БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

Приближение функций

*Выполнил:*

Черепенников Роман

2 курс 8 группа

*Преподаватель:*

Радкевич Елена Владимировна

# Оглавление

# Постановка задачи

По входным данным

$$f(x) = 0.3 * \sin(x) + 0.4 * \cos(x)\,, \quad x \in [0,1]$$

$$x_1 = \frac{h}{2}, x_2 = 1 - \frac{h}{2}, x_3 = \frac{1}{2} + \frac{h}{2}, h = 0.1$$

1. Методом наименьших квадратов построить интерполяционный многочлен 5 степени и вычислить значение в точках $x_1, x_2, x_3$.
2. Построить интерполяционный многочлен Лагранжа 10 степени, вычислить значение в точках $x_1, x_2, x_3$, и оценить погрешность.
3. Построить интерполяционный многочлен Ньютона 10 степени и вычислить значение в точках $x_1, x_2, x_3$.
4. Выбрав узлы интерполирования наилучшим образом построить интерполяционный многочлен 10 степени, оценить погрешность

# Метод наименьших квадратов

Пусть Φ – пространство, порожденное элементами $\phi_0, \phi_1, \dots, \phi_n$. $\Phi_0$ – элемент наилучшей аппроксимации $f(x)$.

$$\Phi_0 = \sum_{i=0}^{n} c_i * \phi_i$$

Задачу построения $\Phi_0$ можно свести к задаче отыскания $c_i$, таких что $(f - \Phi_0, \phi_i) = 0$ .

В качестве $\phi_0, \phi_1, \dots, \phi_n$ возьмем систему многочленов $1, x, \dots, x^n$.

Тогда получим СЛАУ:

$$\begin{cases} c_0 * s_{00} + c_1 * s_{01} + \dots + c_n * s_{0n} = m_0 \\ \dots \\ c_0 * s_{n0} + c_1 * s_{n1} + \dots + c_n * s_{nn} = m_n \end{cases}$$

Где $s_{ij} = \int_a^b x^{i+j} dx, \quad m_i = \int_a^b x^i f(x) dx$

---

## Листинг программы

```
def f(x):
    return 0.3 * np.sin(x) + 0.4 * np.cos(x)


def s(i, j):
    return 1. / (i + j + 1)


def m(i):
    return integrate.quad(lambda x: pow(x, i) * (0.3 * np.sin(x) + 0.4 *
np.cos(x)), 0, 1)


def poly_val(coefs, x):
    i = 0
    res = 0.0
    for coef in coefs:
        res += coef * pow(x,i)
        i += 1

    return res


X = [0.05, 0.95, 0.55]
A = np.array([[s(i, j) for i in range(6)] for j in range(6)])
b = np.array([m(i)[0] for i in range(6)])
```

```
coefs = np.linalg.solve(A, b)
for x in X:
    print('P(', x, ') = ', poly_val(coefs, x), sep='')
    print('f(', x, ') = ', f(x), sep='')
    print('|P(', x, ') - f(',x,')| = ', abs( poly_val(coefs, x) - f(x)))
    print()
```

Вывод программы

P(0.05) = 0.41449367893721256

f(0.05) = 0.41449385493919005

|P( 0.05 ) - f( 0.05 )| = 1.7600197749212398e-07

P(0.95) = 0.47669770820457286

f(0.95) = 0.47669788722236556

|P( 0.95 ) - f( 0.95 )| = 1.7901779270079743e-07

P(0.55) = 0.4978157929605963

f(0.55) = 0.497815977503

|P( 0.55 ) - f( 0.55 )| = 1.8454240369170094e-07

## Интерполяционный многочлен Лагранжа

Интерполяционный многочлен Лагранжа имеет следующий вид:

$$P_n(x) = \sum_{i=0}^{n} l_i\,(x) f(x_i)$$

$$l_i = \frac{(x - x_0)(x - x_1) \dots (x - x_n)}{(x - x_i)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}$$

Обозначим через

$$w(x) = (x - x_0)(x - x_1) \dots (x - x_n)$$

$$w'(x_i) = (x - x_i)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)$$

Тогда многочлен Лагранжа можно записать следующим образом:

$$P_n(x) = \sum_{i=0}^{n} f(x_i) \frac{w(x)}{(x - x_i) w'(x_i)}$$

Погрешность интерполяции для многочлена Лагранжа по n+1 узлу (остаток интерполяции):

$$r_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w(x), x_0 < \xi < x_n$$

$|r_n(x)| \leq |\frac{M_{n+1}}{(n+1)!} w(x)|$, где $M_{n+1} = \max\limits_{x \in [x_0, x_n]} |f^{(n+1)}(x)|$

---

Вычислим $f^{(11)}(x)$:

$$f^{(1)}(x) = 0.3 * \cos(x) - 0.4 * \sin(x)$$

$$f^{(2)}(x) = -0.3 * \sin(x) - 0.4 * \cos(x)$$

$$f^{(3)}(x) = -0.3 * \cos(x) + 0.4 * \sin(x)$$

$$f^{(4)}(x) = 0.3 * \cos(x) + 0.4 * \sin(x)$$

Так как $f^{(4+k)} = f^{(k)}$, то

$$f^{(11)}(x) = -0.3 * \cos(x) + 0.4 * \sin(x)$$

$$M_{n+1} = \max_{x \in [0,1]} |-0.3 * \cos(x) + 0.4 * \sin(x)| = 0.3$$

## Листинг программы

```
def w(x, nodes):
    res = 1.0
    for node in nodes:
        res *= (x - node)
    return res


def lagrange_error(M, x, nodes):
    return np.abs(M * w(x, nodes) / np.math.factorial(nodes.size))


def lagrange_value(x, nodes):
    res = 0.0
    for i in range(nodes.size):
        p = 1.0
        for j in range(nodes.size):
            if i == j:
                continue
            p *= (x - nodes[j])
            p /= (nodes[i] - nodes[j])
        res += p * f(nodes[i])
    return res


interpolation_nodes = np.linspace(0.0, 1.0, num=11)

for x in X:
    print('P(', x, ') = ', lagrange_value(x,interpolation_nodes), sep='')
    print('f(', x, ') = ', f(x), sep='')
    print('r(', x, ') = ', lagrange_error(0.3,x,interpolation_nodes), sep='')
    print('|P(', x, ') - f(', x, ')| = ', abs(lagrange_value(x,interpolation_nodes) - f(x)))
    print()
```

## Вывод программы

```
P(0.05) = 0.4144938549391982

f(0.05) = 0.41449385493919005

r(0.05) = 2.402687072753908e-14

|P( 0.05 ) - f( 0.05 )| =  8.1601392309949e-15


P(0.95) = 0.4766978872223631

f(0.95) = 0.47669788722236556
```

r(0.95) = 2.402687072753901e-14

|P( 0.95 ) - f( 0.95 )| =  2.4424906541753444e-15

P(0.55) = 0.49781597750300005

f(0.55) = 0.497815977503

r(0.55) = 3.6048889160156323e-16

|P( 0.55 ) - f( 0.55 )| =  5.551115123125783e-17

## Интерполяционный многочлен Ньютона

Введем понятие разделенных разностей.

Разделенная разность нулевого порядка совпадает с $f(x_i)$. Разделенная разность $k$-го порядка определяется следующим образом:

$$f(x_i, \dots, x_j) = \frac{f(x_{i+1}, \dots x_j) - f(x_i, \dots, x_{j-1})}{x_j - x_i}, \qquad k = j - i$$

Интерполяционным многочленом Ньютона с разделенными разностями называется интерполяционный многочлен следующего вида:

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) + \dots + (x - x_0)\dots(x - x_{n-1})f(x_0, \dots, x_n)$$

---

Листинг программы

```
def build_table(nodes):  # построение таблицы разделенных разностей
    table = np.zeros(shape=(nodes.size, nodes.size))
    for i in range(nodes.size):
        table[i, 0] = f(nodes[i])
    for i in range(1, nodes.size):
        for j in range(nodes.size - i):
            table[j, i] = (table[j + 1, i - 1] - table[j, i - 1]) / (i * 0.1)
    return table


def newton_value(x, rr_table, nodes):
    n = rr_table.shape[0]
    p = 1.0
    res = rr_table[0, 0]
    for i in range(1, n):
        p *= (x - nodes[i - 1])
        res += p * rr_table[0, i]
    return res


rr_table = build_table(interpolation_nodes)

for x in X:
    print('P(', x, ') = ', newton_value(x,rr_table, interpolation_nodes), sep='')
    print('f(', x, ') = ', f(x), sep='')
    print('|P(', x, ') - f(', x, ')| = ', abs(newton_value(x,rr_table, interpolation_nodes) - f(x)))
    print()
```

Вывод программы

P(0.05) = 0.414493854939197

f(0.05) = 0.41449385493919005

|P( 0.05 ) - f( 0.05 )| =  6.938893903907228e-15

P(0.95) = 0.4766978872223611

f(0.95) = 0.47669788722236556

|P( 0.95 ) - f( 0.95 )| =  4.440892098500626e-15

P(0.55) = 0.4978159775029999

f(0.55) = 0.497815977503

|P( 0.55 ) - f( 0.55 )| =  1.1102230246251565e-16

# Минимизация остатка интерполирования

Многочлены Чебышева имеют следующий вид:

$$T_0(x) = 1, T_1(x) = x,$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n \geq 1$$

Если в качестве узлов интерполирования взять корни многочлена Чебышева, то остаток интерполирования будут минимальным.

Для $x \in [a, b]$ корни многочлена Чебышева можно найти следующим образом:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} * \cos\frac{\pi(2k+1)}{2n}$$

По полученным узлам можно построить интерполяционный многочлен, остаток которого будет оцениваться следующим образом:

$$|r_n(x)| \leq |\frac{M(b-a)^{n+1}}{(n+1)!\,2^{2n+1}}|$$

---

Листинг программы

```
def chebyshev_nodes(n, a, b):
    res = []
    for i in range(n):
        x = (a + b) / 2 + (b - a) / 2 * np.cos(np.pi * (2 * i + 1) / (2 * n))
        res.append(x)
    return np.array(res)



print('Узлы выбранные наилучшим образом: ')
best_nodes = chebyshev_nodes(11, 0, 1)
print(best_nodes)

for x in X:
    print('P(', x, ') = ', lagrange_value(x,best_nodes), sep='')
    print('f(', x, ') = ', f(x), sep='')
    print('r(', x, ') = ', lagrange_error(0.3,x,best_nodes), sep='')
    print('|P(', x, ') - f(', x, ')| = ', abs(lagrange_value(x,best_nodes) - f(x)))
    print()
```

Вывод программы

Узлы, выбранные наилучшим образом:

[0.99491072 0.954816   0.87787479 0.77032041 0.64086628 0.5
 0.35913372 0.22967959 0.12212521 0.045184   0.00508928]

P(0.05) = 0.41449385493918967

f(0.05) = 0.41449385493919005

r(0.05) = 8.828302688928786e-16

|P( 0.05 ) - f( 0.05 )| =  3.885780586188048e-16

P(0.95) = 0.47669788722236567

f(0.95) = 0.47669788722236556

r(0.95) = 8.828302688929056e-16

|P( 0.95 ) - f( 0.95 )| =  1.1102230246251565e-16

P(0.55) = 0.49781597750299933

f(0.55) = 0.497815977503

r(0.55) = 3.196837373396815e-15

|P( 0.55 ) - f( 0.55 )| =  6.661338147750939e-16

## Вывод

Наименьшую погрешность имеет интерполяционный многочлен Лагранжа, построенный по узлам, которые являются корнями многочлена Чебышева. Многочлены Ньютона и Лагранжа (без выбора узлов наилучшим образом) имеют приблизительно одинаковую погрешность. Точность метода наименьших квадратов была самой низкой, но стоит отметить что в этом методе строился многочлен 5ой степени, а не 10ой как в остальных случаях.