



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 «Информатика и системы  
управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные  
технологии»

### Лабораторная работа № 3

**Тема** Построение и программная реализация алгоритма  
сплайн-интерполяции табличных функций.

**Студент** Андреев А.А.

**Группа** ИУ7-44Б

**Оценка (баллы)** \_\_\_\_\_

**Преподаватель** \_\_\_\_\_

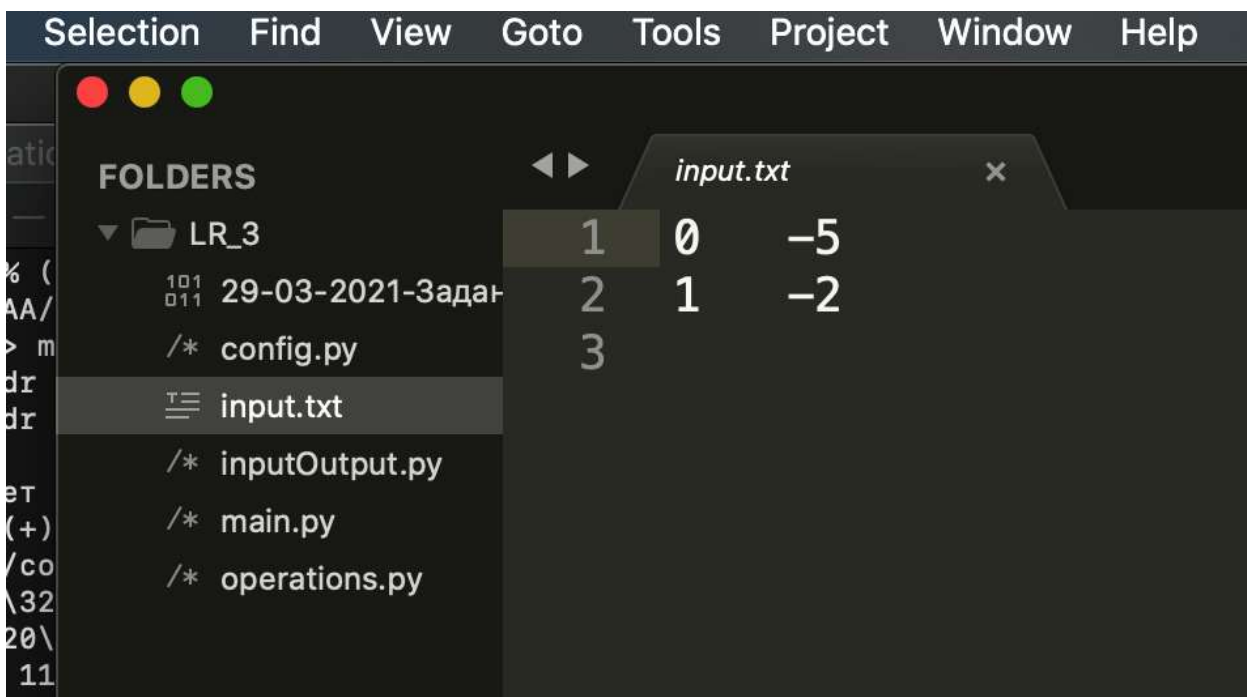
Москва.  
2021 г.

# Описание работы

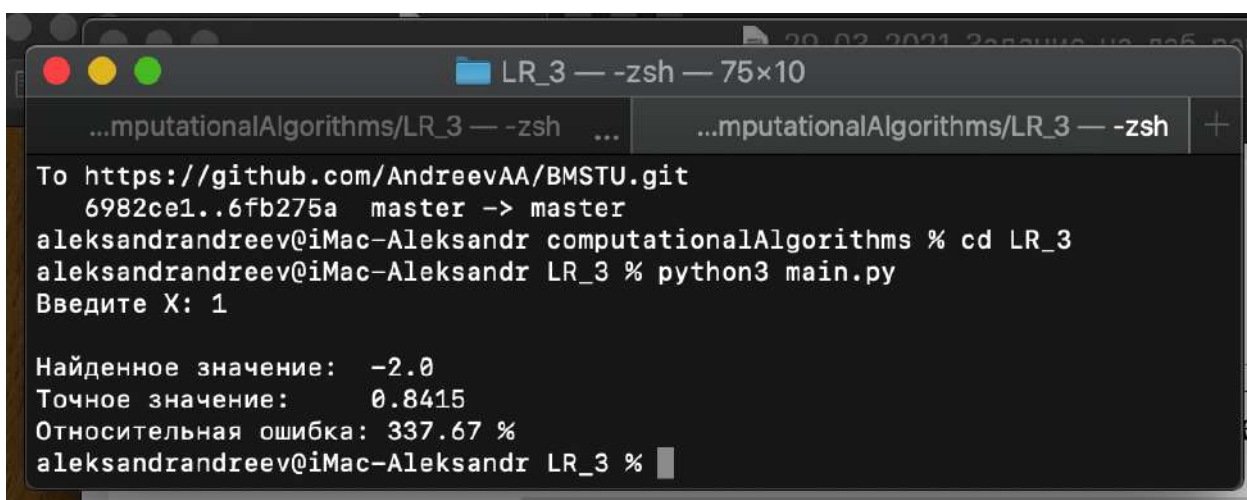
**Тема:** Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

**Цель работы:** реализовать нахождение значения функции методом кубического сплайна.

**Входные данные:** таблица исходных данных; аргумент функции.



**Выходные данные:** Результат интерполяции, точное значение функции, относительная ошибка.



# Алгоритм выполнения

## Сплайн.

**Сплайн** - функция, область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым алгебраическим многочленом (полиномом). Максимальная из степеней использованных полиномов называется степенью сплайна.

Интерполяция кубическими сплайнами является частным случаем кусочно-полиномиальной интерполяции. В этом специальном случае между любыми двумя соседними узлами функция интерполируется кубическим полиномом. его коэффициенты на каждом интервале определяются из условий сопряжения в узлах:

$$f_i = y_i, f'(x_i - 0) = f'(x_i + 0), f''(x_i - 0) = f''(x_i + 0), i = 1, 2, \dots, n-1.$$

Будем искать кубический полином в виде

$$f(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, x_{i-1} \leq x \leq x_i.$$

Из условия  $f_i = y_i$  имеем

$$f(x_{i-1}) = a_i = y_{i-1}, f(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = y_i, h_i = x_i - x_{i-1}, i = 1, 2, \dots, n-1.$$

Вычислим производные:

$$f'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{i-1})^2, f''(x) = 2c_i + 6d_i(x - x_{i-1}), x_{i-1} \leq x \leq x_i,$$

и потребуем их непрерывности при  $x = x_i$ :

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2, c_{i+1} = c_i + 3d_i h_i, i = 1, 2, \dots, n-1.$$

Общее число неизвестных коэффициентов, очевидно, равно  $4n$ , число уравнений (4) и (5) равно  $4n-2$ . Недостающие два уравнения получаем из условия (2) при  $x = x_0$  и  $x = x_n$ :

$$c_1 = 0, c_n + 3d_n h_n = 0.$$

Выражение из (5)  $d_i = \frac{c_{i+1} - c_i}{3h_i}$ , подставляя это выражение в (4) и исключая  $a_i = y_{i-1}$ , получим

$$b_i = \left[ \frac{y_i - y_{i-1}}{h_i} \right] - \frac{1}{3} h_i (c_{i+1} + 2c_i), i = 1, 2, \dots, n-1, b_n = \left[ \frac{y_n - y_{n-1}}{h_n} \right] - \frac{2}{3} h_n c_n.$$

Подставив теперь выражения для  $b_i, b_{i+1}$  и  $d_i$  в первую формулу (5), после несложных преобразований получаем для определения  $c_i$  разностное уравнение второго порядка

$$h_i c_i + 2(h_i + h_{i+1}) c_{i+1} + h_{i+1} c_{i+2} = 3 \left( \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \right), i = 1, 2, \dots, n-1.$$

С краевыми условиями

$$c_1 = 0, c_{n+1} = 0.$$

## Метод прогонки

Метод прогонки, основан на предположении, что искомые неизвестные связаны рекуррентным соотношением:

$$y_i = \beta_{i+1} y_{i+1} + \gamma_{i+1}$$

## Алгоритм метода прогонки

1. (Прямой ход). Вычисление коэффициентов прогонки через рекуррентные формулы, с соответствующими начальными значениями.

$$z_{i+1} = \frac{D_i}{B_i - A_i z_i}, \quad \eta_{i+1} = \frac{F_i + A_i \eta_i}{B_i - A_i z_i}, \quad \begin{matrix} z_1 = \frac{M_0}{k_0} \\ \eta_1 = \frac{P_0}{k_0} \end{matrix}$$

2. (Обратный ход). При известных коэффициентах прогонки и найденном  $Y_n$  по рекуррентной формуле находятся все  $Y_i$ .

$$y_i = z_{i+1} y_{i+1} + \eta_{i+1}$$

# Код программы

Программа состоит из 5 файлов.

## *config.py*

```
# Статусы ошибок
SUCCESS_STATUS = 0
ERROR_STATUS = 1
```

## *input.txt*

```
0      -5
1      -2
```

## *inputOutput.py*

```
# Подгрузка данных из файла
def uploadData(inputDataFileName):
    inputDataTable = []; tempInputFile = open(inputDataFileName, "r").read().split();
    for tempCur in range(0, len(tempInputFile), 2):
        inputDataTable.append([float(tempInputFile[tempCur]), float(tempInputFile[tempCur + 1])])
    return inputDataTable;

def inputTemperatedX():
    inputTemperatedX = float(input('Введите X: '))
    return inputTemperatedX;

def inputDataComparator():
    return inputTemperatedX(), uploadData("input.txt");

def outputResults(founded_root, exact_root):
    print("\nНайденное значение: ", founded_root)
    print("Точное значение:   ", exact_root)
    print("Относительная ошибка: %.2f" % abs(abs(exact_root - founded_root) / exact_root *
100), '%')
```

## *operations.py*

```
from math import sin
import config

# Задание функции.
def func(x):
    return sin(x)

# Заполнение массива шагов
def fillSteps(table, vectorLength):

    # Нулевой шаг равен нулю
    h_arr = [0]

    for tempCur in range(1, vectorLength):
        h_arr.append(table[tempCur][0] - table[tempCur - 1][0])
```

```

return h_arr;

def fillALAE(h_arr, table, vectorLength):
    A_arr = [0 if tempCur < 2 else h_arr[tempCur - 1] for tempCur in range(vectorLength)]
    B_arr = [0 if tempCur < 1 else -2 * (h_arr[tempCur - 1] + h_arr[tempCur]) for tempCur in
range(vectorLength)]
    D_arr = [0 if tempCur < 1 else h_arr[tempCur] for tempCur in range(vectorLength)]
    F_arr = [0 if tempCur < 2 else -3 * ((table[tempCur][1] - table[tempCur - 1][1]) / h_arr[tempCur] -
(table[tempCur - 1][1] - table[tempCur - 2][1]) / h_arr[tempCur - 1]) for tempCur in range(vectorLength)]

    return A_arr, B_arr, D_arr, F_arr;

# Прямой ход (Вычисление прогоночных коэффициентов).
def straightGetting(A_arr, B_arr, D_arr, F_arr, vectorLength):
    ks_arr = [0 for tempCur in range(vectorLength + 1)]
    nu_arr = [0 for tempCur in range(vectorLength + 1)]

    for tempCur in range(1, vectorLength):
        ks_arr[tempCur + 1] = D_arr[tempCur] / (B_arr[tempCur] - A_arr[tempCur] * ks_arr[tempCur])
        nu_arr[tempCur + 1] = (A_arr[tempCur] * nu_arr[tempCur] + F_arr[tempCur]) / (B_arr[tempCur] -
A_arr[tempCur] * ks_arr[tempCur])

    return ks_arr, nu_arr;

# Обратный ход (Нахождение всех Ci).
def reverseGetting(ks_arr, nu_arr, vectorLength):
    c_arr = [0 for tempCur in range(vectorLength + 1)]
    c_arr[vectorLength - 1] = nu_arr[vectorLength]

    for tempCur in range(vectorLength - 2, -1, -1):
        c_arr[tempCur] = ks_arr[tempCur + 1] * c_arr[tempCur + 1] + nu_arr[tempCur + 1]

    return c_arr

# Нахождение коэффициентов, выраженных через Ci.
def getAllCoffs(table, vectorLength, c_arr, h_arr):
    a_arr = [0 if tempCur < 1 else table[tempCur - 1][1] for tempCur in range(vectorLength)]
    b_arr = [0 if tempCur < 1 else (table[tempCur][1] - table[tempCur - 1][1]) / h_arr[tempCur] -
h_arr[tempCur] / 3 * (c_arr[tempCur + 1] + 2 * c_arr[tempCur]) for tempCur in range(vectorLength)]
    d_arr = [0 if tempCur < 1 else (c_arr[tempCur + 1] - c_arr[tempCur]) / (3 * h_arr[tempCur]) for
tempCur in range(vectorLength)]

    return a_arr, b_arr, d_arr;

# Нахождение коэффициентов, выраженных через Ci.
def getIndexOfInteval(table, vectorLength, x):
    ind = vectorLength - 1

    for tempCur in range (vectorLength - 1):
        if table[tempCur][0] <= x and x < table[tempCur + 1][0]:
            ind = tempCur + 1
            return ind;

    return ind;

# Получение значения интерполяции.
def getInterpolationResult(a_arr, b_arr, c_arr, x, table, ind, d_arr):
    return a_arr[ind] + b_arr[ind] * (x - table[ind - 1][0]) + c_arr[ind] * ((x - table[ind - 1][0]) ** 2)\
+ d_arr[ind] * ((x - table[ind - 1][0]) ** 3)

# Верификация таблицы
def verificateTable(table, x):

```

```

# Проверка на необходимое кол-во узлов.
if len(table) < 1:
    print('Недостаточное кол-во узлов для интерполяции.')
    return config.ERROR_STATUS;

# Исключение экстраполяции
if x > table[len(table) - 1][0] or x < table[0][0]:
    print('Экстраполяция недоступна.')
    return config.ERROR_STATUS;

return config.SUCCESS_STATUS;

# Функция интерполяции
def interpolation(table, x):
    # Стандартная сортировка таблицы
    table.sort()

    # Проверка таблицы на необходимое количество узлов и исключение интерполяции
    if (verificateTable(table, x) != config.SUCCESS_STATUS):
        exit(1)

    # Заполнение массива шагов
    h_arr = fillSteps(table, len(table));

    # Заполнение массивов коэффициентов СЛАУ.
    A_arr, B_arr, D_arr, F_arr = fillALAE(h_arr, table, len(table));

    # Прямой ход (Вычисление прогоночных коэффициентов).
    ks_arr, nu_arr = straightGetting(A_arr, B_arr, D_arr, F_arr, len(table));

    # Обратный ход (Нахождение всех Ci).
    c_arr = reverseGetting(ks_arr, nu_arr, len(table));

    # Нахождение коэффициентов, выраженных через Ci.
    a_arr, b_arr, d_arr = getAllCoffs(table, len(table), c_arr, h_arr);

    # Нахождение индекса интервала расположения введенного X.
    ind = getIndexOfInteval(table, len(table), x);

    # Получение значения интерполяции.
    interpolationResult = getInterpolationResult(a_arr, b_arr, c_arr, x, table, ind, d_arr);

    return interpolationResult

```

main.py

```

# Интерполяция кубическими сплайнами.
# Андреев Александр Алексеевич ИУ7-44Б.

import inputOutput, operations

# Коренная функция программы
def main():

    # Подгрузка данных и ввод с клавиатуры
    inputX, inputDataTable = inputOutput.inputDataComparator();

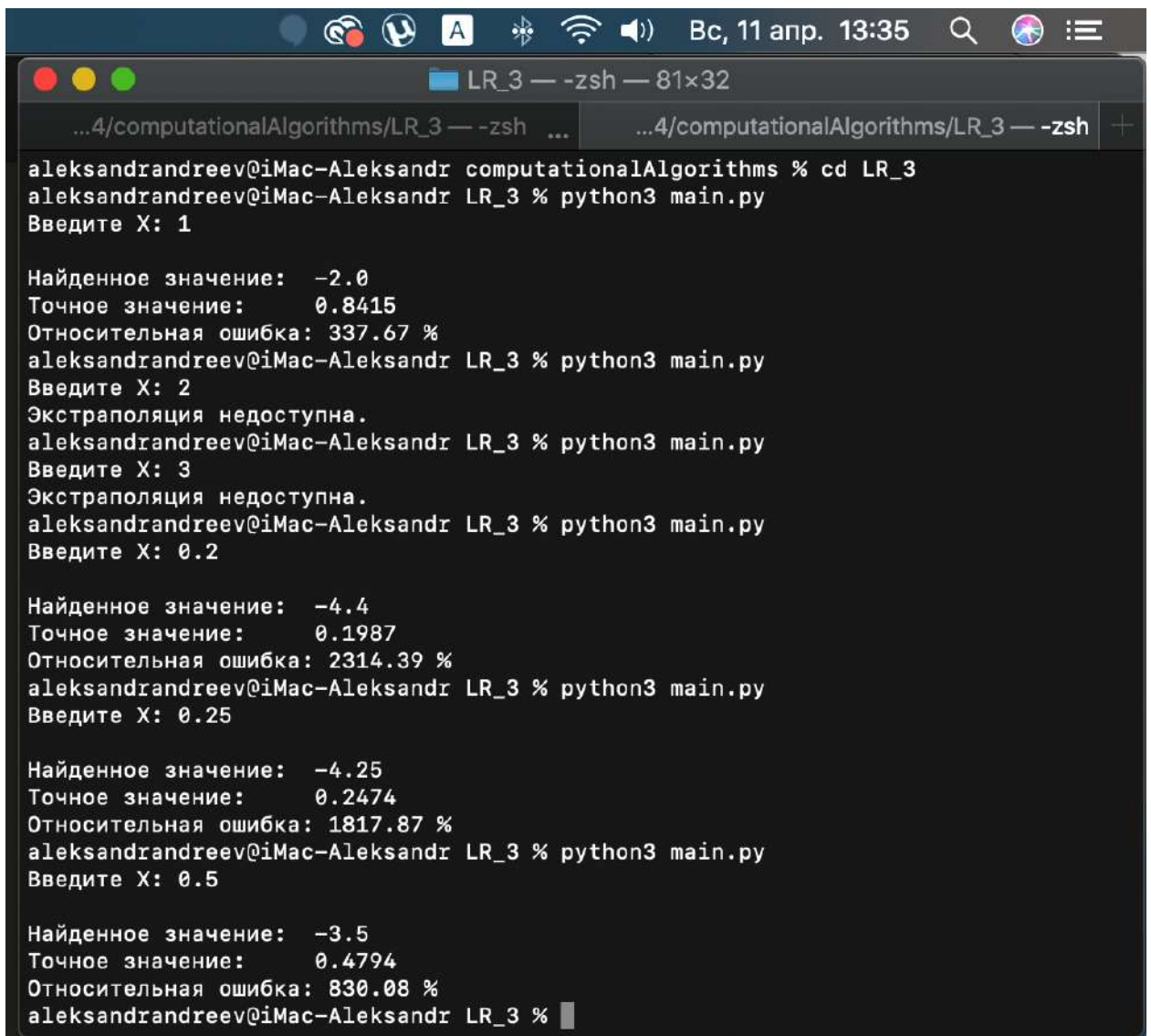
    # Выполнение алгоритмов.
    founded_root = float("%.4f" % (operations.interpolation(inputDataTable, inputX)))
    exact_root = float("%.4f" % operations.func(inputX))

```

```
# Вывод результата на экран
inputOutput.outputResults(founded_root, exact_root);

if __name__ == '__main__':
    main()
```

## Результат работы программы



```
...4/computationalAlgorithms/LR_3 — -zsh ...  ...4/computationalAlgorithms/LR_3 — -zsh +
aleksandrandreev@iMac-Aleksandr computationalAlgorithms % cd LR_3
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 1

Найденное значение:  -2.0
Точное значение:      0.8415
Относительная ошибка: 337.67 %
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 2
Экстраполяция недоступна.
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 3
Экстраполяция недоступна.
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 0.2

Найденное значение:  -4.4
Точное значение:      0.1987
Относительная ошибка: 2314.39 %
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 0.25

Найденное значение:  -4.25
Точное значение:      0.2474
Относительная ошибка: 1817.87 %
aleksandrandreev@iMac-Aleksandr LR_3 % python3 main.py
Введите X: 0.5

Найденное значение:  -3.5
Точное значение:      0.4794
Относительная ошибка: 830.08 %
aleksandrandreev@iMac-Aleksandr LR_3 %
```



# Контрольные вопросы:

1. Получить выражения для коэффициентов кубического сплайна, построенного на двух точках.

$\square y = x^2$  и  $\begin{cases} x_1 = 1 \\ x_2 = 2 \end{cases}$  Азреев А.А, ИУ7-445

$h = 1$  и  $S' = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3$

$a_1 = 4$  и  $c_0 = c_2 = 0$  и  $c_1 = 15$

$d_1 = c_1/h = 15$  и  $b_1 = 8$

2. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках

w2. Азреев А.А, ИУ7-445

Для однократной записи полинома 3-й степени 4 условия и про 3 точки - 8 усл.

1-й полином определен на 1-й и 2-й точках

2-й полином определен на 2-й и 3-й точках

$S''(a) = S''(b) = 0$  - усл. непрерывности

$S'(a) = S'(b) = 0$  - непрерывность

$S''(a) = S''(b)$

$S'(a) = S'(b)$  - непрерывность.

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо  $C_1 = C_2$ .

W3. Андрейв А.А., ИУ7-44Б

$$C_1 = C_2$$

$$C_1 = \xi_2 C_2 + \eta_2$$

$$\xi_2 C_2 + \eta_2 \Rightarrow \xi_2 \pm 1 \quad \eta_2 = 0$$

4. Написать формулу для определения последнего коэффициента сплайна  $C_N$ , чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано  $kC_{N-1} + mC_N = p$ , где  $k, m$  и  $p$  - заданные числа.

W4.

$$kC_{N-1} + mC_N = p$$

$$C_{N-1} = \xi_N C_N + \eta_N \Rightarrow k\xi_N C_N + k\eta_N + mC_N = p$$

$$\Rightarrow C_N = (p - k\eta_N) / (k\xi_N + m)$$