Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Лабораторна робота #2

Інтерполяція функцій сплайнами: кубічні інтерполяційні сплайни

Виконав:

Захаров Дмитро Олегович

Група МП-31

Харків – 2023

Зміст

1 Постановка задачі

Побудувати інтерполяційні кубічні сплайни $S_n(x)$ для функції $f : [\alpha, \beta] \to \mathbb{R}$, заданої в узлах:

1. $x_k = \alpha + k \cdot h,\ h = \frac{\beta - \alpha}{n},\ k \in \{0, \ldots, n\}$
2. $\hat{x}_k = \frac{1}{2}\left(\beta + \alpha - (\beta - \alpha)\cos\frac{2k+1}{2(n+1)}\pi\right),\ k \in \{0, \ldots, n\}$

значеннями $\{f_i\}_{i=0}^n$.

На друк вивести результати у вигляді таблиць:

$$x_i^* \qquad f(x_i^*) \qquad S_n(x_i^*) \qquad |f(x_i^*) - S_n(x_i^*)|$$

де $x_i^* = x_i + \alpha h,\ i \in \{0, \ldots, n-1\}, \alpha \in (0, 1)$.

Варіант 5. $\alpha = 0, \beta = 1$,

$$f(x) = x^3 + \cos x + e^{\frac{x}{10}} \sin x$$

2 Опис методу

2.1 Кубічний інтерполяційний сплайн

Кубічним інтерполяційним сплайном $S(x)$ для функції $f \in \mathcal{C}[\alpha, \beta]$ для вузлів $\{x_i\}_{i=0}^n \subset [\alpha, \beta]$ називається функція, що має наступні властивості:

1. $S(x)$ є кубічним многочленом на відрізку між суміжними вузлами. Формально:
$$S(x) = a_0^{[i]}x^3 + a_1^{[i]}x^2 + a_2^{[i]}x + a_3^{[i]},\ x \in [x_{i-1}, x],\ i \in \{1, \dots, n\}$$
2. $S \in \mathcal{C}^2[\alpha, \beta]$
3. $S(x_i) = f(x_i)\, \forall i \in \{0, \dots, n\}$
4. $S''(\alpha) = S''(\beta) = 0$ (кубічний сплайн дефекту 1)

2.2 Теорема про знаходження кубічного інтерполяційного сплайну

Теорема. Умови, що зазначені вище, однозначно визначають кубічний інтерполяційний сплайн $S(x)$, дефекту 1, що має вигляд:

$$S(x) = \frac{m_{i-1}(x_i - x)^3}{6h_i} + \frac{m_i(x - x_{i-1})^3}{6h_i} + \left(f(x_{i-1}) - \frac{h_i^2 m_{i-1}}{6}\right)\frac{x_i - x}{h_i} + \left(f(x_i) - \frac{h_i^2 m_i}{6}\right)\frac{x - x_{i-1}}{h_i}$$

для $x \in [x_{i-1}, x],\ i \in \{1, \dots, n\}$ де числа m_i є розв'язком системи рівнянь:

$$\begin{cases} m_0 = 0 \\ \frac{h_i m_{i-1}}{6} + \frac{h_i + h_{i+1}}{3}m_i + \frac{h_{i+1}m_{i+1}}{6} = -\frac{f(x_i) - f(x_{i-1})}{h_i} + \frac{f(x_{i+1}) - f(x_i)}{h_{i+1}},\ i \in \{1, \dots, n-1\} \\ m_n = 0 \end{cases}$$

де ми позначили $h_i := x_i - x_{i-1},\ i \in \{1, \dots, n\}$.

2.3 Метод прогонки

Нехай маємо лінійне рівняння виду $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ де матриця $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ має вигляд:

$$\boldsymbol{A} = \begin{bmatrix} d_1 & u_1 & 0 & 0 & \dots & 0 \\ \ell_2 & d_2 & u_2 & 0 & \dots & 0 \\ 0 & \ell_3 & d_3 & u_3 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \dots & \vdots \\ 0 & 0 & 0 & \dots & \ell_n & d_n \end{bmatrix}$$

Існує швидкий спосіб знаходження розв'язку $\mathbf{x} = [x_1, \ldots, x_n]^\top$, який називають методом прогонки. Для цього спочатку знаходять наступні значення (прямий хід):

$$\alpha_i = -\frac{u_i}{\ell_i \alpha_{i-1} + d_i},\ \beta_i = \frac{b_i - \ell_i \beta_{i-1}}{\ell_i \alpha_{i-1} + d_i},\ i \in \{1, \ldots, n\}$$

де для зручності ми поклали $\ell_1 = u_n = 0$. Далі застосовуємо зворотний хід:

$$\begin{aligned} x_n &= \beta_n, \\ x_{i-1} &= \alpha_{i-1} x_i + \beta_{i-1},\ i \in \{n, n-1, \ldots, 2\} \end{aligned}$$

для знаходження x_i.

Знаходження кубічного сплайну

Поєднуючи ідеї розділів 2.2 та 2.3, можемо знайти коефіцієнти m_i, що потрібні для побудови $S(x)$. З розділу 2.2 перепишемо рівняння для знаходження m_i наступним чином:

$$\begin{bmatrix} \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & 0 & \ldots & 0 & 0 \\ \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \ldots & 0 & 0 \\ 0 & \frac{h_3}{6} & \frac{h_3+h_4}{3} & \frac{h_4}{6} & \ldots & 0 & 0 \\ 0 & 0 & \frac{h_4}{6} & \frac{h_4+h_5}{3} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & \frac{h_{n-1}}{6} & \frac{h_{n-1}+h_n}{3} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} -\frac{f(x_1)-f(x_0)}{h_1} + \frac{f(x_2)-f(x_1)}{h_2} \\ -\frac{f(x_2)-f(x_1)}{h_2} + \frac{f(x_3)-f(x_2)}{h_3} \\ -\frac{f(x_3)-f(x_2)}{h_3} + \frac{f(x_4)-f(x_3)}{h_4} \\ \vdots \\ -\frac{f(x_{n-2})-f(x_{n-3})}{h_{n-2}} + \frac{f(x_{n-1})-f(x_{n-2})}{h_{n-1}} \\ -\frac{f(x_{n-1})-f(x_{n-2})}{h_{n-1}} + \frac{f(x_n)-f(x_{n-1})}{h_n} \end{bmatrix}$$

Тобто в нашому випадку $\ell_i = \frac{h_i}{6}, u_i = \frac{h_{i+1}}{6}, d_i = \frac{h_i+h_{i+1}}{3}, b_i = -\frac{f(x_i)-f(x_{i-1})}{h_i} + \frac{f(x_{i+1})-f(x_i)}{h_{i+1}}$.
Далі, застосовуючи метод прогонки з розділу 2.3, легко знаходимо m_i.

3 Текст програми

Повний текст програми можна знайти за цим посиланням (← напис клікабельний) на *Github* сторінку.

3.1 Генерація вузлів

Для початку, створимо файл `generators.py`, завантажимо залежності та створимо свій тип для інтервалу:

```python
from math import cos, pi
from typing import Tuple, TypeAlias, List
from abc import ABC, abstractmethod

Interval: TypeAlias = Tuple[float, float]
```

Лістинг 1: Завантаження залежностей

Зробимо генерацію вузлів через абстрактний клас, котрий буде вміти створюватись та генерувати набір x координат вузлів через функцію `generate_nodes`. Також, тут же задамо функцію `generate_test_points`, котра буде видавати набір x координат точок, на котрих ми будемо оцінювати інтерполяцію (як сказано в умові, використовуючи формулу $x_i^* = x_i + \alpha h$ де x_i це координата вузла).

```python
class IDataPointsGenerator(ABC):
    """Interface for generating data points"""

    def __init__(self, interval: Interval, number: int) -> None:
        """
        Function initializing the generator

        ### Args:
        - interval (‘Interval‘): Interval on which the data points will be generated
        - number (‘int‘): Number of data points to generate
        """

        assert number > 0, "Number of data points must be greater than 0"
        assert interval[0] < interval[1], "Lower bound must be less than upper bound"

        self._lower, self._upper = interval
        self._number = number

    @abstractmethod
    def generate_nodes(self) -> List[float]:
        """
        Function generating data points
```

```
        ### Returns:
            `List[float]`: List of generated x coordinates
        """
        pass

    @abstractmethod
    def generate_test_points(self, alpha: float = 0.1) -> List[float]:
        """
        Function generating test points for evaluating the polynomial
        ### Args:
        - alpha (`float`): We will evaluate the polynomial at points
    x_i + alpha*h

        Returns:
        - List[float]: List of x coordinates
        """

        nodes = self.generate_nodes()
        h = (self._upper - self._lower) / self._number
        return [node + alpha*h for node in nodes]
```

Лістинг 2: Задання абстрактного класу

Далі, наводимо конкретні реалізації цього інтерфейсу.

3.1.1 Лінійно розбитий проміжок

```
class LinearDataPointsGenerator(IDataPointsGenerator):
    """Class for generating linearly spaced data points"""

    def __init__(self, interval: Interval, number: int) -> None:
        super().__init__(interval, number)

    def generate_nodes(self) -> List[float]:
        fn = lambda i: self._lower + i * (self._upper - self._lower) /
    (self._number)
        return [fn(i) for i in range(self._number + 1)]

    def generate_test_points(self, alpha: float = 0.1) -> List[float]:
        return super().generate_test_points(alpha)
```

Лістинг 3: Генерація лінійно розкинутих точок

3.1.2 Проміжок розбитий по гармонічному закону

```
class CosineDataPointsGenerator(IDataPointsGenerator):
    """Class for generating cosine spaced data points"""

    def __init__(self, interval: Interval, number: int) -> None:
```

```
        super().__init__(interval, number)

    def generate_nodes(self) -> List[float]:
        fn = lambda i: 0.5*(self._lower+self._upper-(self._upper-self._lower)*cos((2*i+1)*pi/(2*(self._number+1))))
        return [fn(i) for i in range(self._number + 1)]

    def generate_test_points(self, alpha: float = 0.1) -> List[float]:
        return super().generate_test_points(alpha)
```

Лістинг 4: Генерація точок по закону з косинусом

3.1.3 Перевірка генерації

Перевіримо роботу програми, поклавши $n := 4$ для нашого конкретного інтервалу $[-2, 2]$. В коді, це виглядає так:

```
linear_generator = LinearDataPointsGenerator((-2.0, 2.0), 4)
cosine_generator = CosineDataPointsGenerator((-2.0, 2.0), 4)

print(f"Linear generator nodes: {linear_generator.generate_nodes()}")
print(f"Cosine generator nodes: {cosine_generator.generate_nodes()}")
```

Лістинг 5: Використання генераторів

Якщо запустити, отримаємо наступний результат:

```
Linear generator nodes: [-2.0, -1.0, 0.0, 1.0, 2.0]
Cosine generator nodes: [-1.902113032590307, -1.1755705045849463, -1.2246467991473532e-16, 1.175570504584946, 1.902113032590307]
```

Лістинг 6: Результат запуску генераторів

Перевіримо аналітично. У випадку лінійного розбиття, маємо:

$$x_k = -2 + k \cdot \frac{2+2}{4} = -2 + k$$

Дійсно, якщо підставляти $k \in \{0, \dots, 4\}$, отримаємо точки $\{-2, -1, 0, 1, 2\}$.

У випадку розбиття по косинусу:

$$\hat{x}_k = \frac{1}{2}\left(2 - 2 - (2+2)\cos\frac{2k+1}{10}\pi\right) = -2\cos\frac{(2k+1)\pi}{10}$$

Тому, наприклад, $\hat{x}_0 = -2\cos\frac{\pi}{10} \approx -1.902$. Аналогічно можна перевірити схожість для інших k. Отже, ми дійсно отримали схожі результати.

3.2 Метод прогонки

Створимо файл `solver.py` та реалізуємо алгоритм 2.3 у функції `tridiagonal_matrix_algorithm` котра приймає набір нижньодіагональних елементів $\{\ell_i\}_{i=2}^{n}$, наддіагональних елементів $\{u_i\}_{i=1}^{n-1}$, діагональні елементи $\{d_i\}_{i=1}^{n}$ та праву частину системи $\{b_i\}_{i=1}^{n}$ і повертає розв'язок рівняння:

```python
import numpy as np

def tridiagonal_matrix_algorithm(
    l: np.ndarray,
    u: np.ndarray,
    d: np.ndarray,
    b: np.ndarray
) -> np.ndarray:
    """
    Solve a system of linear equations with the tridiagonal matrix algorithm.
    ### Args:
        l (List[float]): Lower diagonal
        u (List[float]): Upper diagonal
        d (List[float]): Main diagonal
        b (List[float]): Right hand side
    ### Returns:
        List[float]: Solution to the system of linear equations
    """
    assert len(l) == len(u) == len(d) - 1 == len(b) - 1, "Invalid input shapes"

    l = np.insert(l, 0, 0.0, axis=0) # Add zero to the beginning of the list
    u = np.append(u, 0.0) # Add zero to the end of the list
    n = len(d) # Number of equations

    # Forward substitution
    alpha, beta = np.zeros(n), np.zeros(n)
    for i in range(n):
        alpha[i] = -u[i] / (l[i] * alpha[i-1] + d[i])
        beta[i] = (b[i] - l[i] * beta[i-1]) / (l[i] * alpha[i-1] + d[i])

    # Backward substitution
    x = np.zeros(n) # Defining a solution
    x[n-1] = beta[n-1]
    for i in range(n-2, -1, -1):
        x[i] = alpha[i] * x[i+1] + beta[i]
    return x
```

Лістинг 7: Реалізація методу прогонки

3.3 Кубічний сплайн

Створимо файл `spline.py` та завантажимо залежності:

```python
from math import prod
import numpy as np
import solver

from abc import ABC, abstractmethod
from typing import List, Tuple, TypeAlias

Point: TypeAlias = Tuple[float, float]
```

Лістинг 8: Завантаження залежностей

Зокрема, ми завантажили `solver.py` з попереднього розділу.

Задамо абстракцію для інтерполяційної функції (буде корисно, якщо у майбутньому це доведеться робити знову):

```python
class IInterpolate(ABC):
    """Interface any interpolating function should implement"""

    def __init__(self, points: List[Point]) -> None:
        """
        Initializes the interpolating function
        ### Args:
        - points (`List[Point]`): List of points to interpolate on
        """
        assert len(points) > 1, "At least two points are required"
        self._points = points

    @abstractmethod
    def evaluate(self, x: float) -> float:
        """
        Evaluates the interpolating function at x
        ### Args:
        - x (`float`): Point to evaluate the polynomial at
        ### Returns:
            `float`: Value of the polynomial at x
        """
        pass
```

Лістинг 9: Інтерфейс для інтерполяційної функції

Далі, наводимо реалізацію кубічного сплайну

```python
class CubicSpline(IInterpolate):
    """Cubic spline interpolating function"""

    def __init__(self, points: List[Point]) -> None:
        super().__init__(points)
```

```
6
7           # Defining the number of points
8           self._n = len(points)
9           # Finding h_{i} = x_{i} - x_{i-1}
10          self._differences = self._find_differences()
11          # Finding m_{i} coefficients
12          self._m = self._find_m_coefficients()
13
14      def _find_differences(self) -> np.ndarray:
15          """ Generates a list of pairwise differences between x
    coordinates of points
16
17          Returns:
18              np.ndarray: numpy array of length n-1 where n is the
    number of points
19          """
20          return np.array([self._points[i][0] - self._points[i-1][0] for
     i in range(1, self._n)])
21
22      def _find_m_coefficients(self) -> np.ndarray:
23          """ Generates a list of m coefficients needed for the
    evaluation
24
25          Returns:
26              np.ndarray: numpy array of length n where n is the number
    of points
27          """
28          l = np.array([self._differences[i] / 6.0 for i in range(1,
    self._n-2)]) # Initializing the lower diagonal
29          u = np.array([self._differences[i+1] / 6.0 for i in range(self
    ._n-3)]) # Initializing the upper diagonal
30          d = np.array([(self._differences[i] + self._differences[i+1])
    / 3.0 for i in range(self._n-2)]) # Initializing the main diagonal
31          b = np.array([-(self._points[i][1] - self._points[i-1][1]) /
    self._differences[i-1]
32                  + (self._points[i+1][1] - self._points[i][1]) / self.
    _differences[i] for i in range(1, self._n-1)]) # Initializing the
    right hand side
33
34          m = solver.tridiagonal_matrix_algorithm(l, u, d, b)
35          m = np.insert(m, 0, 0.0, axis=0) # Add zero to the beginning
    of the list
36          m = np.append(m, 0.0) # Add zero to the end of the list
37          return m
38
39      def evaluate(self, x: float) -> float:
40          # Defining a list of terms to sum
41          for i in range(1, self._n):
42              if self._points[i-1][0] <= x <= self._points[i][0]:
43                  return self._m[i-1] * (self._points[i][0] - x)**3 /
```

```
    (6.0 * self._differences[i-1]) \
44                  + self._m[i] * (x - self._points[i-1][0])**3 /
    (6.0 * self._differences[i-1]) \
45                  + (self._points[i-1][1] - self._m[i-1] * self.
    _differences[i-1]**2 / 6.0) * (self._points[i][0] - x) / self.
    _differences[i-1] \
46                  + (self._points[i][1] - self._m[i] * self.
    _differences[i-1]**2 / 6.0) * (x - self._points[i-1][0]) / self.
    _differences[i-1]
47
48          raise ValueError("x is not in the interval")
```

Лістинг 10: Реалізація кубічного сплайну

Пару коментарів по реалізації:

1. Ми знаходимо елементи $\{m_i\}_{i=0}^{n}$ за допомогою функції `_find_m_coefficients` під час ініціалізації.
2. У функції `evaluate` ми проходимось по всім парам точок і коли знаходимо потрібній сегмент, підставляємо все у формулу з розділу 2.2.

3.4 Програма для оцінки

Тепер напишемо програму, котра оцінює сплайн і видає потрібні нам таблиці. Створюємо файл `cli.py` і знову завантажуємо залежності:

```
1  # Math imports
2  from math import exp, sin, cos
3  import numpy as np
4
5  # Internal imports
6  from generators import Interval, IDataPointsGenerator,
       LinearDataPointsGenerator, CosineDataPointsGenerator
7  from spline import CubicSpline
8
9  # Rich logging
10 from rich.console import Console
11 from rich.table import Table
12
13 import matplotlib.pyplot as plt
14 from typing import Callable
```

Лістинг 11: Завантаження залежностей

Далі створюємо функцію, що створює 2 генератора, наш сплайн і будує 2 таблиці, як і просили у умові:

```
1  def evaluate_accuracy(
2      fn: Callable[[float], float],
```

```
    interval: Interval,
    segments_number: int = 20,
    alpha: float = 0.2,
) -> None:
    """Evaluates the accuracy of the spline

    ### Args:
    - fn (Callable[[float], float]): Function to interpolate
    - interval (`Interval`): Interval on which to plot the polynomial
    - segments_number (int, optional): _description_. Defaults to 20.
    - alpha (float, optional): Alpha parameter for the test points.
    Defaults to 0.2.
    """

    # Defining the generator and defining a set of points
    generators: dict[str, IDataPointsGenerator] = {
        "linear generation": LinearDataPointsGenerator(interval,
    segments_number),
        "cosine generation": CosineDataPointsGenerator(interval,
    segments_number)
    }

    # For rich logging
    console = Console()

    for generator_name, generator in generators.items():
        # Defining the points on which to interpolate the polynomial
        node_x = generator.generate_nodes()
        node_points = [(x, fn(x)) for x in node_x]

        # Defining the spline
        spline = CubicSpline(node_points)

        # Defining the test points
        test_x = generator.generate_test_points(alpha=alpha)
        test_points = [(x, fn(x)) for x in test_x]

        table = Table(title=f"Spline evaluation using {generator_name}
    ")
        table.add_column("x", justify="center", style="cyan", no_wrap=
    True)
        table.add_column("f(x)", justify="center", style="magenta")
        table.add_column("S(x)", justify="center", style="green")
        table.add_column("|f(x)-S(x)|", justify="center", style="blue"
    )

        for test_point in test_points[:-1]:
            x_label = "{:.18f}".format(test_point[0])
            f_x_label = "{:.18f}".format(test_point[1])
            p_x_label = "{:.18f}".format(spline.evaluate(test_point
```

```
    [0]))
47              difference_label = "{:.18f}".format(abs(test_point[1] -
    spline.evaluate(test_point[0])))
48
49              table.add_row(x_label, f_x_label, p_x_label,
    difference_label)
50
51          console.print(table)
```

Лістинг 12: Реалізація побудови таблиць з результатами

Також додатково побудуємо 2 графіки для порівняння: один буде зображати функцію $f(x)$ на проміжку $[\alpha, \beta]$, а другий сплайн $S(x)$:

```
1  def plot_polynomial(
2      fn: Callable[[float], float],
3      interval: Interval,
4      segments_number: int = 2,
5  ) -> None:
6      """
7      Plots the polynomial and the spline on the same plot
8
9      ### Args:
10     - fn (`Callable[[float], float]`): Function to plot
11     - interval (`Interval`): Interval on which to plot the polynomial
12     - segments_number (`int`): Number of segments to use for the
       spline
13
14     ### Returns:
15     Displayed polynomial and spline
16     """
17
18     # Defining the spline
19     generator = LinearDataPointsGenerator(interval, segments_number)
20     node_x = generator.generate_nodes()
21     node_points = [(x, fn(x)) for x in node_x]
22     spline = CubicSpline(node_points)
23
24     x = np.arange(*interval, 0.02)
25     plt.figure()
26     plt.subplot(211)
27     plt.plot(x, [fn(x) for x in x], 'b', x, [spline.evaluate(x) for x
       in x], 'r--')
28     plt.show()
```

Лістинг 13: Побудова графіків

Нарешті, викликаємо ці дві функції послідовно, ініціалізувавши нашу функцію і проміжок:

```
1  if __name__ == "__main__":
```

```
    # Defining the task parameters
    interval = (0.0, 1.0)
    segments_number = 20
    alpha = 0.2
    fn = lambda x: x**3 + cos(x) + exp(x / 10.0) * sin(x)

    # Evaluating the accuracy
    evaluate_accuracy(fn, interval, segments_number=segments_number,
    alpha=alpha)

    # Plotting the polynomial
    plot_polynomial(fn, interval, segments_number=3)
```

Лістинг 14: Вхід на програму

Для побудови графіків ми скористалися 3 сегментами, оскільки для більшої кількості функції повністю накладаються.

4 Результати

Для експериментів ми взяли $n = 20$ та $\alpha = 0.2$. Якщо ви хочете спробувати вибрати інші параметри, то можете запустити програму, що прикріплена у додатку :)

4.1 Лінійно розбитий проміжок

Spline evaluation using linear generation

x	f(x)	S(x)	|f(x)-S(x)|
0.010000000000000002	1.009960838585749920	1.009881970621430591	0.000078867964319329
0.060000000000000005	1.058741411967588197	1.058762538821313326	0.000021126853725129
0.110000000000000014	1.106279626109884617	1.106273959073755364	0.000005667036129253
0.160000000000000003	1.153211083223817379	1.153212595241071670	0.000001512017254290
0.210000000000000020	1.200175761329135726	1.200175349388090229	0.000000411941045497
0.260000000000000009	1.247818275597583737	1.247818378858803090	0.000000103261219353
0.309999999999999998	1.296788131212514239	1.296788096106098465	0.000000035106415774
0.359999999999999987	1.347739967170664244	1.347739968879828165	0.000000001709163922
0.410000000000000031	1.401333790467244711	1.401333781896601627	0.000000008570643084
0.460000000000000020	1.458235200121894115	1.458235194540623336	0.000000005581270779
0.510000000000000009	1.519115600520601461	1.519115592021368366	0.000000008499233095
0.560000000000000053	1.584652403567399670	1.584652402587378273	0.000000000980021397
0.609999999999999987	1.655529219159457499	1.655529189937182011	0.000000029222275488
0.660000000000000031	1.732436033520094565	1.732436108746882031	0.000000075226787466
0.709999999999999964	1.816069374946203574	1.816069059703459221	0.000000315242744353
0.760000000000000009	1.907132466549541494	1.907133607907962158	0.000001141358420664
0.810000000000000053	2.006335365595303788	2.006331070280590545	0.000004295314713243
0.859999999999999987	2.114395089066302358	2.114411083196936492	0.000015994130634134
0.910000000000000031	2.232035725106862323	2.231975997577113180	0.000059727529749143
0.959999999999999964	2.359988530027222975	2.360211399237170493	0.000222869209947518

Рис. 1: Результат для сплайну для лінійного розбиття

Графік, на якому ми зобразили порівняння сплайну та функції:

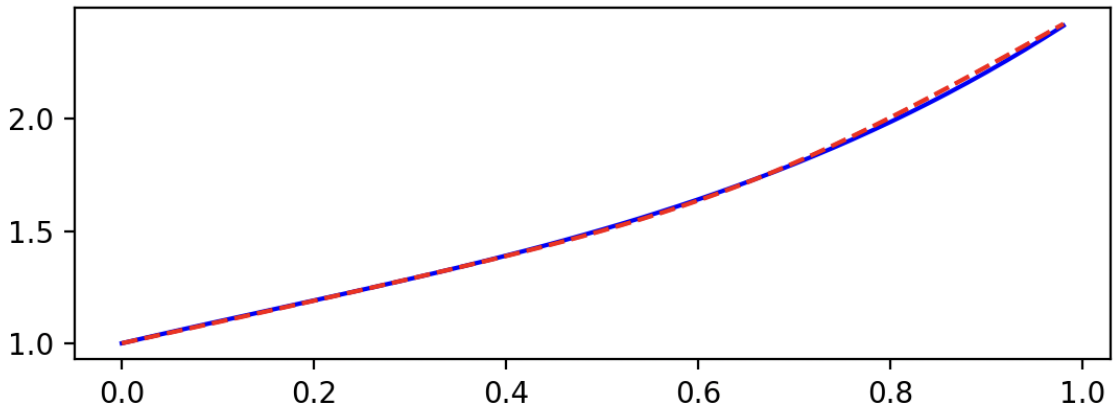

Рис. 2: Порівняння сплайну та функції. синім ми намалювали графік, а червоним пунктиром – сплайн

4.2 Проміжок розбитий по гармонічному закону

Spline evaluation using cosine generation

x	f(x)	S(x)	|f(x)-S(x)|
0.011398101409409937	1.011347376556598565	1.011346235556860318	0.000001140999738247
0.022536043909088192	1.022342496239935228	1.022346034185621422	0.000003537945686194
0.044563125677897879	1.043843067082767417	1.043841449715516845	0.000001617367250573
0.076987298107780655	1.074999222211988812	1.075000488684341615	0.000005566472352803
0.119084258765985107	1.114832801520816208	1.114832614257762033	0.000000187263054174
0.169913631114540276	1.162499856435996737	1.162499902173773858	0.000000045737777121
0.228339970968189032	1.217538035579843081	1.217538007669720423	0.000000027910122657
0.293058130441220921	1.280006139006429677	1.280006128629188389	0.000000010377241288
0.362622412794547988	1.350479441599602204	1.350479423222373798	0.000000018377228406
0.435478866911912676	1.429874085974469189	1.429874068268522702	0.000000017705946487
0.509999999999999898	1.519115600520601239	1.519115581353224886	0.000000019167376353
0.584521133088087175	1.618702061772642420	1.618702047698848956	0.000000014073793464
0.657377587205452141	1.728241359692897161	1.728241333574533467	0.000000026118363694
0.726941869558779041	1.846057372413028919	1.846057400995657627	0.000000028582628708
0.791660029031810986	1.968957355210169569	1.968957198381493967	0.000000156828675602
0.850086368885459631	2.092231399667561664	2.092231921316094478	0.000000521648532814
0.900915741234014855	2.209917089413021341	2.209915236162054875	0.000001853250966466
0.943012701892219196	2.315314787043107447	2.315321029724313640	0.000006242681206192
0.975436874322102243	2.401690330398300155	2.401671471244574363	0.000018859153725792
0.997463956090911763	2.463062223474222545	2.463077201666552085	0.000014978192329540

Рис. 3: Результат для сплайну для розбиття по косинусу

5 Висновки

В цій лабораторній роботі ми:

- навчилися будувати кубічний сплайн теоретично;
- навчилися писати комп'ютерну програму (на прикладі мови `Python`), що будує кубічний сплайн;
- оцінювати написану програму та діставати дані з експериментів.

Також, як бачимо, модуль різниці $|f(x^*) - S(x^*)|$ – дуже мала величина, що говорить про точність інтерполяції на заданому проміжку (оскільки ми брали набір точок x_i^*, що відносно близький до вузлів x_i).