МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

По курсу «Численные методы»

Приближение функций

Вариант №9

Работу выполнил:
студент 3 курса 7 группы
Шатерник Артём
Преподаватель:
Будник А. М.

Минск 2024

1. **Постановка задачи.**

На отрезке $[0.55, 1.55]$ восстановить значения функции

$$f(x) = 0.55 * e^{-x} + (1 - 0.55) * cos(x)$$

В точках $x^* = [0.617, 1.1, 1.517]$ по $N = 11$ узлам:

x	0.55	0.65	0.75	0.85	0.95	1.05	1.15	1.25	1.35	1.45	1.55
$f(x)$	0.701	0.6454	0.5891	0.5321	0.4745	0.4164	0.358	0.2995	0.2411	0.1832	0.1261

Используя метод наименьших квадратов, многочлен Лагранжа, многочлен Ньютона, многочлен Чебышева и интерполирование в конце таблицы.

2. **Алгоритм решения.**

a) Метод наименьших квадратов.

Будем строить полином степени $n = 5$:

$$P(x) = \sum_{i=0}^{n} c_i x^i.$$

При табличном задании функции коэффициенты c_i находим из системы уравнений:

$$\sum_{i=0}^{n} (\sum_{j=0}^{N} p(x_j)\, x_j^{i+k})\, c_i = \sum_{j=0}^{N} p(x_j)\, f(x_j) x_j^k, \qquad k = 0, \dots, n,$$

$$p(x) = 1.$$

Решать полученную систему будем обычным методом Гаусса.

Оценка погрешности:

Истинное значение погрешности будем считать в этом и дальнейших методах по формуле:

$$r(x^*) = |f - P|,$$

где f — точное значение в точке, P — приближённое значение.

Теоретическую погрешность для МНК найдём по формуле:

$$\Delta(f) = \sqrt{\sum_{j=0}^{n} \left(f(x_j) - P(x_j)\right)^2}$$

b) Многочлен Лагранжа.

Строим по формуле:

$$P(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=1, i \neq j}^{n} \left(\frac{x - x_j}{x_i - x_j}\right)$$

Оценка погрешности по формуле:

$$|r_n(x)| \le \frac{|w_{n+1}(x)|}{(n+1)!} M,$$

где

$$M = \max\left|f^{(n+1)}(x)\right|,$$

и

$$w_{n+1}(x) = (x - x_0) \dots (x - x_n).$$

c) Многочлен Ньютона.

Строим по формуле:

$$P_n(x) = f(x_0) + (x - x_0) * f(x_0, x_1) + (x - x_0) * (x - x_1) * f(x_0, x_1, x_2) + \cdots + (x - x_0) * \dots * (x - x_{n-1}) * f(x_0, \dots, x_n),$$

где

$$f\left(x_{0,} \dots, x_k\right) - \text{разделённые разности } k - \text{го порядка.}$$

Оценка погрешности аналогична оценке погрешности для метода Лагранжа, так как оба этих метода строят один и тот же многочлен.

Также оценку можно провести по формуле:

$$r_n(x) = w_{n+1}(x) * f(x, x_0, \dots, x_n)$$

d) Многочлены Чебышева.

Для минимизации остатка интерполирования воспользуемся чебышёвской сеткой узлов, для перехода к которой используем формулу:

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

Далее строим многочлен Ньютона на новой сетке.

Оценки погрешности:

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

$$|r_n^2(x)| \le \frac{|w_{n+1}(x)|}{(n+1)!} M.$$

e) Интерполирование в конце таблицы.

Для интерполирования в конце таблицы будем строить многочлен $k = 3$ степени, для чего будем использовать 4 последних узла:

1.25	1.35	1.45	1.55
0.2995	0.2411	0.1832	0.1261

Находить приближённое значение будем только для точки $x^* = 1.517$, так как только она лежит между этими узлами.

Сделаем замену вида:

$$t = \frac{x - x_n}{h},$$

учитывая, что $h = 0.1$ и $x_n = 1.55$.

Далее будем строить многочлен по формуле:

$$P_k(t) = f_n + \frac{t}{1!}\Delta f_{n-1} + \frac{t(t+1)}{2!}\Delta f_{n-2} + \dots + \frac{t(t+1)\dots(t+k-1)}{k!}\Delta f_{n-k}.$$

Оценка погрешности:

$$r_k(t) \le h^{k+1}\frac{t(t+1)\dots(t+k)}{(k+1)!}M_k,$$

где

$$M_k = \max\left|f^{(k+1)}(x)\right| \text{ на } [x_{n-k}, x_n]$$

3. Листинг программы.

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

# Метод Гаусса
def gaussFunc(matrix):
    copy_matrix = np.copy(matrix)
    for nrow, row in enumerate(copy_matrix):
        divider = row[nrow]
        row /= divider
        for lower_row in copy_matrix[nrow+1:]:
            factor = lower_row[nrow]
            lower_row -= factor * row
    return copy_matrix

def gauss_reverse(matrix):
    n_row=matrix.shape[0]
    x= [None] * n_row
    for i in range(n_row-1, -1,-1):
        x[i]=matrix[i,-1]-np.dot(matrix[i, i+1:n_row], x[i+1:])
    return np.array(x)

def gauss_method(matrix):
    return gauss_reverse(gaussFunc(matrix))

# Отрезок
my_N = 9
a = 0.1 + 0.05 * my_N
b = 1 + a
print(f"a: {a} \nb: {b}")
# Узлы
n = 10
h = 1 / 10
nodes = [round(a + i * h, 3) for i in range(0, n + 1)]
print("Узлы:", nodes)
# Значения в узлах
f_nodes = [a * math.e**(-x) + (1 - a) * math.cos(x) for x in nodes]
print("Значения в узлах:\n", f_nodes)
for x in f_nodes:
    print(round(x, 4), end=', ')
# Точки для восстановления
x_find = [nodes[0] + 2 * h / 3, nodes[int(n / 2)] + 0.5 * h, nodes[n] - h / 3]
print("Точки для восстановления:\n", x_find)
```

```
# Метод наименьших квадратов (МНК)
m = int(n / 2)
print("Степень полинома:", m)

def MNK(nodes, funcs, m):
    n = len(nodes)
    matrix = np.zeros((m + 1, m + 2))
    for k in range(0, m + 1):
        for i in range(0, m + 1):
            for j in range(0, n):
                matrix[k][i] += nodes[j]**(i + k)
        for i in range(0, n):
            matrix[k][m + 1] += funcs[i] * nodes[i]**k
    return gauss_method(matrix)


coeff = MNK(nodes, f_nodes, m)
print("Коэффициенты многочлена:\n", coeff)


def func(x, a):
    return a * math.e**(-x) + (1 - a) * math.cos(x)

def polynom(x, coeffs):
    res = 0
    for i in range(len(coeffs)):
        res += x**i * coeffs[i]
    return res


# Значения в точках восстановления
y_find_MNK = [polynom(i, coeff) for i in x_find]
print("Восстановленные значения:", y_find_MNK)

# Истинная погрешность
r_MNK = [abs(func(i, a) - polynom(i, coeff)) for i in x_find]
print("Истинная погрешность:", r_MNK)

# Теоретическая погрешность
delta_f_MNK = 0
for i in range(len(nodes)):
    delta_f_MNK += (f_nodes[i] - polynom(nodes[i], coeff))**2
delta_f_MNK = math.sqrt(delta_f_MNK)
print("Погрешность:", delta_f_MNK)

# Многочлен Лагранжа
def lagrange_interp(nodes, funcs, find_x):
    results = []
    n = len(nodes)
    for x in find_x:
        pol = 0
        for i in range(n):
            base = 1
            for j in range(n):
                if i != j:
                    base *= (x - nodes[j]) / (nodes[i] - nodes[j])
            pol += base * funcs[i]
        results.append(pol)
    return results

y_find_lagrange = lagrange_interp(nodes, f_nodes, x_find)
print("Восстановленные значения:", y_find_lagrange)
```

```
# Истинная погрешность
y_real = [func(x, a) for x in x_find]
r_lagrange = [abs(y_real[i] - y_find_lagrange[i]) for i in range(len(x_find))]
print("Истинная погрешность:", r_lagrange)
# Теоретическая погрешность
# Производная 11 порядка
def func_der(x, a):
    return abs(- a * math.e**(-x) + (1 - a) * math.sin(x))

# Находим максимум модуля функции
x = np.arange(a, b + 0.001, 0.01)
y = [func_der(i, a) for i in x]
M = max(y)
plt.plot(x, y, color="c")
plt.plot(b, M, "go", color="m")
plt.axhline(func_der(b, a), linestyle="--", alpha=0.5)
plt.axvline(b, linestyle="--", alpha=0.5)
plt.xticks([b] + list(np.arange(a, 1.0, 0.25)))
plt.yticks([M] + list(np.arange(0.1, 0.80, 0.25)))
plt.grid(axis="y")
print("M:", M)

w = [1 for i in range(len(x_find))]
for i in range(len(x_find)):
    for j in range(n + 1):
        w[i] *= (x_find[i] - nodes[j])
print(w)

# Погрешности для точек
delta_r_lagrange = []
fact = math.factorial(n + 1)
for i in range(len(w)):
    delta_r_lagrange.append(abs(M * w[i]) / fact)
print("Оценки погрешностей для точек восстановления:", delta_r_lagrange)
for i in range(len(w)):
    print(f"r_{i} <= {delta_r_lagrange[i]}")

# Метод Ньютона
def get_sep_diff_table(nodes, funcs):
    n = len(nodes)
    sep_diff_table = [
        nodes,
        funcs
    ]
    # Столбцы
    for i in range(1, n):
        column = []
        # Строки
        for j in range(n - i):
            column.append((sep_diff_table[i][j + 1] - sep_diff_table[i][j]) /
(sep_diff_table[0][j + i] - sep_diff_table[0][j]))
        sep_diff_table.append(column)
    return sep_diff_table

def newton_interp(nodes, funcs, find_x):
    n = len(nodes)
    sep_diff_table = get_sep_diff_table(nodes, funcs)
    results = []
    for x in find_x:
        find_y = funcs[0]
        prev = 1
        for i in range(1, n):
            prev *= (x - nodes[i - 1])
            find_y += prev * sep_diff_table[i + 1][0]
```

```
        results.append(find_y)
    return results

y_find_newton = newton_interp(nodes, f_nodes, x_find)
print("Восстановленные значения:", y_find_newton)

# Истинная погрешность
r_newton = [abs(y_real[i] - y_find_newton[i]) for i in range(len(x_find))]
print("Истинная погрешность:", r_newton)

# Теоретическая погрешность как у метода Лагранжа
delta_r_newton = delta_r_lagrange
for i in range(len(w)):
    print(f"r_{i} <= {delta_r_newton[i]}")


# Многочлен Чебышева
# Пересчитываем узлы
nodes_cheb = []
for i in range(n + 1):
    nodes_cheb.append((a + b) / 2 + math.cos((2 * i + 1) * math.pi / (2 * (n + 1)))
* (b - a) / 2)
print("Новые узлы:", nodes_cheb)
# Значения в узлах
f_nodes_cheb = []
for i in range(n + 1):
    f_nodes_cheb.append(func(nodes_cheb[i], a))
print("Новые значения в узлах:", f_nodes_cheb)

# Воспользуемся многочленом Ньютона по новой сетке узлов
y_find_cheb = newton_interp(nodes_cheb, f_nodes_cheb, x_find)
print("Восстановленные значения:", y_find_cheb)

# Истинная погрешность
r_cheb = [abs(y_real[i] - y_find_cheb[i]) for i in range(len(x_find))]
print("Истинная погрешность:", r_cheb)
# Теоретическая погрешность общая
delta_r_cheb_gen = (M * pow(b - a, n + 1)) / (math.factorial(n + 1) * pow(2, 2 * n
+ 1))
print("Оценка для погрешностей:", delta_r_cheb_gen)

# Теоретическая погрешность для каждой точки
# Ищем w на чебышевской сетке
w_cheb = [1 for i in range(len(x_find))]
for i in range(len(x_find)):
    for j in range(n + 1):
        w_cheb[i] *= (x_find[i] - nodes_cheb[j])
print("w:", w_cheb)
# Погрешности
delta_r_cheb = []
fact = math.factorial(n + 1)
for i in range(len(w)):
    delta_r_cheb.append(abs(M * w_cheb[i]) / fact)
print("Оценки погрешностей для точек восстановления:", delta_r_cheb)
for i in range(len(w)):
    print(f"r_{i} <= {delta_r_cheb[i]}")

# Интерполяция в конце таблицы
# Берём 4 узла
m = 4
nodes_end = nodes[len(nodes) - m:]
f_nodes_end = f_nodes[len(f_nodes) - m:]
print("Узлы:", nodes_end)
print("Значения в них:", f_nodes_end)
```

```
# Таблица конечных разностей
def get_finite_diff_table(nodes, funcs):
    n = len(nodes)
    finite_diff_table = [
        nodes,
        funcs
    ]
    # Столбцы
    for i in range(1, n):
        column = []
        # Строки
        for j in range(n - i):
            column.append((finite_diff_table[i][j + 1] - finite_diff_table[i][j]))
        finite_diff_table.append(column)
    return finite_diff_table

finite_table = get_finite_diff_table(nodes_end, f_nodes_end)
print("Таблица конечных разностей\n", finite_table)
# Замена на t
t_find = []
for node in x_find:
    t_find.append((node - nodes_end[::-1][0]) / h)
print(t_find)

# Интерполирование
def table_end_interp(nodes, funcs, find_x):
    n = len(nodes)
    fin_tab = get_finite_diff_table(nodes, funcs)
    results = []
    for iter, x in enumerate(find_x):
        find_y = funcs[n - 1]
        prev = 1
        fact = 1
        for i in range(1, n):
            prev *= x + i - 1
            find_y += prev * fin_tab[i + 1][n - i - 1] / fact
            fact *= i + 1
        results.append(find_y)
    return results

y_find_end = table_end_interp(nodes_end, f_nodes_end, t_find)
print("В конце таблицы:", y_find_end)
# Истинная погрешность
r_end = [abs(y_real[i] - y_find_end[i]) for i in range(len(x_find))]
print("Истинная погрешность:", r_end)
# Теоретическая погрешность
# Производная 4 порядка совпадет с самой функцией
a, b = 1.25, 1.55
x = np.arange(a, b + 0.001, 0.01)
y = [func(i, 0.55) for i in x]
M = max(y)
k = m - 1
w_end = []
for t in t_find:
    w_temp = t
    for i in range(k):
        w_temp *= t + i + 1
    w_end.append(w_temp)
print(w_end)

delta_r_end = []
for i in range(len(t_find)):
```

```
    delta_r_end.append(pow(h, k + 1) * w_end[i] * M / math.factorial(k + 1))

print(delta_r_end)
```

4. Результат и его анализ.

a) Метод наименьших квадратов.

Результат.

Получаем следующие коэффициенты многочлена:

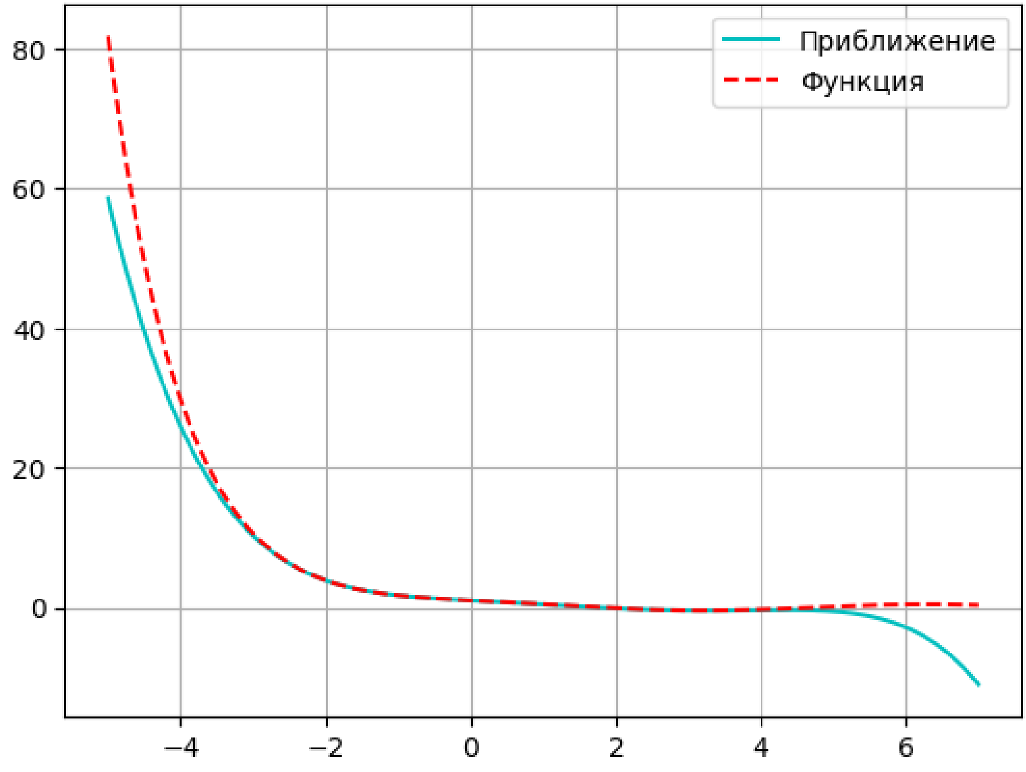
1.00005981	-0.55040221	0.05109348	-0.09318502	0.04274368	-0.00484003

Восстановленные значения:
[0.6639721063254683, 0.38719733942043694, 0.1450396192228639].
Истинная погрешность:
[3.329822140241134e-08, 1.1255016585387523e-08, 1.3148866689904892e-08].
Теоретическая погрешность: 4.5770075957360186e-08.

Анализ.

Истинные значения погрешности меньше её теоретической оценки, как и должно быть. Получили погрешности 10^{-8} так как строили полином всего 5 степени, точность можно повысить, увеличив степень полинома.

Можно сравнить графики приближённой и реальной функции:



По графику видно, что по мере отдаления от отрезка [a, b] точность начинает заметно падать, в окрестности же [a, b] графики практически неотличимы.

b) Многочлен Лагранжа.

Результат.

Восстановленные значения:
[0.6639721396236813, 0.38719735067545374, 0.14503963237175216].
Истинная погрешность:

[8.43769498715119e-15, 2.220446049250313e-16, 2.1566082253343666e-14].
$M = \max\left|f^{(11)}(x)\right| = \ 0.333166308280502.$

Теоретические погрешности в точках восстановления:
[1.6058213967572225e-14, 4.0034251063674126e-16, 3.4226332364342484e-14].
Анализ.
Истинные значения погрешности меньше её теоретической оценки, для первой точки на порядок ниже. Сравнение с МНК не является корректным, так в данном случае строится полином 10 степени.
c) Многочлен Ньютона.
Результат.
Восстановленные значения:
[0.6639721396236812, 0.38719735067545374, 0.14503963237175233].
Истинная погрешность:
[8.548717289613705e-15, 2.220446049250313e-16, 2.173261570703744e-14].
Теоретическая погрешность та же, что у многочлена Лагранжа.
Анализ.
Опять видно, что истинные значения погрешности ниже, чем теоретические оценки. Если сравнить с многочленом Лагранжа можно заметить, что получили погрешности того же порядка, что неудивительно, так как фактически мы строим один и тот же многочлен разными способами. Также можно заметить, что в средней точке их значения полностью совпадают, на боковых точках многочлен Ньютона дал слегка худший результат, возможно это связанно с накоплением погрешностей при построении таблицы разделённых разностей.
d) Многочлен Чебышева.
Результат.
Новые узлы:

1.5449	1.5049	1.4279	1.3203	1.1909	1.050	0.9091	0.7797	0.6721	0.5952	0.5550

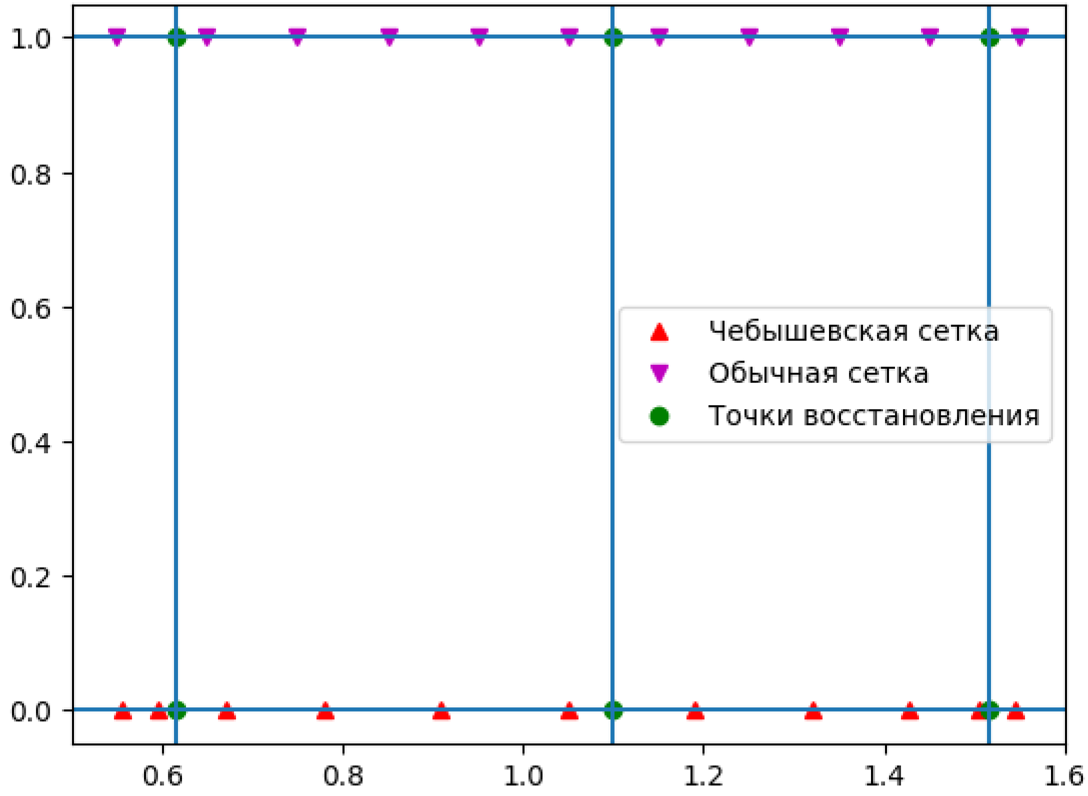
Восстановленные значения (методом Ньютона):
[0.6639721396236916, 0.38719735067545563, 0.14503963237173215].
Истинная погрешность:
[1.887379141862766e-15, 2.1094237467877974e-15, 1.5543122344752192e-15].
Теоретическая погрешность:
Общая оценка.
$|r_n^1(x)| \leq 3.979930145941079e-15.$
Оценка для каждой точки $|r_n^2(x)|$:
[3.418272532648346e-15, 3.5502616862258365e-15, 2.4811419281768608e-15].
Анализ.
Оценки в точках получились того же порядка, как и общая оценка. Сами значения этих оценок получились более точными.
Истинные значения погрешностей ниже, чем их оценки.
Если сравнивать с многочленом Ньютона можно заметить следующее:

- на границах точность повысилась

В левой точке порядок не изменился, в правой же точность лучше на порядок

- в средней точке точность понизилась на один порядок

Если схематически изобразить две стеки узлов, то получим следующее:



Можно заметить, что сетка перестала быть равномерной и средняя точка сдвинулась к одному из узлов, а количество узлов в её окрестности не увеличилось, это может быть причиной ухудшения точности в ней.

e) Интерполяция в конце таблицы.

Результат.

Рассматриваем только последнюю точку восстановления.

После замены на t она станет равна: -0.3333.

Восстановленное значение:

[0.14504045000348767].

Истинная погрешность:

[8.176317570773861e-07]

$M = \max\left|f^{(4)}(x)\right| = M$: 0.2994727013509755

Теоретическая погрешность:

[-2.884602595210065e-06].

Анализ.

Истинная погрешность ниже, чем теоретическая, как и должно быть. Получили точность порядка 10^{-7}, так как строили полином всего 3 степени, так что сравнение с другими методами не совсем корректно. Точность можно повысить, если взять на том же отрезке более мелкое разбиение и строить по нему полином более высокой степени.