БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра вычислительной математики

Лабораторная работа №1
Приближение функций
Вариант № 14

Выполнил:
Кражевский Алексей Игоревич,
3 курс 13 группа (ТП)
Преподаватель:
Будник Анатолий Михайлович

Минск, 2022

Содержание

Постановка задачи

Рассмотрим на отрезке [a, b] набор различных точек: $x_0, x_1, \dots, x_n$. В этих точках заданы значения функции $f(x)$, $f_i = f(x_i)$, $i = \overline{0, n}$.

Дано:

- $[a, b] = [\alpha_j, 1 + \alpha_j]$;
- $\alpha_j = 0{,}1 + 0{,}05\, j$, j — порядковый номер по списку;
- $x_i = \alpha_j + ih$, $h = \frac{1}{n}$, $i = \overline{0,n}$, $n = 10$;
- функция: $f(x) = \alpha_j \cdot e^x + (1 - \alpha_j) \cdot \sin(x)$;
- точки восстановления:
  1. $x_1 = x_o + \frac{2}{3} h$
  2. $x_2 = \frac{1}{2} \cdot h + x_{\frac{n}{2}}$
  3. $x_3 = x_n - \frac{1}{3} \cdot h$

Задача: Построить интерполяционный многочлен Ньютона. Найти значения функции в точках восстановления x^*, x^{**}, x^{***} и оценить погрешность интерполяции в форме Лагранжа, оценить истинную погрешность.

Алгоритм решения

Формула интерполяционного многочлена Ньютона:

$$P_n(x)=f(x_0)+(x-x_0)\cdot f(x_0;x_1)+(x-x_0)\cdot(x-x_1)f(x_0;x_1;x_2)+\ldots+(x-x_0)\cdot\ldots\cdot(x-x_{n-1})\cdot f(x_0;x_1;\ldots;x_n)$$

Формула для подсчета разделенных разностей:

$$f(x_1;\ldots;x_{k+1})=\frac{f(x_2;\ldots;x_{k+1})-f(x_1;\ldots;x_k)}{x_{k+1}-x_1} \quad ,$$

В частности, для двух точек форула принимает вид:

$$f(x_0;x_1)=\frac{f(x_1)-f(x_0)}{(x_1-x_0)}$$

и для трех точек:

$$f(x_0;x_1;x_2)=\frac{f(x_0;x_1)-f(x_0;x_2)}{x_2-x_1}$$

Для интерполирования многочленом Ньютона, нужно построить таблицу разделенных разностей вида:

x_0	$f(x_0)$	$f(x_0;x_1)$	$f(x_0;x_1;x_2)$	$\cdots$	$f(x_0;x_1;\ldots;x_n)$
x_1	$f(x_1)$	$f(x_1;x_2)$	$f(x_1;x_2;x_3)$	$\cdots$	
x_2	$f(x_2)$	$\cdots$	$\cdots$	$\cdots$	
$\cdots$	$\cdots$	$\cdots$	$\cdots$		
x_n	$f(x_n)$				

При выборе метода интерполирования, стоит учитывать, что из-за построения таблицы для метода Ньютона, его лучше использовать при изменении количества узлов (т.е. проще посчитать новые значения). А при изменении функции на тех же точках, стоит использовать интерполяционный многочлен Лагранжа.

Остаток интерполирования в форме Лагранжа:

$$r_n(x)=\frac{\omega(x)\cdot f^{(n+1)}(\xi)}{(n+1)!}$$

Код решения

Константы по условию:

```
# constants
alpha = 0.1 + 0.05 * 14
n_ = 10
h = 1 / n_
```

Функция для подсчета иксов:

```
def x_i(indx: int) -> float:
    return alpha + (indx * h)
```

Функция из условия (для интерполирования):

```
def func(x: float) -> float:
    """
    Function to interpolate.
    """
    return (alpha * np.power(np.e, x)) + (1 - alpha) * np.sin(x)
```

Производная:

```
def der_11_func(x: float) -> float:
    return alpha * np.power(np.e, x) + (alpha - 1) * np.cos(x)
```

Создание и заполнение таблицы разделенных разностей:

```
def divided_diff(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    """
    function to calculate the divided differences table.
    """
    n = len(y)
    coef = np.zeros([n, n])
    # the first column is y
    coef[:, 0] = y

    for j in range(1, n):
        for i in range(n - j):
            coef[i][j] = (coef[i + 1][j - 1] - coef[i][j - 1]) / (x[i + j] - x[i])

    return coef
```

Подсчет полинома Ньютона:

```
def newton_poly(coef: np.ndarray, x_data: np.ndarray, x: np.ndarray) -> np.ndarray:
    """
    Evaluate the newton polynomial at x.
    """
```

```
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1, n + 1):
        p = coef[n - k] + (x - x_data[n - k]) * p
    return p
```

Функция, подсчитывающая значение $\omega_{n+1}(x)$:

```
def w(x: float, x_arr: np.ndarray) -> float:
    ans = 1
    for i in range(n_ + 1):
        ans *= x - x_arr[i]
    return ans
```

Подсчет погрешности интерполирования в форме Лагранжа:

```
def lagrange_diff(x: float, x_arr: np.ndarray) -> float:
    """
    Function to count difference in Lagrange's form
    """
    xi = 1.3
    return np.abs(w(x, x_arr) / np.math.factorial(n_ + 1) * der_11_func(xi))
```

Главная функция:

```
def interpolate():
    """
    Main method to interpolate the function.
    """
    import matplotlib.pyplot as plt

    first_x = x_i(0) + 2/3 * h
    second_x = x_i(int(n_ / 2)) + 1 / 2 * h
    third_x = x_i(n_) - 1 / 3 * h
    x1 = np.array([first_x, second_x, third_x])
    y1 = np.array([func(first_x), func(second_x), func(third_x)])

    points = np.zeros(11)
    values = np.zeros(11)
    for i in range(0, 11):
        points[i] = x_i(i)
        values[i] = func(points[i])

    a_s = divided_diff(points, values)[0, :]
    inter = newton_poly(a_s, points, x1)

    print(f'Interpolating from {alpha} to {1 + alpha}')

    print(f'Points to interpolate: {points}', end='\n')
    print(f'Values of points: {values}', end='\n')
    print()

    print(f'Interpolated in recovery points: {inter}')
    print(f'Real values in recovery points: {y1}')
```

```
print()

print(f'Real difference: {[y1[i] - inter[i] for i in range(len(y1))]}')
print(f'Lagrange difference in x*: {lagrange_diff(first_x, points)}')
print(f'Lagrange difference in x**: {lagrange_diff(second_x, points)}')
print(f'Lagrange difference in x***: {lagrange_diff(third_x, points)}')

plt.plot(points, values, 'g')
plt.plot(x1, inter, 'r+')
plt.show()
```

Результат выполнения программы

Отрезок интерполирования согласно варианта:

```
Interpolating from 0.8 to 1.8
```

Точки (x, y) для интерполирования:

```
Points to interpolate: [0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8]
Values of points: [1.92390396 2.12434787 2.34291966 2.58157429 2.84250136 3.12814897
 3.44124992 3.78485025 4.16234066 4.57749088 5.0344875 ]
```

Значения в точках восстановления после интерполирования:

```
Interpolated in recovery points: [2.05560924 3.2810851  4.87722887]
```

Реальные значения в точках восстановления:

```
Real values in recovery points: [2.05560924 3.2810851  4.87722887]
```

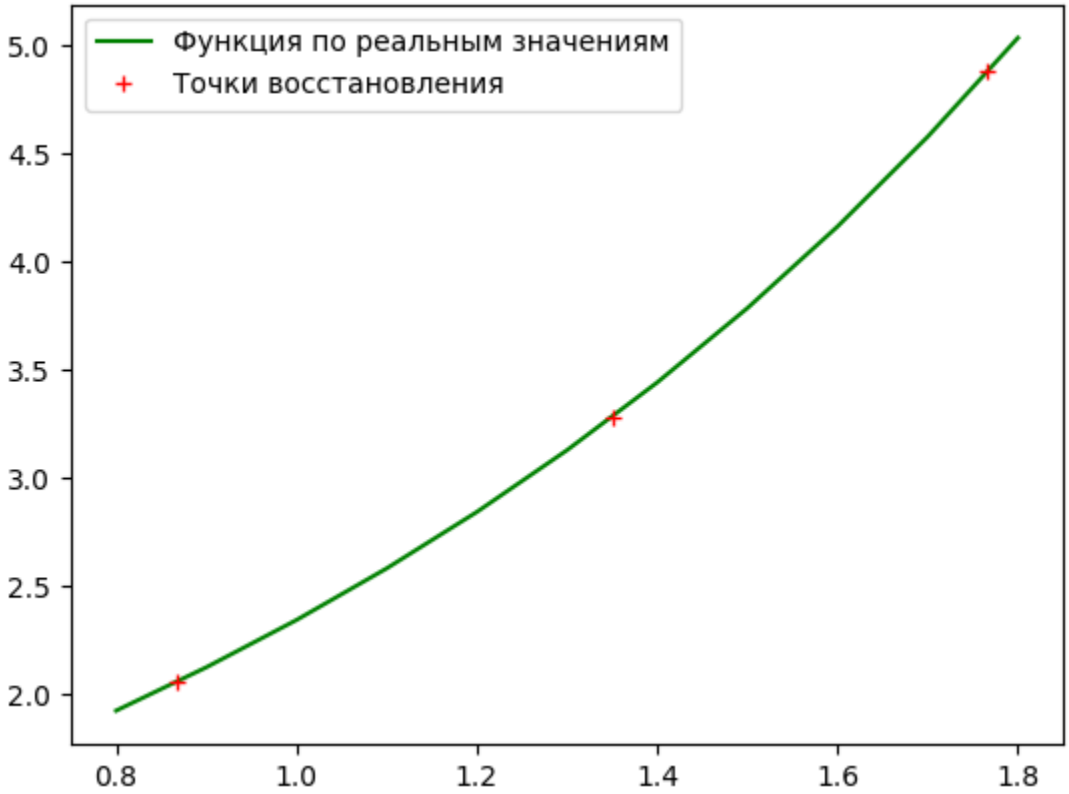
Реальная погрешность интерполирования:

```
Real difference: [1.3455903058456897e-13, -3.1086244689504383e-15, -3.1175062531474396e-13]
```

Погрешность интерполирования по Лагранжу (берем ξ такое, что производная будет максимальной (для оценки погрешности сверху)):

```
Lagrange difference in x*: 1.72127713915908e-13
Lagrange difference in x**: 4.291264350967897e-15
Lagrange difference in x***: 3.668714564083596e-13
```

График функции:



Выводы

В данной лабораторной работе мы построили интерполяционный многочлен Ньютона с помощью таблицы разделенных разностей, оценили погрешность в форме Лагранжа и истинную погрешность.

При восстановлении значений в точках x^*, x^{**}, x^{***} мы получили значения с точностью до 10^{-13}. Погрешность в форме Лагранжа получилась больше истинной погрешности.

Погрешность в разных точках разная, т.к. погрешность в точке зависит от удаления от узла интерполирования. В самой точке погрешность нулевая, и при удалении от точки погрешность возрастает.

Теоретическая и истинная погрешность отличаются, так как для теоретической оценки мы берем максимальные значения (оцениваем сверху), поэтому истинная погрешность получается меньше.

Часть 2. Интерполяция на равномерной сетке узлов

Постановка задачи

Построить интерполяционный многочлен Ньютона третьей степени на равномерной сетке узлов для конца таблицы. Найти приближенное значение в точке восстановления x^{***} и оценить погрешность.

Дано:

1. $f(x_i)=f_i \quad , \quad [a,b]=[\alpha_j, 1+\alpha_j]$
2. $\alpha_j=0{,}1+0{,}05\cdot j$, j - номер в списке (14)
3. $x_i=\alpha_j+i\cdot h, i=\overline{0,n}, h=\frac{1}{n}, n=10$
4. $f(x)=\alpha_j\cdot e^x+(1-\alpha_j)\cdot\sin(x)$
5. $x^{***}=x_n-\frac{1}{3}\cdot h$

Алгоритм решения

Конечная разность:

$$\Delta y(x)=y(x+h)-y(x),$$
$$\Delta^2 y=\Delta(\Delta y)=\Delta(y(x+h)-y(x))=y(x+2\cdot h)-2\cdot y(x+h)+y(x)$$

Таблица конечных разностей:

$$\begin{array}{ccccc} f_0 & \Delta f_0 & \Delta^2 f_o & \cdots & \Delta^n f_0 \\ f_1 & \Delta f_1 & \cdots & & \\ \vdots & \vdots & \vdots & & \\ f_{n-1} & \Delta f_{n-1} & & & \\ f_n & & & & \end{array}$$

Формула для конца таблицы:

$$x=x_n+t\cdot h, t\in[-1,0]$$
$$P_k(x_n+t\cdot h)=f_n+t\cdot\Delta f_{n-1}+\frac{t\cdot(t+1)}{2!}\Delta^2 f_{n-2}+\ldots+\frac{t\cdot(t+1)\cdot\ldots\cdot(t+k-1)}{k!}\Delta^n f_{n-k}, x\in[x_{n-1},x_n]$$

Погрешность:

$$r_k(x)=\frac{h^{k+1}\cdot t\cdot(t+1)\cdot\ldots\cdot(t+k)}{(k+1)!}\cdot f^{(k+1)}(\xi), \xi\in[x_n-k\cdot h, x_n]$$

Код решения

Построение конца таблицы:

```
def diff_table(n, values):
    table = np.zeros([n, n])
    table[:, 0] = values[6:]
    for i in range(1, n):
        for j in range(0, n - i):
            table[j, i] = table[j + 1, i - 1] - table[j, i - 1]
    return table
```

Формула Ньютона для конца таблицы:

```
def numerator(t, n):
    ans = 1
    for i in range(n):
        ans *= t + i
    return ans


def newton_table_end(x, n, h_, x_n, table):
    t = np.abs(x_n - x) / h_
    ans = 0
    k_ = 0
    for i in range(n - 1, -1, -1):
        ans += (numerator(t, k_) * table[i, k_]) / np.math.factorial(k_)
        k_ += 1
    return ans
```

Подсчет погрешности:

```
def r_k(n, h_, t, a, b):
    return abs((np.power(h_, n+1) * numerator(t, n+1) * max_der_(a, b, 10)) / np.math.factorial(n+1))
```

Результат выполнения

Мы получили следующую таблицу конечных разностей:

```
Table: [[3.44124992e+00 3.43600334e-01 3.38900724e-02 3.76973640e-03]
 [3.78485025e+00 3.77490407e-01 3.76598088e-02 0.00000000e+00]
 [4.16234066e+00 4.15150215e-01 0.00000000e+00 0.00000000e+00]
 [4.57749088e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

После интерполяции в точке x^{***} мы получили следующее значение:

$$f(x^{***}) = 4.87722877$$

Погрешность вычисления составила $r_k(x^{***}) = 1.404963050385228 \cdot 10^{-5}$.

Выводы

Мы построили таблицу конечных разностей и вычислили приблизительное значение точки восстановления формулой Ньютона для конца таблицы.

Часть 3. Интерполяция по узлам — корням многочлена Чебышева. Минимизация погрешности

Постановка задачи

Построить интерполяционный многочлен Ньютона, в качестве узлов использовать корни многочлена Чебышева. Найти приближенное значение в точках восстановления x^*, x^{**}, x^{***}. Оценить погрешность интерполирования.

Дано:

1. $f(x_i)=f_i \quad , \quad [a,b]=[\alpha_j, 1+\alpha_j]$
2. $\alpha_j=0{,}1+0{,}05\cdot j$, j - номер в списке (14)
3. $x_i=\alpha_j+i\cdot h, i=\overline{0,n}, h=\frac{1}{n}, n=10$
4. $f(x)=\alpha_j\cdot e^x+(1-\alpha_j)\cdot\sin(x)$
5. $x^*=x_0+\frac{2}{3}\cdot h, x^{**}=x_{\frac{n}{2}}+\frac{1}{2}\cdot h, x^{***}=x_n-\frac{1}{3}\cdot h$

Алгоритм решения

Формула корней многочлена Чебышева:

$$\overline{x_k}=\cos\left(\frac{2\cdot k-1}{2\cdot n}\cdot\pi\right), k=\overline{1,n}, n- \text{ количество узлов.}$$

При отображении узлов $[-1,1]\rightarrow[a,b]$:

$$x_k=\frac{a+b}{2}+\frac{b-a}{2}\cdot\overline{x_k}, x_k\in[a,b].$$

Погрешность:

$$r_n(x)=f(x)-L_n(x)=\frac{\omega(x)\cdot f^{(n+1)}(\xi)}{(n+1)!},$$

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

Код решения

Нахождение корней многочлена Чебышева:

```
def chebyshev_points(a: float, b: float) -> np.ndarray:
    """
    Method to get chebyshev points to interpolate.
    """
    points = np.zeros(n_)
    for k in range(0, n_):
        xk = np.cos(((2 * k + 1) * np.pi) / (2 * n_))
        points[k] = ((a + b) / 2) + (((b - a) / 2) * xk)
    return points
```

Нахождение погрешности вычислений:

```
def max_der(n, a, b):
    ans = max(der_11_func(x) for x in np.linspace(a, b, n))
    return ans


def r_n(n, a, b):
    return (max_der(n, a, b) / np.math.factorial(n + 1)) * (np.power(b - a, n + 1) / np.power(2, 2*n + 1))
```

Результаты выполнения

Найденные точки — корни многочлена Чебышева:

```
Chebyshev points: [1.79384417 1.74550326 1.65355339 1.52699525 1.37821723 1.22178277
 1.07300475 0.94644661 0.85449674 0.80615583]
```

Истинные значения точек восстановления:

```
Real values in recovery points: [2.05560924 3.2810851  4.87722887]
```

Восстановленные значения в этих точках:

```
Interpolated in recovery points: [2.05560924 3.2810851  4.87722887]
```

Истинная погрешность и погрешность по формуле:

```
Real difference: [6.803446694902959e-13, 7.851497230149107e-13, 1.3082868122182845e-12]
R_n(x) <= 5.835700868514955e-12
```

Выводы

При использовании узлов интерполирования — корней многочлена Чебышева, мы минимизировали остаток интерполирования, т.е. погрешность в точках восстановления уменьшилась (по сравнению с первой частью лабораторной работы).

Часть 4. Среднеквадратичное приближение

Постановка задачи

Построить полином наилучшего среднеквадратичного приближения степени 5. Найти приближенные значения в точках x^*, x^{**}, x^{***}. Оценить погрешность.

Дано:

1. $f(x_i)=f_i \quad , \quad [a,b]=[\alpha_j, 1+\alpha_j]$
2. $\alpha_j=0{,}1+0{,}05\cdot j$, j - номер в списке (14)
3. $x_i=\alpha_j+i\cdot h, i=\overline{0,n}, h=\frac{1}{n}, n=10$
4. $f(x)=\alpha_j\cdot e^x+(1-\alpha_j)\cdot\sin(x)$
5. $x^*=x_0+\frac{2}{3}\cdot h, x^{**}=x_{\frac{n}{2}}+\frac{1}{2}\cdot h, x^{***}=x_n-\frac{1}{3}\cdot h$

Алгоритм решения

Формула квадратичного приближения:

$$\varphi(x)=\sum_{i=0}^{n}\alpha_i\cdot\varphi_i(x)$$

Для построения квадратичного приближения нужно решить систему вида:

$$A\,\alpha=F,$$

где $a_{i,j}=(\varphi_j,\varphi_i), \quad F=\sum_{i=0}^{m}\varphi_0(x_i)\cdot f_i$

Где (φ_i,φ_j) считается аналогично $(f,g)=\sum_{i=o}^{m}f(x_i)\cdot g(x_i)$

Погрешность:

$$\Delta=\left(\sum_{i=0}^{m}(\varphi(x_i)-f(x_i))^2\right)^{\left(\frac{1}{2}\right)}$$

Код решения

Функции для построения матрицы A:

```
def phi_i(i, x):
    return np.power(x, i)


def get_value_x(i, j, points, m):
    ans = 0
    for r in range(m):
        ans += phi_i(i, points[r]) * phi_i(j, points[r])
    return ans


def get_matrix_a(n, m, points):
    a = np.zeros([n+1, n+1])
    for i in range(n+1):
        for j in range(n+1):
            a[i][j] = get_value_x(i, j, points, m)
    return a
```

Функции для построения вектора F:

```
def get_value_y(i, values, points, m):
    ans = 0
    for r in range(m):
        ans += values[r] * phi_i(i, points[r])
    return ans


def get_vector_f(n, m, values, points):
    ans = np.zeros(n+1)
    for i in range(n+1):
        ans[i] = get_value_y(i, values, points, m)
    return ans
```

Функция $\varphi(x)$:

```
def phi(x, n, coeffs):
    ans = 0
    for i in range(n+1):
        ans += coeffs[i] * np.power(x, i)
    return ans
```

Подсчет погрешности:

```
def delta(m, inter, values):
    sum_ = 0
    for i in range(m):
        sum_ += np.power(inter[i] - values[i], 2)
    return np.sqrt(sum_)
```

Результат выполнения

Построенная матрица A:

```
Matrix A:
10.00000 12.50000 16.45000 22.62500 32.26930 47.38625
12.50000 16.45000 22.62500 32.26930 47.38625 71.21289
16.45000 22.62500 32.26930 47.38625 71.21289 108.99946
22.62500 32.26930 47.38625 71.21289 108.99946 169.29300
32.26930 47.38625 71.21289 108.99946 169.29300 266.05563
47.38625 71.21289 108.99946 169.29300 266.05563 422.16844
```

Полученный вектор F:

```
Vector F:
30.90933 41.04779 56.94372 81.85189 121.02362 182.96126
```

Вычисленный вектор α:

```
Vector alpha:
0.79045 1.04791 0.30234 0.20276 -0.02453 0.02398
```

Полученный многочлен пятой степени:

$$0.02398\,x^5 - 0.02453\,x^4 + 0.20276\,x^3 + 0.30234\,x^2 + 1.04791\,x + 0.79045$$

Полученные значения в точках восстановления:

```
Approximate values in recovery points:
2.05561 3.28109 4.87722
```

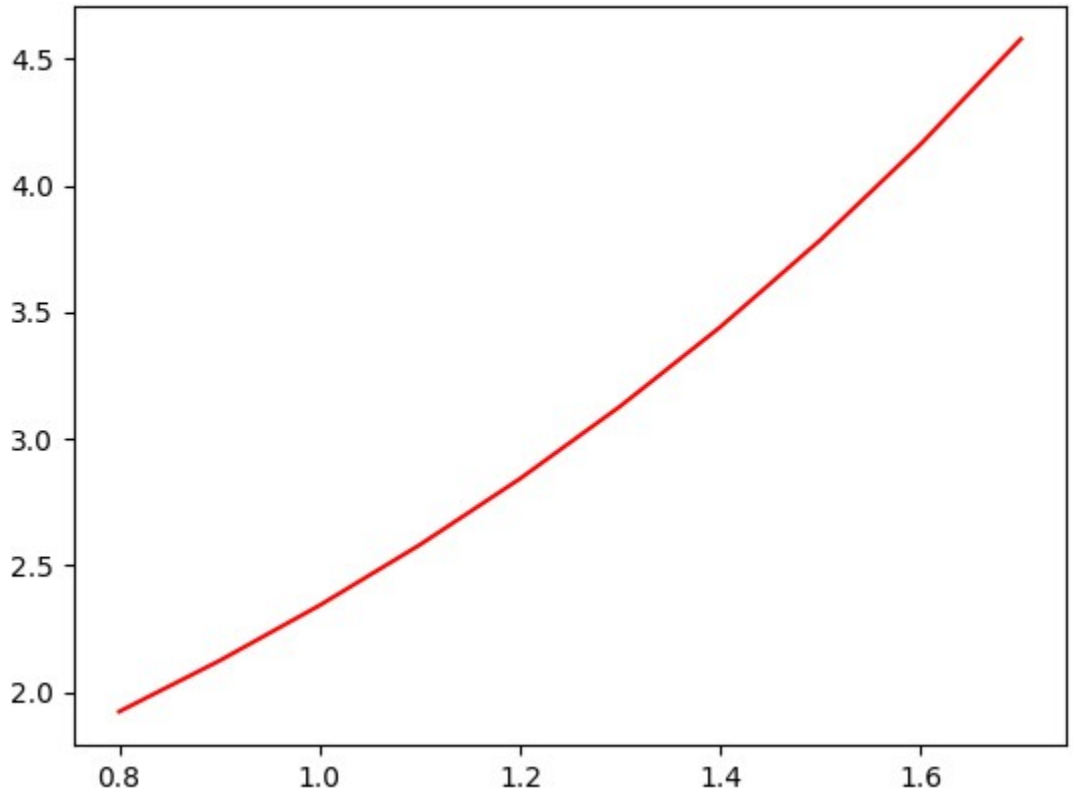
Истинная погрешность в точках восстановления:

```
Real error:
1.31178e-06 1.50525e-07 1.15515e-05
```

Вычисленная по формуле погрешность:

```
Delta:
2.04247e-06
```

График построенного полинома:



Выводы

Мы построили многочлен пятой степени методом меньших квадратов. При сравнении с вторым пунктом лабораторной работы, выяснилось, что погрешность в данном пункте меньше. Это произошло потому, что в данном пункте мы построили полином пятой степени, а во втором пункте третьей, и из-за этого погрешность интерполирования уменьшилась.