step=0.1,
                description='Макс. t'
            ),
            t_step=widgets.FloatSlider(
                value=0.1,
                min=0.1,
                max=1,
```

```
                step=0.1,
                description='Крок t'
            )
        )


    def interactive_solve_laguerre_transform(self, f: Callable[[float], float]):
        return widgets.interact(
            self.solve_laguerre_transform,
            f=widgets.fixed(f),
            n_max=widgets.IntSlider(
                value=20,
                min=0,
                max=20,
                step=1,
                description='Макс. n'
            ),
            int_points=widgets.IntSlider(
                value=10000,
                min=1,
                max=100000,
                step=1000,
                description='К-сть інтервалів розбиття для інтегрування'
            ),
        )


    def interactive_tabulate_laguerre_transform(self, f: Callable[[float],␣
↪float]):
        return widgets.interact(
            self.tabulate_laguerre_transform,
            f=widgets.fixed(f),
            n_max=widgets.IntSlider(
                value=20,
                min=0,
                max=20,
                step=1,
                description='Макс. n'
            ),
            int_points=widgets.IntSlider(
                value=10000,
                min=1,
                max=100000,
                step=1000,
                description='К-сть інтервалів розбиття для інтегрування'
            ),
        )
```

```python
def interactive_solve_inverse_laguerre_transform(self, h: list[float]):
    return widgets.interact(
        self.solve_inverse_laguerre_transform,
        h=widgets.fixed(h),
        t=widgets.FloatSlider(
            value=2,
            min=0,
            max=10,
            step=0.1,
            description='t'
        ),
    )


def interactive_plot_laguerre_polynomials(self):
    return widgets.interact(
        self.plot_laguerre_polynomials,
        t_max=widgets.FloatSlider(
            value=10,
            min=0,
            max=20,
            step=0.1,
            description='Макс. t'
        ),
        n_max=widgets.IntSlider(
            value=20,
            min=0,
            max=20,
            step=1,
            description='Макс. n'
        ),
        t_step=widgets.FloatSlider(
            value=0.1,
            min=0.1,
            max=1,
            step=0.1,
            description='Крок t'
        )
    )


def interactive_plot_laguerre_transform(self, f: Callable[[float], float]):
    return widgets.interact(
        self.plot_laguerre_transform,
        f=widgets.fixed(f),
```

```
            n_max=widgets.IntSlider(
                value=20,
                min=0,
                max=20,
                step=1,
                description='Макс. n'
            ),
            t_max=widgets.FloatSlider(
                value=10,
                min=0,
                max=20,
                step=0.1,
                description='Макс. t'
            ),
            t_step=widgets.FloatSlider(
                value=0.1,
                min=0.1,
                max=1,
                step=0.1,
                description='Крок t'
            ),
            int_points=widgets.IntSlider(
                value=10000,
                min=1,
                max=100000,
                step=1000,
                description='К-сть інтервалів розбиття для інтегрування'
            ),
        )
```

Unit тестування модулів програми

[9]:
```
unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_solve (__main__.TestIntegralSolver.test_solve) ... ok
test_constructor (__main__.TestLaguerreSolver.test_constructor) ... ok
test_find_optimal_t (__main__.TestLaguerreSolver.test_find_optimal_t) ... ok
test_solve_inverse_laguerre_transform
(__main__.TestLaguerreSolver.test_solve_inverse_laguerre_transform) ... ok
test_solve_laguerre_transform
(__main__.TestLaguerreSolver.test_solve_laguerre_transform) ... ok
test_solve_polynomial (__main__.TestLaguerreSolver.test_solve_polynomial) ... ok
test_tabulate_laguerre_transform
(__main__.TestLaguerreSolver.test_tabulate_laguerre_transform) ... ok
test_tabulate_polynomial (__main__.TestLaguerreSolver.test_tabulate_polynomial)
... ok
```

```
----------------------------------------------------------------------
Ran 8 tests in 0.141s

OK
```

[9]:
```
<unittest.main.TestProgram at 0x12aa14d40>
```

1.5.1 Многочлени Лаґерра

Функція, що обчислює многочлен Лаґерра порядку n, для заданого значення аргументу t та параметрів beta та sigma

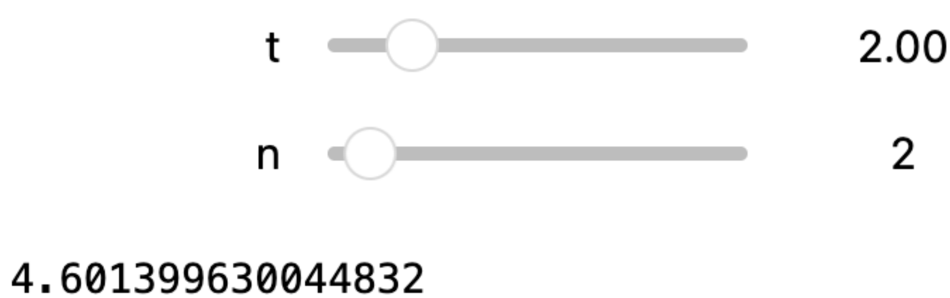
[10]:
```python
# Створення об'єкту класу для обрахунку многочленів Лаґерра
solver = LaguerreSolver(2, 4)

# Створення об'єкту класу для графічного інтерфейсу
ui = LaguerreUI(2, 4)
```

[11]:
```python
ui.interactive_polynomial_solve()
```

```
interactive(children=(FloatSlider(value=2.0, description='t', max=10.0),␣
↪IntSlider(value=2, description='n', m...
```

[11]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```



1.5.2 Табуляція многочленів Лаґерра

Функція, що табулює многочлен Лаґерра порядку n від 0 до t_max

[12]:
```python
ui.interactive_tabulate_polynomial()
```

```
interactive(children=(IntSlider(value=2, description='n', max=20),␣
↪FloatSlider(value=5.0, description='Макс. t...
```

[12]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```

n 2

Макс. t 5.00

Крок t 0.10

	L_2
t	
0.0	2.000000
0.1	0.506709
0.2	-0.458489
0.3	-1.007513
0.4	-1.233389
0.5	-1.213061
0.6	-1.009813
0.7	-0.675356
0.8	-0.251624
0.9	0.227679
1.0	0.735759
1.1	1.251595
1.2	1.758974

1.5.3 Обчислювальний експеримент

Функція, що визначає найменше значення аргументу t, при якому всі поліноми Лаґерра порядку від 0 до n_max — менші за epsilon

[13]:
```
t, df = solver.find_optimal_t()

print(f'Оптимальне значення t: {t}')
```

```
df
```

```
Оптимальне значення t: 79.07907907907908
```

[13]:

```
              L_n
n
0    9.066138e-35
1   -2.858701e-32
2    4.478343e-30
3   -4.647081e-28
4    3.593209e-26
5   -2.208132e-24
6    1.123332e-22
7   -4.865604e-21
8    1.831625e-19
9   -6.087176e-18
10   1.808168e-16
11  -4.848845e-15
12   1.183547e-13
13  -2.647728e-12
14   5.460659e-11
15  -1.043487e-09
16   1.855654e-08
17  -3.082750e-07
18   4.800407e-06
19  -7.027805e-05
20   9.699021e-04
```

1.5.4 Обчислення значень інтегралів

Функція для обчислення прямого перетворення Лаґерра за допомогою апроксимації інтегралу методом прямокутників

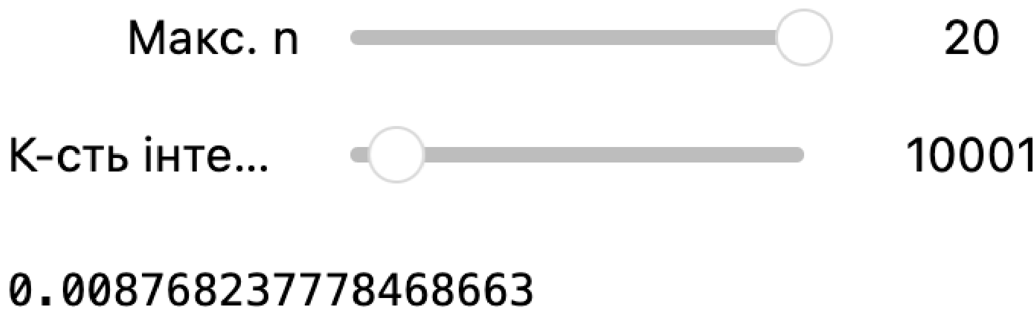
[14]:

```
ui.interactive_solve_laguerre_transform(lambda t: np.exp(-t ** 2))
```

```
interactive(children=(IntSlider(value=20, description='Макс. n', max=20),␣
↪IntSlider(value=10000, description='...
```

[14]:

```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```



1.5.5 Перетворення Лаґерра

Функція для табулювання прямого перетворення Лаґерра порядку від 0 до n_max

[15]:
```
# Табулювання
ui.interactive_tabulate_laguerre_transform(f_1)
```

```
interactive(children=(IntSlider(value=20, description='Макс. n', max=20),␣
↪IntSlider(value=10000, description='...
```

[15]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```

Макс. n 20

К-сть інте... 10001

	L_n
n	
0	0.066660
1	-0.182204
2	0.178056
3	-0.074283
4	0.007263
5	0.007587
6	-0.003096
7	-0.000615
8	0.000799
9	-0.000026
10	-0.000236
11	0.000053

1.5.6 Обернене перетворення Лаґерра

Функція для обчислення оберненого перетворення Лаґерра. Приймає послідовність коефіцієнтів h, які можна отримати з табуляції перетворення Лаґерра

[16]:
```
# Отримуємо послідовність h з табуляції перетворення Лаґерра
h = solver.tabulate_laguerre_transform(
    f=f_1,
    n_max=20,
    int_points=10000
)['L_n'].tolist()

ui.interactive_solve_inverse_laguerre_transform(h)
```

```
interactive(children=(FloatSlider(value=2.0, description='t', max=10.0),␣
↪Output()), _dom_classes=('widget-inte...
```

[16]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```


```
1.4159688430387627
```

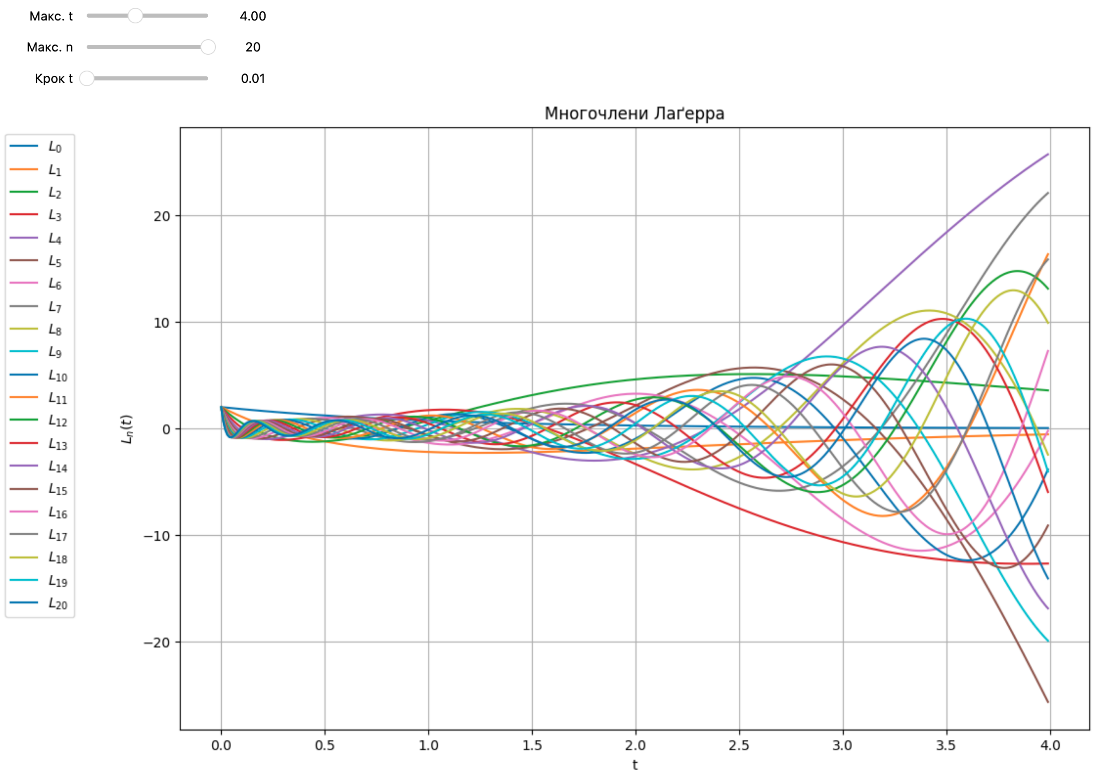
1.5.7 Графік функції Лаґерра

Функція для побудови графіку многочленів Лаґерра порядку від 0 до n_max

[17]:
```
ui.interactive_plot_laguerre_polynomials()
```

```
interactive(children=(FloatSlider(value=10.0, description='Макс. t', max=20.0),␣
↪IntSlider(value=20, descriptio...
```

[17]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```



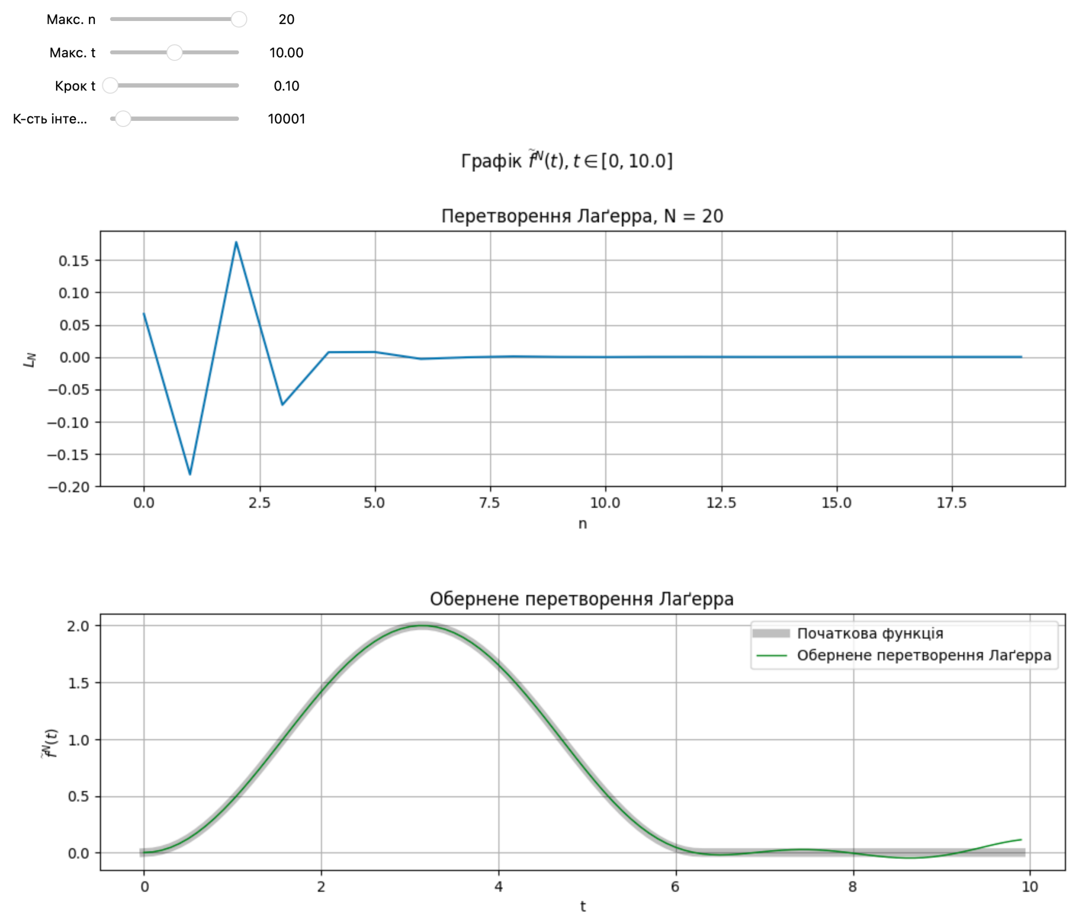
1.5.8 Графік $\widetilde{f_1}^N(t)$

```
def f_1(t):
    return np.cos(t + np.exp(t) / 2)
```

[18]:
```
ui.interactive_plot_laguerre_transform(f_1)
```

```
interactive(children=(IntSlider(value=20, description='Макс. n', max=20),␣
↪FloatSlider(value=10.0, description=...
```

[18]:
```
<function ipywidgets.widgets.interaction._InteractFactory.__call__.<locals>.<lam
bda>(*args, **kwargs)>
```



Висновок

У лабораторній роботі було розглянуто використання многочленів Лаґерра у контексті обчислення їхнього прямого та оберненого перетворення. Були реалізовані функції для обчислення многочленів, їхньої табуляції, проведення обчислювального експерименту для знаходження оптимального значення аргументу t та обчислення перетворень.

В результаті роботи було показано, як можна використовувати многочлени Лаґерра для перетворення функцій та як здійснювати обернене перетворення для відновлення початкових функцій. Проведено аналіз та вивчення залежностей між параметрами многочленів Лаґерра та їхніми властивостями.

Окремий акцент був зроблений на побудові графіків многочленів Лаґерра для різних значень степенів. Це дозволило візуально спостерігати їхні властивості та залежність від параметрів.

Також було проведено функції f1. Для неї було побудовано графік перетворення Лаґерра та оберненого перетворення, що дозволило візуально спостерігати схожість графіку оберненого перетворення та початкової функції.

Отже, лабораторна робота надала можливість здобути практичні навички використання многочленів Лаґерра, використовуючи об'єктно-орієнтований підхід розробки програмного забезпечення.