**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Фундаментальные НаукиКафедра ИУ-6Математическое моделированиеГруппа ИУ6-63Б

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

**Отчет по домашнему заданию 3:**

**Аппроксимация функций Вариант 7**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | 28.04.2025 |  | Д.Г. Донских |
| Преподаватель: | дата, подпись  28.04.2025 |  | Ф.И.О.  Я.Ю. Павловский |
|  | дата, подпись |  | Ф.И.О. |

*Москва, 2025*

**Оглавление**

[Цель домашней работы 3](#_Toc196757350)

[Постановка задачи и исходные данные 3](#_Toc196757351)

[Краткое описание реализуемых методов 3](#_Toc196757352)

[Текст программы 4](#_Toc196757353)

[Результаты выполнения программы 8](#_Toc196757354)

[Анализ результатов 10](#_Toc196757356)

## Цель домашней работы

Изучение интерполяционного полинома Лагранжа, метода построения кубических сплайнов.

## Постановка задачи и исходные данные

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *x i* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| *y* i | 2,3 | 3,71 | 4,8 | 5,9 | 6,3 | 6,25 | 5,87 | 4,82 | 3,7 | 2,2 |

Рисунок 1 – условие домашней работы.

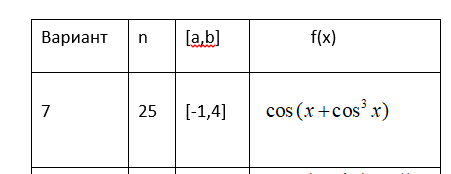
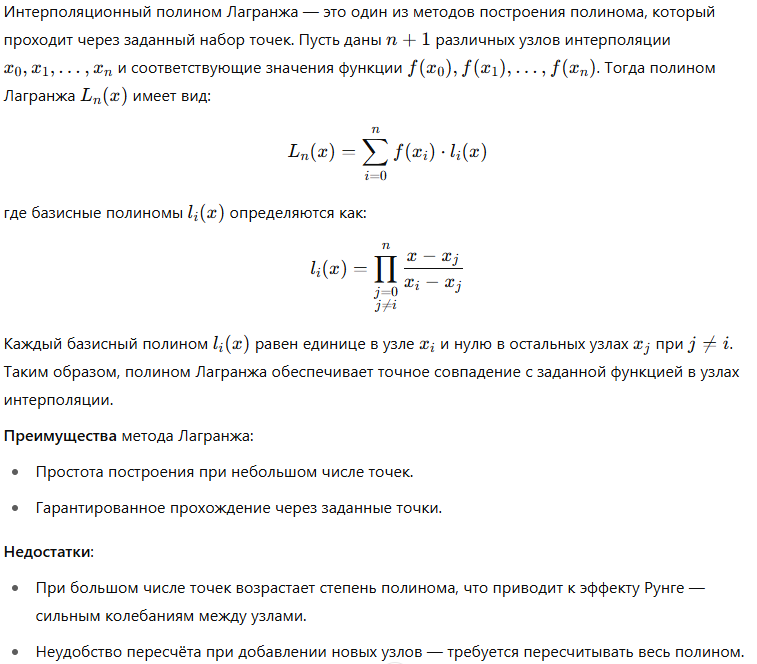


Рисунок 2 – условие домашней работы.

## Краткое описание реализуемых методов

**Интерполяционный полином Лагранжа**



**Кубические сплайны**

## Текст программы

Ниже в листингах приведены листинги кода проекта, реализованного на языке python версии 3.10

Вводим данные (файл data\_input.py) представлен в листинге 1:

Листинг 1 – функции инициализирующие данные моего варианта.

|  |
| --- |
| from math import cos  def get\_tabulated\_points():  xi = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  yi = [2.3, 3.71, 4.8, 5.9, 6.3, 6.25, 5.87, 4.82, 3.7, 2.2]  return xi, yi  def f(x):  return cos(x + cos(x)\*\*3)  def get\_spline\_function\_and\_range():  return 25, (-1, 4), f |

Метод простых итераций (файл lagrange.py) представлен в листинге 2:

Листинг 2 – функции lagrange\_polynomial, plot\_lagrange.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  from data\_input import get\_tabulated\_points  def lagrange\_polynomial(x, xi, yi):  """Вычисляет значение полинома Лагранжа в точке x."""  n = len(xi)  result = 0  for i in range(n):  term = yi[i]  for j in range(n):  if i != j:  term \*= (x - xi[j]) / (xi[i] - xi[j])  result += term  return result  def plot\_lagrange():  xi, yi = get\_tabulated\_points()  x = np.linspace(min(xi), max(xi), 500)  y = [lagrange\_polynomial(xi\_val, xi, yi) for xi\_val in x]  plt.figure(figsize=(10, 6))  plt.plot(x, y, label="Полином Лагранжа", color='blue')  plt.scatter(xi, yi, color='red', label="Узлы интерполяции")  plt.title("Интерполяция Полиномом Лагранжа")  plt.xlabel("x")  plt.ylabel("y")  plt.grid(True)  plt.legend()  plt.show()  if name == "\_\_main\_\_":  plot\_lagrange() |

Отрисовка метода кубических сплайнов (файл spline.py) представлен в листинге 3:

Листинг 3 – функция spline.py.

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  from custom\_spline import NaturalSpline |

Продолжение Листинга 3.

|  |
| --- |
| from data\_input import get\_spline\_function\_and\_range  def build\_cubic\_spline():  n, (a, b), f = get\_spline\_function\_and\_range()  xi = np.linspace(a, b, n)  yi = [f(x) for x in xi]  spline = NaturalSpline(xi, yi)  # для отрисовки  x\_dense = np.linspace(a, b, n)  x\_exact = np.linspace(a, b, 500)  y\_dense = spline(x\_dense)  y\_exact = [f(x) for x in x\_exact]  # Вывод значений в консоль (бонусом)  print(f"{'x':>10} {'Spline(x)':>15} {'Exact f(x)':>15} {'|Error|':>10}")  print("-" \* 55)  for x\_val, spline\_val, exact\_val in zip(x\_dense, y\_dense, y\_exact):  error = abs(spline\_val - exact\_val)  print(f"{x\_val:10.5f} {spline\_val:15.5f} {exact\_val:15.5f} {error:10.5e}")  plt.figure(figsize=(10, 6))  plt.plot(x\_exact, y\_exact, label="Точная функция", linestyle="--", color='green')  plt.plot(x\_dense, y\_dense, label="Кубический сплайн", color='blue')  plt.scatter(xi, yi, color='red', label="Узлы интерполяции")  plt.title("Интерполяция Кубическим Сплайном")  plt.xlabel("x")  plt.ylabel("y")  plt.grid(True)  plt.legend()  plt.show()  if name == "\_\_main\_\_":  build\_cubic\_spline() |

Метод кубических сплайнов (файл custom\_spline.py) представлен в листинге 4:

Листинг 4 – функция custom\_spline.py.

|  |
| --- |
| import numpy as np  from data\_input import f  class NaturalCubicSpline:  def \_\_init\_\_(self, x, y):  self.n = len(x) - 1  self.x = x  self.y = y  self.h = [x[i+1] - x[i] for i in range(self.n)]  # Решаем систему для коэффициентов c  self.a = y  self.c = self.\_solve\_c()  self.b = [0] \* self.n  self.d = [0] \* self.n  for i in range(self.n):  self.d[i] = (self.c[i+1] - self.c[i]) / (3 \* self.h[i])  self.b[i] = (self.a[i+1] - self.a[i]) / self.h[i] - self.h[i] \* (self.c[i+1] + 2\*self.c[i]) / 3  def \_solve\_c(self):  """Решение трёхдиагональной системы методом прогонки"""  n = self.n  A = [0] + [self.h[i-1] for i in range(1, n)]  B = [2 \* (self.h[i-1] + self.h[i]) for i in range(1, n)]  C = [self.h[i] for i in range(1, n)] + [0]  F = [0] \* (n-1)  for i in range(1, n):  F[i-1] = 3 \* ((self.a[i+1] - self.a[i]) / self.h[i] - (self.a[i] - self.a[i-1]) / self.h[i-1])  # Прямой ход  alpha = [0] \* (n-1)  beta = [0] \* (n-1)  alpha[0] = -C[0] / B[0]  beta[0] = F[0] / B[0]  for i in range(1, n-1):  denom = B[i] + A[i] \* alpha[i-1]  alpha[i] = -C[i] / denom  beta[i] = (F[i] - A[i] \* beta[i-1]) / denom |

Продолжение Листинга 4.

|  |
| --- |
| # Обратный ход  c = [0] \* (n+1)  for i in reversed(range(1, n)):  c[i] = alpha[i-1] \* c[i+1] + beta[i-1]  c[0] = c[n] = 0 # Свободные края  return c  def \_\_call\_\_(self, x\_val):  """Вычисляет значение сплайна в точке x\_val"""  # Находим нужный отрезок  i = self.\_find\_segment(x\_val)  dx = x\_val - self.x[i]  return self.a[i] + self.b[i]\*dx + self.c[i]\*dx\*\*2 + self.d[i]\*dx\*\*3  def \_find\_segment(self, x\_val):  """Бинарный поиск интервала"""  left = 0  right = self.n  while left <= right:  mid = (left + right) // 2  if self.x[mid] <= x\_val <= self.x[mid+1]:  return mid  elif x\_val < self.x[mid]:  right = mid - 1  else:  left = mid + 1  return self.n - 1 # если вдруг x\_val == x[-1] |

Ссылка на репозиторий: <https://github.com/Karielka/VichMat/blob/master/DZ3/>

## **Результаты выполнения программы**

Ниже представлены результаты выполнения файла lagrange.py и spline.py

~ cd DZ3

~ python lagrange.py

~ python spline.py

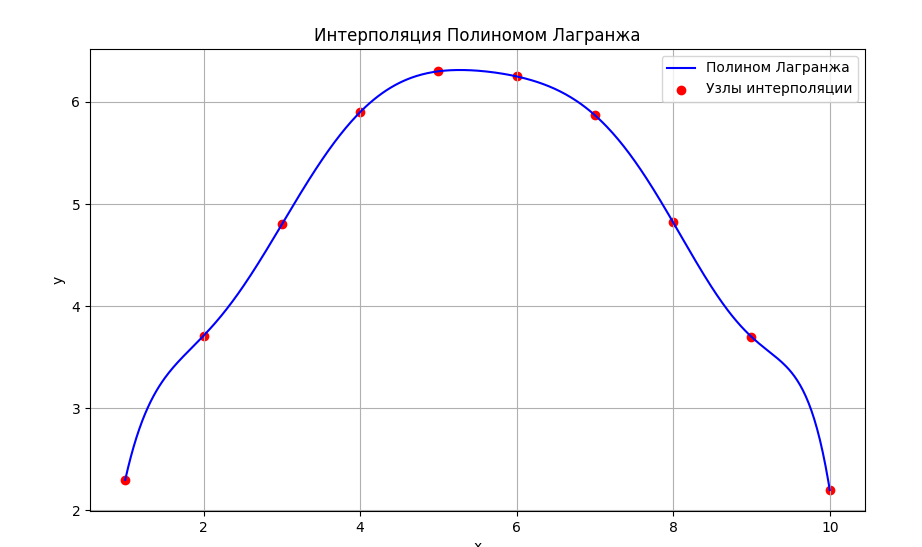


Рисунок 3 – результаты выполнения программы.

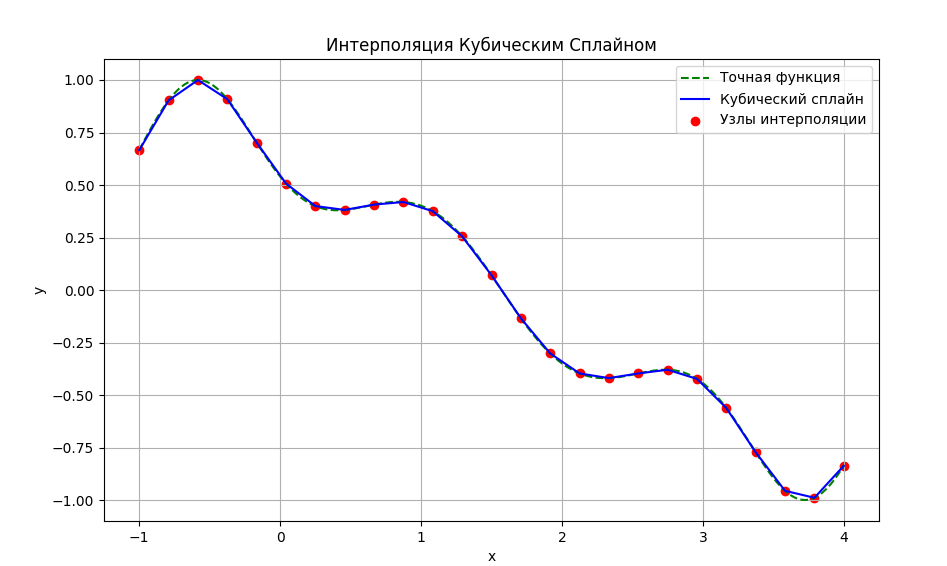


Рисунок 4 – результаты выполнения программы.

## 

Рисунок 5 – результаты выполнения программы.

## Анализ результатов

Все методы дали корректные результаты.

1. При сравнении значений функций в промежуточных точках (не совпадающих с узлами интерполяции) установлено, что кубический сплайн более точно воспроизводит поведение исходной функции по сравнению с полиномом Лагранжа. Ошибки сплайна в промежуточных точках оказались значительно меньше.
2. На практике метод кубических сплайнов оказался предпочтительнее для большого числа узлов, поскольку избегает резких колебаний и обеспечивает высокую точность аппроксимации. Метод Лагранжа целесообразно применять при малом количестве узлов или при необходимости точного восстановления значения функции в ограниченном числе точек.
3. Графики аппроксимации на заданной сетке продемонстрировали, что обе методики корректно выполняют поставленную задачу интерполяции. Однако различия в поведении между узлами становятся критичными при увеличении сложности функции.