Красникова Д.Я., Красников Д.Я., Славный Д.М.

Интерполяция отчет

Интерполяционный полином Лагранжа

Цель

Реализовать алгоритм вычисления приближенного значения функции по данным дискретным точкам. Отобразить график интерполяционного полинома Лагранжа для функции.

Постановка задачи

Разработать программу для интерполяции функции полиномом Лагранжа. Построить интерполяционный полином Лагранжа для функции по узлам.

Задание 1.

Реализовать задачу, рассмотренную на Лекции.
Использовать табличный процессор Excel или язык программирования

Начальные условия

Значение аргумента x = 0.263

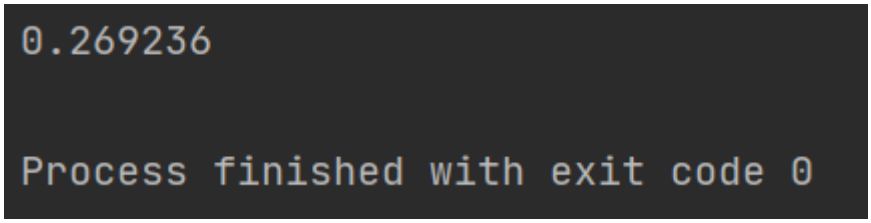
Значения узлов xi = [0.05, 0.10, 0.17, 0.25, 0.30, 0.36]

Значения функции yi = [0.050042, 0.100335, 0.171657, 0.255342, 0.309336, 0.376403]

Код программы

```
arg_x = 0.263  # Значение аргумента
# Узлы и значения функции
x = [0.05, 0.10, 0.17, 0.25, 0.30, 0.36]
y = [0.050042, 0.100335, 0.171657, 0.255342, 0.309336, 0.376403]
d = []

# Вычисляем разности и перемножаем их
for i in range(len(x)):
    mult = 1
    for j in range(len(x)):
        if i != j:
            difference = x[i] - x[j]
            mult *= difference
    d.append(round(mult * (arg_x - x[i]), 11))

# Суммируем частные
sum = 0
for i in range(len(x)):
    sum += y[i]/d[i]

p = 1
for i in range(len(x)):
    p *= arg_x - x[i]

print(round(p*sum, 6))
```

Результат



Задание 2.

Начальные условия

Определить значение функции y(x) при x = 0,1157.

Базовые значения следующие:

x	y
0.101	1.26183
0.106	1.27644
0.111	1.29122
0.116	1.30617
0.121	1.32130
0.126	1.32660

Значение аргумента x = 0.1157

Значения узлов xi = [0.101, 0.106, 0.111, 0.116, 0.121, 0.126]

Значения функции yi = [1.26183, 1.27644, 1.29122, 1.30617, 1.32130, 1.32660]

Код программы

```
arg_x = 0.1157  # Значение аргумента
# Узлы и значения функции
x = [0.101, 0.106, 0.111, 0.116, 0.121, 0.126]
y = [1.26183, 1.27644, 1.29122, 1.30617, 1.32130, 1.32660]
d = []
diffs = [[], [], [], [], [], []]



# Вычисляем разности и перемножаем их
for i in range(len(x)):
    mult = 1
    for j in range(len(x)):
        if i != j:
            difference = x[i] - x[j]
            mult *= difference
            diffs[i].append(round(difference, 6))
    diffs[i].insert(i, round(arg_x - x[i], 6))
    d.append(mult * (arg_x - x[i]))

# Суммируем частные
sum = 0
for i in range(len(x)):
    sum += y[i]/d[i]

p = 1
for i in range(len(x)):
    p *= arg_x - x[i]

print(round(p*sum, 6))


# Функции для вывода таблицы
def print_table_header():
    # Заголовки столбцов
    header = f"{'i':^5} {'differences':^50} {'di':^10} {'yi/di':^20}"
    # Разделительная линия
    separator = "-" * len(header)
```

```
    print(header)
    print(separator)


def print_table_row(i, di, yi_di, differences):
    # Форматирование строки таблицы
    str_differences = ', '.join([str(i) for i in differences])
    table_row = f"{i:^5} {str_differences:^50} {di:^10} {yi_di:^20}"
    print(table_row)


print_table_header()
for i in range(len(x)):
    print_table_row(i, round(d[i], 13), round(y[i]/d[i], 1),  diffs[i])
```

Из работы программы видно, что y = 1.30524, при данном x и базовых значениях.

```
1.30524


Process finished with exit code 0
```

Таблица промежуточных результатов (структура взята из лекции)

i	differences	di	yi/di
0	0.0147, -0.005, -0.01, -0.015, -0.02, -0.025	-5.5e-12	-228903401360.5
1	0.005, 0.0097, -0.005, -0.01, -0.015, -0.02	7e-13	1754556701030.9
2	0.01, 0.005, 0.0047, -0.005, -0.01, -0.015	-2e-13	-7326070921985.8
3	0.015, 0.01, 0.005, -0.0003, -0.005, -0.01	-0.0	-116103999999996.7
4	0.02, 0.015, 0.01, 0.005, -0.0053, -0.005	4e-13	3324025157232.7
5	0.025, 0.02, 0.015, 0.01, 0.005, -0.0103	-3.9e-12	-343456310679.6

Задание 3.

Найти для функции sin(πx) полином Лагранжа для данных узлов и его значения для аргумента равному x = 1/4 и x = 1/3.

Начальные условия

Значение аргумента x = [1/4, 1/3]

Значения узлов xi = [0, 1/6, 1/2]

Значения функции yi = [0.0, 0.5, 1.0]

Код программы

```
import math


def lagrange(x, y, arg_x):
    d = []  # Сохранение промежуточных вычислений разностей

    # Вычисляем разности
    for i in range(len(x)):
        diff = 1
        for j in range(len(x)):
            if i != j:
                diff *= (arg_x - x[j]) / (x[i] - x[j])
        d.append(diff)

    # Вычисляем значения многочлена Лагранжа в точке
    result = sum([y[i] * d[i] for i in range(len(x))])

    return round(result, 6)


# Определение узлов и соответствующих значений функции
x = [0, 1/6, 1/2]
y = [math.sin(math.pi * xi) for xi in x]
arg_x = [1/4, 1/3]

for xi in arg_x:
    yp = lagrange(x, y, xi)
    print(f"Значение интерполяционного полинома Лагранжа в точке x = {xi}
равно {yp:.5f}")
```

Результат

```
Значение интерполяционного полинома Лагранжа в точке x = 0.25 равно 0.68750
Значение интерполяционного полинома Лагранжа в точке x = 0.3333333333333333 равно 0.83333
```

Задание 4. Построить интерполяционный полином Лагранжа для функции $f(x) = x^2$ по узлам.

$x_1 = -1 \quad x_2 = 0 \quad x_3 = 1.$

В данном случае полином Лагранжа полностью совпадает с исходной функцией.

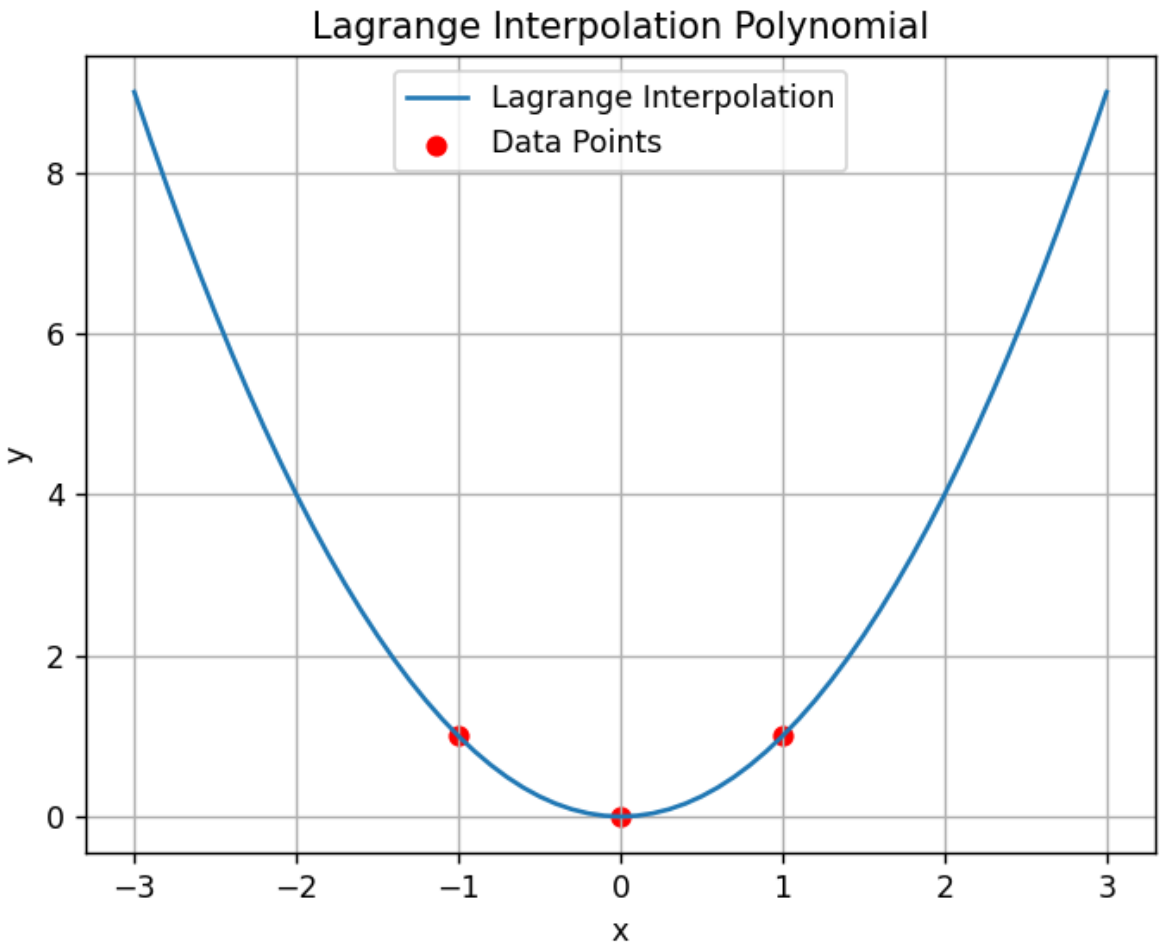
Начальные условия

x = [-1, 0, 1]
y = [1, 0, 1]

Код программы

```
import matplotlib.pyplot as plt


def lagrange(x, y, arg_x):
    d = []  # Сохранение промежуточных вычислений разностей

    # Вычисляем разности
    for i in range(len(x)):
        diff = 1
        for j in range(len(x)):
            if i != j:
                diff *= (arg_x - x[j]) / (x[i] - x[j])
        d.append(diff)

    # Вычисляем значения многочлена Лагранжа в точке
    result = sum([y[i] * d[i] for i in range(len(x))])

    return result


# Определение узлов и соответствующих значений функции
x = [-1, 0, 1]
y = [xi**2 for xi in x]

# Входные данные для построения графика
arg_x = [i/10 for i in range(-30, 31)]  # Генерация значений от -3 до 3
arg_y = [lagrange(x, y, val) for val in arg_x]

# Построение графика
plt.plot(arg_x, arg_y, label='Lagrange Interpolation')
plt.scatter(x, y, color='red', label='Data Points')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Lagrange Interpolation Polynomial')
plt.legend()
plt.grid(True)
plt.show()
```

Результат



Вывод: Интерполяционный многочлен Лагранжа полезный метод для вычисления промежуточных значений функции, который помогает предугадывать данные для сложных, экспериментальных и табличных графиков.

Интерполяционный полином Ньютона.

Цель

Реализовать алгоритм вычисления приближенного значения функции по данным дискретным точкам.

Постановка задачи

Разработать программу для интерполяции функции с помощью формулы Ньютона. Вычислить значения функции при заданных значения аргумента, используя инторполяционную формулу Ньютона для неравноотстояцщих узлов.

Задание 1.

Определить значения функции y(x) при следующих значениях аргумента: x1 = 0.112, x2 = 0.133

Начальные условия

Значения узлов xi = [0.103, 0.108, 0.115, 0.120, 0.128, 0.136, 0.141, 0.150]

Значения функции yi = [2.01284, 2.03342, 2.06070, 2.07918, 2.10721, 2.13354, 2.14922, 2.17609]

Для вычисления разделенных разностей использовалась формула:

$$f(x_0; x_1; \ldots; x_{n-1}; x_n) = \sum_{j=0}^{n} \frac{f(x_j)}{\prod_{\substack{i=0 \\ i \neq j}}^{n} x_j - x_i}$$

Эта формула реализована в программе на языке Python в функции под названием div_diff (divided difference). На вход ей передается смещение относительно списка аргументов и порядок распределенной разности.

Для расчета значения функции использовалась формула Ньютона. Для большей точности она использовалась два раза для разных значений x из списка, а затем находилось среднее арифметическое.

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} f(x_0; \ldots; x_k) \cdot \prod_{i=0}^{k-1} (x - x_i)$$

Код программы

```
x = [0.103, 0.108, 0.115, 0.120, 0.128, 0.136, 0.141, 0.150]
y = [2.01284, 2.03342, 2.06070, 2.07918, 2.10721, 2.13354, 2.14922, 2.17609]


def div_diff(bias, n):
    """
```

```
    Функция для вычисления разделенных разностей по заданному смещению и
размеру.

    Args:
        bias (int): Смещение для работы с отдельными значениями из
глобального списка.
        n (int): Размер порядка разделенной разности.

    Returns:
        float: Результат вычисления разделенных разностей.

    Raises:
        ValueError: Если параметр n превышает количество значений в списке с
учетом смещения.
    """

    global x, y  # Предполагается, что x и y являются глобальными переменными

    if n - bias > len(x):
        raise ValueError('Параметр n не может быть больше чем значений в
списке с учетом смещения')

    # Функция для вычисления произведения в знаменателе
    def product(j, n):
        prod = 1
        for i in range(bias, n + bias):
            if i == j:
                continue
            else:
                prod *= x[j] - x[i]
        return prod

    # Функция для вычисления суммы
    total = 0
    for j in range(bias, n + bias):
        total += y[j] / product(j, n)

    return total


print('Решение для условия x = 0.112')
print('    1. За x0 берем x0 = 0.103')
result1 = y[0] + div_diff(0, 2)*(0.112-0.103) + \
         div_diff(0, 3)*(0.112-0.103)*(0.112-0.106)
print('       f(0.112) ≈ ', round(result1, 5))

print('    2. За x0 берем x1 = 0.108')
result2 = y[1] + div_diff(1, 2)*(0.112-0.108) + \
         div_diff(1, 3)*(0.112-0.108)*(0.112-0.115)
print('       f(0.112) ≈ ', round(result2, 5))
print('Окончательный ответ f(0.112) ≈ ', round((result1 + result2) / 2, 5))

print()

print('Решение для условия x = 0.133')
print('    1. За x0 берем x3 = 0.120')
result1 = y[3] + div_diff(3, 2)*(0.133-0.120) + \
         div_diff(3, 3)*(0.133-0.120)*(0.133-0.128)
print('       f(0.133) ≈ ', round(result1, 5))

print('    2. За x0 берем x4 = 0.128')
result2 = y[4] + div_diff(4, 2)*(0.133-0.128) + \
         div_diff(4, 3)*(0.133-0.128)*(0.133-0.136)
```

```
print('       f(0.133) ≈ ', round(result2, 5))
print('Окончательный ответ f(0.133) ≈ ', round((result1 + result2) / 2, 5))
```

Результат

```
Решение для условия x = 0.112
    1. За x0 берем x0 = 0.103
       f(0.112) ≈  2.0489
    2. За x0 берем x1 = 0.108
       f(0.112) ≈  2.04921
Окончательный ответ f(0.112) ≈  2.04905

Решение для условия x = 0.133
    1. За x0 берем x3 = 0.120
       f(0.133) ≈  2.12387
    2. За x0 берем x4 = 0.128
       f(0.133) ≈  2.12385
Окончательный ответ f(0.133) ≈  2.12386
```

Задание 2.

Применяя первую интерполяционную формулу Ньютона, приближенно найти значение интеграла вероятностей в точке x = 1.43

Начальные условия

x = [1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]
y = [0.8427, 0.8802, 0.9103, 0.9340, 0.9523, 0.9661, 0.9763, 0.9838, 0.9891, 0.9928, 0.9953]

Для вычисления конечных разностей использовался следующий принцип:

1) $\Delta y_i = y_{i+1} - y_i$ конечные разности первого порядка,

2) $\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$ конечные разности второго порядка,

.....

3) $\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$ конечные разности k- ого порядка.

Эта формула реализована в программе на языке Python в функции под названием finite_diff (finite difference). На вход ей передается смещение относительно списка аргументов и порядок конечной разности.

Для расчета значения интеграла использовалась первая интерполяционная формула Ньютона.

$$y(x) = P_n(x) \approx y_0 + q\,\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \frac{q(q-1)(q-2)}{3!}\Delta^3 y_0$$

где $q = (x - x_0)/h$.

Код программы

```
x = [1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0]
y = [0.8427, 0.8802, 0.9103, 0.9340, 0.9523, 0.9661, 0.9763, 0.9838, 0.9891,
0.9928, 0.9953]


def finite_diff(bias, n):
    """
    Рекурсивная функция для вычисления разделенных разностей.

    Args:
        bias (int): Смещение для работы с отдельными значениями из
глобального списка y.
        n (int): Размер порядка разделенной разности.

    Returns:
        float: Результат вычисления разделенных разностей.
    """

    global y

    # Базовый случай: если n равно 1, вычисляем разность между двумя
соседними значениями
    if n == 1:
        return y[bias + 1] - y[bias]

    # Рекурсивно вычисляем разделенную разность n-го порядка через n-1
```

```
порядок разностей
    return finite_diff(bias + 1, n - 1) - finite_diff(bias, n - 1)


def coef_q(q, num):

    def fact(n):
        factorial_result = 1
        while n > 1:
            factorial_result *= n
            n -= 1
        return factorial_result

    prod = q
    for i in range(1, num):
        prod *= q - i

    return prod/fact(num)


x_value = 1.43
h = 0.1
q = (x_value - x[4]) / h

print('Решение для условия x = 1.43')
print('За x0 берем x5 = 1.4')
result = y[4] + coef_q(q, 1)*finite_diff(4, 1) + coef_q(q, 2)*finite_diff(4,
2) + \
         coef_q(q, 3)*finite_diff(4, 3)
print('f(1.43) ≈ ', round(result, 5))
```

Результат

```
Решение для условия x = 1.43
За x0 берем x5 = 1.4
f(1.43) ≈  0.95687
```

Задание 3.

Построить эмпирическую формулу для функции y, заданной таблично.

Начальные условия

x = [0, 1, 2, 3, 4, 5]
y = [5.2, 8.0, 10.4, 12.4, 14.0, 15.2]

Для вычисления конечных разностей использовался следующий принцип:

1) $\Delta y_i = y_{i+1} - y_i$ конечные разности первого порядка,

2) $\Delta^2 y_i = \Delta y_{i+1} - \Delta y_i$ конечные разности второго порядка,

.....

3) $\Delta^k y_i = \Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i$ конечные разности k- ого порядка.

Эта формула реализована в программе на языке Python в функции под названием finite_diff (finite difference). На вход ей передается смещение относительно списка аргументов и порядок конечной разности.

Для расчета значения интеграла использовалась первая интерполяционная формула Ньютона в форме.

$$y(x) = P_n(x) \approx y_0 + q\,\Delta y_0 + \frac{q(q-1)}{2!}\Delta^2 y_0 + \frac{q(q-1)(q-2)}{3!}\Delta^3 y_0$$

где $q = (x - x_0)/h$.

Данная программа реализована для упрощения алгебраического выражения, используя библиотеку SymPy в языке программирования Python. Она выполняет задачу упрощения сложных выражений, содержащих переменные, дроби и арифметические операции.

Программа формирует выражение на основе заданных значений x, y и других параметров, затем упрощает это выражение, с участием переменных и арифметических операций, для получения более компактного и понятного вида.

SymPy позволяет работать с символами, решать уравнения, упрощать выражения, находить производные и многое другое. Она широко используется для символьных вычислений в Python и обладает богатыми функциональными возможностями.

Код программы

```
from sympy import symbols, simplify, expand

x = [0, 1, 2, 3, 4, 5]
y = [5.2, 8.0, 10.4, 12.4, 14.0, 15.2]


def simplify_expression(expression):
    x, y = symbols('x y')
    expr = eval(expression)
    expanded_expr = expand(expr, mul=True)
    simplified_expr = simplify(expanded_expr)
    return str(simplified_expr)
```

```
def finite_diff(bias, n):
    global y
    if n == 1:
        return y[bias + 1] - y[bias]
    return finite_diff(bias + 1, n - 1) - finite_diff(bias, n - 1)


def fact(n):
    factorial_result = 1
    while n > 1:
        factorial_result *= n
        n -= 1
    return factorial_result


h = 1
q = f'(x - {x[0]})/{h}'
expression = f"{y[0]} + {finite_diff(0, 1)}*{q} + {round(finite_diff(0, 2) /
fact(2), 5)}*{q}*({q}-1)"
result = simplify_expression(expression)
print(result)
```

Результат

```
-0.2*x**2 + 3.0*x + 5.2
```

Вывод: Интерполяционная формула Ньютона на практике полезна, потому что число используемых узлов может быть увеличено или уменьшено без повторения всех предыдущих вычислений. Первую интерполяционную формулу Ньютона выгодно использовать для интерполирования функции в окрестности начального значения x0, где q мало по абсолютной величине.

Интерполяционный полином Стерлинга.

Цель

Реализовать алгоритм вычисления приближенного значения функции по данным дискретным точкам.

Постановка задачи

Разработать программу для интерполяции функции с помощью полинома Стерлинга. Вычислить значение функции при заданных значения аргумента, используя интерполяционный полинома Стерлинга.

Задание 1.

Вычислить значение функции в точке 4,3. Используя полином Стерлинга.

Даны значения функции:

x	0	1	2	3	4	5	6	7
y	0	2	5	10	15	20	22	24

Выбираем нулевой узел интерполяции. Берем ближайший узел к точке 4,3. В данном случае это будет узел слева от интересующей нас точки. Его принимаем за нулевой узел интерполяции. То есть узел $x_0 = 4$.

Справа и слева выбираем нужное количество точек. Можем взять три точки вправо и три точки влево. $x_1 = 5$ $x_2 = 6$ $x_3 = 7$ $x_{-1} = 3$ $x_{-2} = 2$ $x_{-3} = 1$. В программе для удобства реализации запишем их в списке x = [4, 5, 6, 7, 3, 2, 1]. Эти точки будем использовать для построения интерполяционного полинома Стерлинга и вычисления значения в точке x = 4,3.

Формула для вычисления имеет вид:

$$P_n(x) = y_0 + \sum_{i=1}^{n} \left(\frac{u}{(2i-1)!} \left(\prod_{j=1}^{i-1} (u^2 - j^2) \right) \frac{\Delta^{2i-1} y_{-(i-1)} + \Delta^{2i-1} y_{-i}}{2} + \frac{u^2}{(2i)!} \left(\prod_{j=1}^{i-1} (u^2 - j^2) \right) \Delta^{2i} y_{-i} \right)$$

Δ^{2i-1} – это разделенные разности

$u = \frac{x - x_0}{h}$

x – значение $f(x)$, в кот. хотим вычислить (у нас $x = 4,3$)

x_0 – значение нулевого узла интерполяции (в нашем примере $x_0 = 4$)

h – это расстояние между узлами интерполяции (в примере $h = 1$)

Для более понятной структуры программы дополнительно использованы функции для: вычисления факториала (функция fact), вычисления конечной разности (функция finite_diff) и вычисления сокращенного произведения (функция product).

Главная функция, в которой происходит вычисление значения функция через интерполяционную формулу Стирлинга, это stirling_polinomial.

Функция stirling_polynomial(x_value, h) реализует вычисление значения функции в точке с использованием полинома Стирлинга.

Параметры:

- x_value: Точка, для которой вычисляется значение функции.

- h: Шаг интерполяции.

Алгоритм:

1. Инициализация переменных и расчет переменной u.

2. Итерационный расчет суммы полинома Стирлинга для каждого слагаемого.

3. Учет разделенных разностей, факториалов и произведений для формирования полинома.

Код программы

```
x = [4, 5, 6, 7, 1, 2, 3]
y = [15, 20, 22, 24, 2, 5, 10]


def fact(n):
    factorial_result = 1
    while n > 1:
        factorial_result *= n
        n -= 1
    return factorial_result


def finite_diff(bias, n):
    global y
    if n == 1:
        return y[bias + 1] - y[bias]
    elif n <= 0:
        return y[0]
    return finite_diff(bias + 1, n - 1) - finite_diff(bias, n - 1)


def product(i, u):
    # Если переданный параметр меньше 1, вернуть 1
    if i < 1:
        return 1

    # Иначе вычислить произведение от j=1 до n
    prod = 1
    for j in range(1, i + 1):
        prod *= u ** 2 - j ** 2

    return prod
```

```
def stirling_polynomial(x_value, h):
    # x = lst_x
    # y = lst_y
    result = y[0]
    u = round((x_value - x[0]) / h, 3)
    n = int((len(x) - 1) / 2)
    for i in range(1, n+1):
        result += (u / fact(2 * i - 1)) * \
                  (product(i - 1, u) * (finite_diff(-(i-1), 2*i-1) +
finite_diff(-i, 2*i-1)) / 2) + \
                  ((u**2/fact(2*i))*product(i-1, u)*finite_diff(-i, 2*i))

    return result


h = 1
x_value = 4.3
result = stirling_polynomial(x_value, h)
print(round(result, 5))
```

Результат

```
16.6025


Process finished with exit code 0
```

Задание 2.

Построить функцию с найденным значением в указанной точке ($x_0 = 4.3$).

Чтобы построить график интерполирующей функции, основанной на полиноме Стирлинга и отметить точки, через которые она проходит, а также точку, в которой мы вычисляем значение, нам понадобится использовать библиотеку Python для построения графиков. Одним из популярных инструментов для этого является matplotlib.

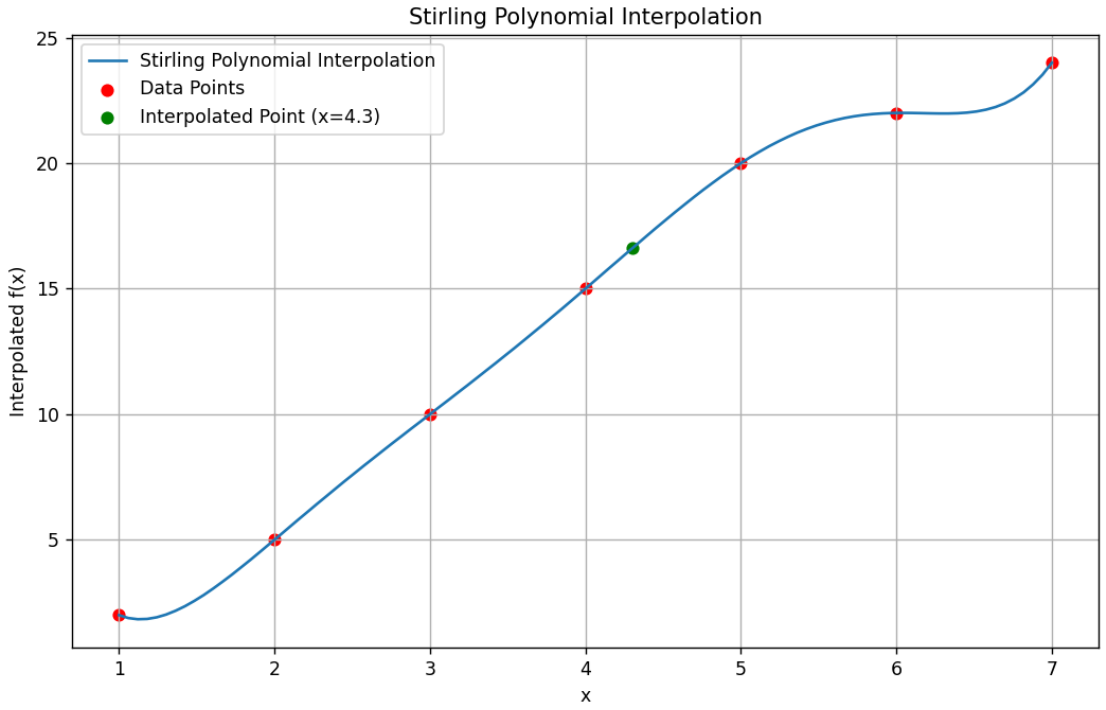
Создадим график для интерполирующей функции, используя данные x и y, а также точку x_value = 4.3, через которую проходит наша интерполяционная функция.

Код программы

```
import matplotlib.pyplot as plt
import numpy as np

x = [4, 5, 6, 7, 1, 2, 3]
y = [15, 20, 22, 24, 2, 5, 10]


def fact(n):
    factorial_result = 1
    while n > 1:
        factorial_result *= n
        n -= 1
    return factorial_result


def finite_diff(bias, n):
    global y
    if n == 1:
        return y[bias + 1] - y[bias]
    elif n <= 0:
        return y[0]
    return finite_diff(bias + 1, n - 1) - finite_diff(bias, n - 1)


def product(i, u):
    # Если переданный параметр меньше 1, вернуть 1
    if i < 1:
        return 1

    # Иначе вычислить произведение от j=1 до n
    prod = 1
    for j in range(1, i + 1):
        prod *= u ** 2 - j ** 2

    return prod


def stirling_polynomial(x_value, h):
    result = y[0]
    u = round((x_value - x[0]) / h, 3)
    n = int((len(x) - 1) / 2)
    for i in range(1, n+1):
        result += (u / fact(2 * i - 1)) * \
                  (product(i - 1, u) * (finite_diff(-(i-1), 2*i-1) +
finite_diff(-i, 2*i-1)) / 2) + \
                  ((u**2/fact(2*i))*product(i-1, u)*finite_diff(-i, 2*i))

    return result
```

```
h = 1
x_value = 4.3
result = stirling_polynomial(x_value, h)
print(round(result, 5))

# Подготовим данные для графика
x_points = np.linspace(min(x), max(x), 100)
y_points = [stirling_polynomial(point, h) for point in x_points]

plt.figure(figsize=(10, 6))
plt.plot(x_points, y_points, label="Stirling Polynomial Interpolation")
plt.scatter(x, y, color='red', label="Data Points")
plt.scatter(x_value, result, color='green', label="Interpolated Point
(x=4.3)")

plt.xlabel('x')
plt.ylabel('Interpolated f(x)')
plt.title('Stirling Polynomial Interpolation')
plt.legend()
plt.grid(True)
plt.show()
```

Результат



Вывод: Интерполяционный полином Стерлинга является частным случаем интерполяционного полинома Ньютона. Его имеет смысл использовать только в случае, если узлы интерполяции находятся на фиксированном расстоянии между собой (равноотстоящие узлы). Его преимущество в том, что он строится и вычисляется более быстро, чем интерполяционный полином Ньютона. В отличии от двух интерполяционных формул Ньютона, которые еще называют вперед и назад, интерполяционный полином Стерлинга удобно применять, если нужно проводить вычисление в центре.