МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информатики и прикладной математики

Кафедра прикладной математики и экономико-математических методов

ЛАБОРАТОРНАЯ РАБОТА

на тему:

«Интерполирование функции одной или двух переменных»

метод:

«Первая интерполяционная формула Ньютона (в начале таблицы) - 1.1.8а»

Направление (специальность)_____01.03.02_____
(код, наименование)

Направленность (специализация)_____

Обучающийся _____Бронников Егор Игоревич_____
(Ф.И.О. полностью)

Группа___ПМ-1901__
(номер группы)

Проверил _____Хазанов Владимир Борисович_____
(*Ф.И.О. преподавателя)*

Должность_____профессор_____

Оценка _____ Дата: _____

Подпись: _____

Санкт-Петербург

2021

Оглавление

1. НЕОБХОДИМЫЕ ФОРМУЛЫ

Данные:

$[x_0, x_1]$ – *интервал*

h – *шаг интерполяции*

$f(x)$ – *функция*

Формулы:

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

где $\Delta^k y_0$ – *конечные разности для первого узла*, $\Delta^k y_0 = \Delta(\Delta y_0) = \Delta^{k-1} y_1 - \Delta^{k-1} y_0$

$q = \frac{x - x_0}{h}$, *где* x_0 – *значение функции в первом узле*, h – *шаг интерполяции*

2. ВХОДНЫЕ ДАННЫЕ

$[1,17]$ – *интервал*

$h=2.5$ – *шаг интерполяции*

$f(x)=\ln(x)+\sqrt{1+x}$ – *функция*

3. СКРИНШОТЫ ПРОГРАММЫ

Импорт модулей

```
import numpy as np                                  # для работы с матрицами и веторами
import matplotlib.pyplot as plt                     # для отрисовки графиков
import sympy as sp                                  # для красивого вывода математических объектов
from math import factorial                          # для нахождения факториала
import functools                                    # немного функционального программирования
from IPython.display import Markdown, display       # для красивого вывода текста

%matplotlib inline
```

```
plt.style.use('seaborn-poster')  # стиль графиков
```

Входные данные

```
def f(x):
    return np.log(x) + np.sqrt(1+x)
```

```
x_d = sp.Symbol('x')
display(Markdown(f"<text style=font-size:16px;font-family:serif>f(x) = </text> {sp.latex(sp.log(x_d) + sp.sqrt(1+x_d), mode='inline')}"))
```

f(x) = $\sqrt{x + 1} + \log(x)$

```
h = 2.5
x_0 = 1
x_1 = 17
```

```
x = np.arange(x_0, x_1, h)
y = f(x)
display(Markdown(f"<text style=font-size:16px;font-family:serif>x = {list(x)}<br>f(x) = {list(y)} </text>"))
```

x = [1.0, 3.5, 6.0, 8.5, 11.0, 13.5, 16.0]
f(x) = [1.4142135623730951, 3.3740833120550104, 4.437510780292646, 5.222273164980759, 5.861996887936125, 6.410576238376338, 6.895694347857441]

Находим конечные разности

```
def finite_differences(y: np.array) -> np.matrix:
    ''' finite_differences - составляет таблицу конечных разностей

        Аргументы:
            * y: np.array - список значений `f(x)`

        Возвращает:
            np.matrix - таблица конечных разностей
    '''

    n = len(y)                # длина списка f(x)
    coef = np.zeros([n, n])   # заполняем матрицу конечных разностей нулями
    coef[:, 0] = y            # пусть первая колонка у нас будет f(x)
    for j in range(1, n):     # итерируемся начиная со второй колонки
        for i in range(n-j):
            coef[i][j] = coef[i+1][j-1]-coef[i][j-1]  # по формуле находим все конечные разности
    return coef
```

```
a = finite_differences(y)
sp.Matrix(a)
```

$$\begin{bmatrix} 1.4142135623731 & 1.95986974968192 & -0.89644228144428 & 0.617777197894757 & -0.484150776077982 & 0.404418643478801 & -0.350897668541171 \\ 3.37408331205501 & 1.06342746823764 & -0.278665083549523 & 0.133626421816775 & -0.0797321325991809 & 0.0535209749376295 & 0.0 \\ 4.43751078029265 & 0.784762384688113 & -0.145038661732747 & 0.0538942892175944 & -0.0262111576615514 & 0.0 & 0.0 \\ 5.22227316498076 & 0.639723722955366 & -0.0911443725151528 & 0.0276831315560431 & 0.0 & 0.0 & 0.0 \\ 5.86199688793613 & 0.548579350440213 & -0.0634612409591098 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.41057623837634 & 0.485118109481103 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.89569434785744 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Интерполяция

Вывод полинома

```
def newton_function(x: np.array, y: np.array, *, var: bool = False) -> None:
    ''' newton_function - выводит на экран интерполяционный полином

        Аргументы:
            * x: np.array - список значений `x`
            * y: np.array - список значений `f(x)`
            ** var: bool - определяет от какого аргумента будет полином от `x` или от `q`
                True - `x`
                False - `q`, где `q` = (x - x_0)/h

        Возвращает:
            None - но печатает на экран интерполяционный полином
    '''

    coef = finite_differences(y)                    # таблица конечных разностей
    q = f'(x-{x[0]})/{x[1]-x[0]}' if var else 'q'   # определяем какой аргумент будет у полинома
    arg = 'x' if var else 'q'
    variables = [list(map(lambda x: f'({q} - {x})', np.arange(0, i))) for i in range(len(y))]  # находим аргументы при коэффициентах
    lst = []  # список со слагаемыми
    for i in range(len(variables)):
        if len(variables[i]) > 0:  # если у нас есть аргумент в слагаемом
            lst.append(str(coef[0, i]) + '*' + functools.reduce((lambda x, y: f'{x}*{y}'), variables[i]) + '/' + str(factorial(i)))  # True: рассчитываем по формуле слагаемое
        else:
            lst.append(str(coef[0, i]))  # False: добавляем коэффициент
    expression = functools.reduce((lambda x, y: f'{x} + {y}'), lst)  # получаем наше выражение
    display(Markdown(f"<text style=font-size:18px;font-family:serif>P({arg}) = </text> {sp.latex(sp.simplify(expression), mode='inline')}"))  # выводим его
```

```
newton_function(x, y)
```

$$P(q) = -0.0004873578729738490 3q^6 + 0.010680523456931076q^5 - 0.095299921829259811q^4 + 0.45161151943572122q^3 - 1.2810560040162616q^2 + 2.8744209905077583q + 1.4142135623730951$$

```
newton_function(x, y, var=True)
```

$$P(x) = -1.9962178477008866 \cdot 10^{-6}x^6 + 0.00012134586728517958x^5 - 0.0030164640675394364x^4 + 0.039795459198146133x^3 - 0.30744006923693995x^2 + 1.6567332613233629x + 0.028022025506627992$$

Процесс интерполяции

```
def newton_interpolation(x: float, x_data: np.array, y: np.array) -> float:
    ''' newton_interpolation - находим значение полинома в точке `x`

        Аргументы:
            * x: float - точка в которой ищется значение полинома
            * x_data: np.array - список исходных значений `x`
            * y: np.array - список исходных значений `f(x)`

        Возвращает:
            float - значение полинома в точке `x`
    '''

    coefs = finite_differences(y)  # таблица конечных разностей
    h = x_data[1] - x_data[0]      # `h` - шаг
    q = (x - x_data[0])/h          # находим `q`
    mul = 1
    s = 0
    for i in range(len(y)):
        mul = 1
        for j in range(i):
            mul *= q-j             # находим значение аргумента в точке `x`
        s += coefs[0, i]*mul/factorial(i)  # находим сумму всех слагаемых
    return s
```

4. РЕЗУЛЬТАТЫ И ГРАФИКИ

Результаты

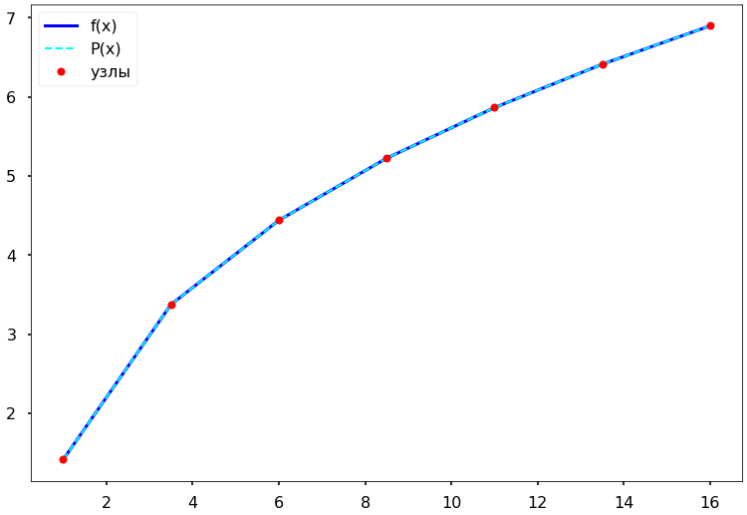
```
res = [newton_interpolation(i, x, y) for i in x]
comp_res = [np.isclose(res[i], y[i]) for i in range(len(res))]
display(Markdown(f"<text style=font-size:16px;font-family:serif>Результат: {res}<br>Значения функции: {list(y)}<br>Сравнение значений в заданных узлах: {comp_res} </text>"))
```

Результат: [1.4142135623730951, 3.3740833120550104, 4.437510780292646, 5.222273164980759, 5.861996887936125, 6.41057623837634, 6.895694347857444]
Значения функции: [1.4142135623730951, 3.3740833120550104, 4.437510780292646, 5.222273164980759, 5.861996887936125, 6.410576238376338, 6.895694347857441]
Сравнение значений в заданных узлах: [True, True, True, True, True, True, True]

Графики

```
newtonxs = newton_interpolation(x, x, y)
```

```
plt.plot(x, y, color='blue', label='f(x)', linewidth=3)
plt.plot(x, newtonxs, linestyle='dashed', color='cyan', linewidth=2, label='P(x)')
plt.plot(x, newtonxs, 'bo', markersize=8, color='red', label='узлы')
plt.rcParams['figure.figsize'] = (10, 10)
plt.legend();
```




```
xs = np.linspace(x_0, x_1, 50)
ys = f(xs)
newtonxs = [newton_interpolation(p, xs, ys) for p in xs]
```

```
plt.plot(xs, ys, color='blue', label='f(x)', linewidth=3)
plt.plot(xs, newtonxs, linestyle='dashed', color='cyan', linewidth=2, label='P(x)')
plt.rcParams['figure.figsize'] = (10,10)
plt.legend();
```