Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Численные методы»
Вариант №1

Студент: А. О. Дубинин
Преподаватель: И. Э. Иванов
Группа: М8О-306Б
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №1

Вариант 1

1

Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках $X_i, i = 0, ..., 3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки X_i, Y_i. Вычислить значение погрешности интерполяции в точке X^*.

1. $y = sin(x), \quad a) X_i = 0.1\pi, 0.2\pi, 0.3\pi, 0.4\pi; \quad b) X_i = 0.1\pi, \frac{\pi}{6}, 0.3\pi, 0.4\pi; \quad X^* = \frac{\pi}{4}$

2

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

1. $X^* = 1.5$

i	0	1	2	3	4
x_i	0.0	1.0	2.0	3.0	4.0
f_i	0	0.5	0.86603	1.0	0.86603

3

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

1.

i	0	1	2	3	4	5
x_i	-1.0	0.0	1.0	2.0	3.0	4.0
f_i	-0.5	0	0.5	0.86603	1.0	0.86603

4

Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i), i = 0, 1, 2, 3, 4$ в точке $x = X^*$.

1. $X^* = 1.0$

i	0	1	2	3	4
x_i	-1.0	0.0	1.0	2.0	3.0
y_i	-0.5	0	0.5	0.86603	1.0

5

Вычислить определенный интеграл $F = \int_{x_0}^{x_1} ydx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

1. $y = \dfrac{x}{2x+5}, \quad X_0 = -1, X_k = 1, h_1 = 0.5, h_2 = 0.25;$

1 Решение

1 Многочлен Лагранжа и многочлен Ньютона.

Пусть на отрезке [a, b] задано множество несовпадающих точек x_i (интерполяционных узлов), в которых известны значения функции $f_i = f(x_i), \quad i = 0, ..., n$. Приближающая функция $\phi(x, a)$ такая, что выполняются равенства

$$\phi(x_i, a_0, ..., a_n) = f(x_i) = f_i, \quad i = 0, ..., n,$$

называется интерполяционной.

Наиболее часто в качестве приближающей функции используют многочлены степени n

$$P_n(x) = \sum_{i=0}^{n} a_i x^i$$

Подставляя в значения узлов интерполяции и используя условие $P_n(x_i) = f_i$,получаем систему линейных алгебраических уравнений относительно коэффициентов a_i.

$$\sum_{i=0}^{n} a_i x^i = f_k, \quad k = 0, ..., n$$

которая в случае несовпадения узлов интерполяции имеет единственное решение.

Для нахождения интерполяционного многочлена не обязательно решать систему. Произвольный многочлен может быть записан в виде

$$L_n(x) = \sum_{i=0}^{n} f_i l_i(x)$$

Здесь $l_i(x)$ – многочлены степени n, так называемые лагранжевы многочлены влияния, которые удовлетворяют условию

$$l_i(x_j) = \begin{matrix} 1, i = j \\ 0, i \neq j \end{matrix}$$

и, соответственно,

$$l_i(x_j) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

а интерполяционный многочлен запишется в виде

$$L_n(x) = \sum_{i=0}^{n} f_i \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{x_i - x_j}$$

Интерполяционный многочлен, записанный в форме называется интерполяционным многочленом Лагранжа.

Если ввести функцию $\omega_{n+1}(x) = (x-x_0)(x-x_1)...(x-x_n) = \prod_{i=0}^{n}(x-x_i)$, то выражение для интерполяционного многочлена Лагранжа примет вид:

$$L_n(x) = \sum_{i=0}^{n} f_i \frac{\omega_{n+1}(x)}{(x-x_i)\omega'_{n+1}(x_i)}$$

Недостатком интерполяционного многочлена Лагранжа является необходимость полного пересчета всех коэффициентов в случае добавления дополнительных интерполяционных узлов. Чтобы избежать указанного недостатка используют интерполяционный многочлен в форме Ньютона.

Введем понятие разделенной разности. Разделенные разности нулевого порядка совпадают со значениями функции в узлах. Разделенные разности первого порядка обозначаются $f(x_i, x_j)$ и определяются через разделенные разности нулевого порядка:

$$f_{(x_i,x_j)} = \frac{f_i - f_j}{x_i - x_j},$$

разделенные разности второго порядка определяются через разделенные разности первого порядка:

$$f(x_i, x_j, x_k) = \frac{f(x_i, x_j) - f(x_j, x_k)}{x_i - x_k}.$$

Разделенная разность порядка – kn + 2 определяется соотношениями

$$f(x_i, x_j, x_k, ..., x_{n-1}, x_n) = \frac{f(x_i, x_j, x_k, ..., x_{n-1}) - f(x_j, x_k, ..., x_n)}{x_i - x_n}$$

Таким образом, для (n - 1)-й точки могут быть построены разделенные разности до n-го порядка; разделенные разности более высоких порядков равны нулю.

Пусть известны значения аппроксимируемой функции f(x) в точках $x_0, x_1, ..., x_n$. Интерполяционный многочлен, значения которого в узлах интерполяции совпадают со значениями функции f(x) может быть записан в виде: $P_n(x) = f(x_0) + (x - x_0)f(x_1, x_0) + (x-x_0)(x-x_1)f(x_0, x_1, x_2) + ... + (x-x_0)(x-x_1)...(x-x_n)f(x_0, x_1, ..., x_n)$.

Запись многочлена в формуле есть так называемый интерполяционный многочлен Ньютона. Если функция $f(x)$ не есть многочлен n-й степени, то формула для $P_n(x)$ приближает функцию f(x) с некоторой погрешностью. Отметим, что при добавлении новых узлов первые члены многочлена Ньютона остаются неизменными

Если функция задана в точках $x_0, x_1, ..., x_n$ то при построении интерполяционного

многочлена Ньютона удобно пользоваться таблицей, называемой таблицей разделенных разностей.

Для повышения точности интерполяции в сумму могут быть добавлены новые члены, что требует подключения дополнительных интерполяционных узлов. При этом безразлично, в каком порядке подключаются новые узлы. Этим формула Ньютона выгодно отличается от формулы Лагранжа.

Погрешность интерполяционных многочленов Лагранжа и Ньютона для случая аналитически заданной функции f(x) априорно может быть оценена по формуле, вывод которой приводится, например.

$$|\epsilon_n(x)| = |f(x) - P_n(x)| \leq \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|,$$

где $M_{n+1} = max|f^{(n+1)}(\epsilon)|, \epsilon \in [x_0, x_n]$.

Если величину производных аппроксимируемой функции оценить сложно (например, для таблично заданной функции), то используется апостериорная оценка по первому отброшенному члену интерполяционного многочлена Ньютона, в который входят разделенные разности, являющиеся аналогами производных соответствующих порядков.

2 Кубический сплайн.

Использование одной интерполяционной формулы на большом числе узлов нецелесообразно. Интерполяционный многочлен может проявить свои колебательные свойства, его значения между узлами могут сильно отличаться от значений интерполируемой функции. Одна из возможностей преодоления этого недостатка заключается в применении сплайн-интерполяции. Суть сплайн-интерполяции заключается в определении интерполирующей функции по формулам одного типа для различных непересекающихся промежутков и в стыковке значений функции и её производных на их границах.

Наиболее широко применяемым является случай, когда между любыми двумя точками разбиения исходного отрезка строится многочлен n-й степени:

$$S(x) = \sum_{k=0}^{n} a_{ik}x^k, \quad x_{i-1} \leq x \leq x_i, \quad i = 1, ..., n,$$

который в узлах интерполяции принимает значения аппрокимируемой функции и непрерывен вместе со своими (n - 1) производными. Такой кусочно-непрерывный интерполяционный многочлен называется сплайном. Его коэффициенты находятся из условий равенства в узлах сетки значений сплайна и приближаемой функции, а также равенства n - 1 производных соответствующих многочленов. На практике наиболее часто используется интерполяционный многочлен третьей степени, который удобно представить как

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

Для построения кубического сплайна необходимо построить n многочленов третьей степени, т.е. определить 4n неизвестных a_i, b_i, c_i, d_i. Эти коэффициенты ищутся из условий в узлах сетки.

$$S(x_{i-1}) = a_i = a_{i-1} + b_{i-1}(x_{i-1} - x_{i-2}) + c_{i-1}(x_{i-1} - x_{i-2})^2 + d_{i-1}(x_{i-1} - x_{i-2})^3 = f_{i-1}$$
$$S'(x_{i-1}) = b_i = b_{i-1} + 2c_{i-1}(x_{i-1} - x_{i-2}) + 3d_{i-1}(x_{i-1} - x_{i-2})^2,$$
$$S''(x_{i-1}) = 2c_i = 2c_{i-1} + 6d_{i-1}(x_{i-1} - x_{i-2}),$$
$$S(x_0) = a_1 = f_0$$
$$S''(x_0) = c_1 = 0$$
$$S(x_n) = a_n + b_n(x_n - x_{n-1}) + c_n(x_n - x_{n-1})^2 + d_n(x_n - x_{n-1})^3 = f_n$$
$$S''(x_n) = c_n + 3d_n(x_n - x_{n-1}) = 0$$

Предполагается, что сплайны имеют нулевую кривизну на концах отрезка. В общем случае могут быть использованы и другие условия.

Если ввести обозначение $h_i = x_i - x_{i-1}$, и исключить из системы a_i, b_i, d_i, то можно получить систему из n - 1 линейных алгебраических уравнений относительно $c_i, i = 2, ..., n$ с трехдиагональной матрицей:

$$2(h_1 + h_2)c_2 + h_2c_3 = 3[(f_2 - f_1)/h_2 - (f_1 - f_0)/h_1]$$
$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_ic_{i+1} = 3[(f_i - f_{i-1})/h_i - (f_{i-1} - f_{i-2})/h_{i-1}], i = 3, ..., n-1$$
$$h_{n-1}c_{n-1} + 2(h_{n-1} + h_n)c_n = 3[(f_n - f_{n-1})/h_n - (f_{n-1} - f_{n-2})/h_{n-1}],$$

Остальные коэффициенты сплайнов могут быть восстановлены по формулам:

$$a_i = f_{i-1}, i = 1, ..., n; b_i = (f_i - f_{i-1})/h_i - \frac{1}{3}h_i(c_{i+1} + 2c_i), d_i = \frac{c_{i+1} - c_i}{3h_i}, i = 1, ..., n-1$$

$$c_1 = 0, b_n = (f_n - f_{n-1})/h_n - \frac{2}{3}h_nc_n, d_n = -\frac{c_n}{3h_n}$$

3 Метод наименьших квадратов.

Пусть задана таблично в узлах x_j функция $y_j = f(x_j), j = 0, 1, ..., N$. При этом значения функции y_j определены с некоторой погрешностью, также из физических

соображений известен вид функции, которой должны приближенно удовлетворять табличные точки, например: многочлен степени n, у которого неизвестны коэффициенты $a_i, F_n(x) = \sum_{i=0}^{n} a_i x^i$.

Неизвестные коэффициенты будем находить из условия минимума квадратичного отклонения многочлена от таблично заданной функции.

$$\Phi = \sum_{j=0}^{N} [F_n(x_j) - y_j]^2.$$

Минимума Ф можно добиться только за счет изменения коэффициентов многочлена $F_n(x)$. Необходимые условия экстремума имеют вид

$$\frac{\delta \Phi}{\delta a_k} = 2 \sum_{j=0}^{N} [\sum_{i=0}^{n} a_i x_j^i - y_j] x_j^k = 0, k = 0, 1, ..., n$$

Эту систему для удобства преобразуют к следующему виду:

$$\sum_{i=0}^{n} a_i \sum_{j=0}^{N} x_j^{k+i} = \sum_{j=0}^{N} y_j x_j^k, k = 0, 1, ..., n.$$

Система называется нормальной системой метода наименьших квадратов (МНК) представляет собой систему линейных алгебраических уравнений относительно коэффициентов a_i. Решив систему, построим многочлен $F_n(x)$, приближающий функцию $f(x)$ и минимизирующий квадратичное отклонение.

Необходимо отметить, что система с увеличением степени n приближающего многочлена становится плохо обусловленной и решение её связано с большой потерей точности. Поэтому при использовании метода наименьших квадратов, как правило, используют приближающий многочлен не выше третьей степени.

4 Численное дифференцирование.

Формулы численного дифференцирования в основном используются при нахождении производных от функции y = f(x), заданной таблично. Исходная функция $y_i = f(x_i), i = 0, 1, ..., M$ на отрезках $[x_j, x_{j+k}]$ заменяется некоторой приближающей, легко вычисляемой функцией $\phi(x, \widetilde{a}), y = \phi(x, \widetilde{a}) + R(x)$, где R(x) – остаточный член приближения, $\widetilde{a}$ - набор коэффициентов, вообще говоря, различный для каждого из рассматриваемых отрезков, и полагают, что $y'(x) \approx \phi'(x, \widetilde{a})$. Наиболее часто в качестве приближающей функции $\phi(x, \widetilde{a})$ берется интерполяционный многочлен $\phi(x, \widetilde{a}) = P_n(x) = \sum_{i=0}^{n} a_i x^i$, а производные соответствующих порядков определяются дифференцированием многочлена.

При решении практических задач, как правило, используются аппроксимации первых и вторых производных.

В первом приближении, таблично заданная функция может быть аппроксимирована отрезками прямой $y(x) \approx \phi(x) = y_i + \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i), x \in [x_i, x_{i+1}]$. В этом случае:

$$y'(x) \approx \phi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = const, x \in [x_i, x_{i+1}]$$

производная является кусочно-постоянной функцией и рассчитывается, по формуле с первым порядком точности в крайних точках интервала, и со вторым порядком точности в средней точке интервала.

При использовании для аппроксимации таблично заданной функции интерполяционного многочлена второй степени имеем:

$$y(x) \approx \phi(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + \frac{\frac{y_{i+2}-y_{i+1}}{x_{i+2}-x_{i+1}} - \frac{y_{i+1}-y_i}{x_{i+1}-x_i}}{x_{i+2} - x_i}(x - x_i)(x - x_{i+1}), x \in [x_i, x_{i+1}]$$

$$y'(x) \approx \phi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + \frac{\frac{y_{i+2}-y_{i+1}}{x_{i+2}-x_{i+1}} - \frac{y_{i+1}-y_i}{x_{i+1}-x_i}}{x_{i+2} - x_i}(2x - x_i - x_{i+1}), x \in [x_i, x_{i+1}]$$

При равностоящих точках разбиения, данная формула обеспечивает второй порядок точности.

Для вычисления второй производной, необходимо использовать интерполяционный многочлен, как минимум второй степени. После дифференцирования многочлена получаем

$$y''(x) \approx \phi''(x) = 2\frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + \frac{\frac{y_{i+2}-y_{i+1}}{x_{i+2}-x_{i+1}} - \frac{y_{i+1}-y_i}{x_{i+1}-x_i}}{x_{i+2} - x_i}, x \in [x_i, x_{i+1}].$$

5 Численное интегрирование.

Формулы численного интегрирования используются в тех случаях, когда вычислить аналитически определенный интеграл $F = \int_a^b f(x)dx$ не удается. Отрезок [a,b] разбивают точками $x_0, ..., x_N$, так что $a = x_0 \leq x_1 \leq ... \leq x_N = b$ с достаточно мелким шагом $h_i = x_i - x_{i-1}$ и на одном или нескольких отрезках h_i подынтегральную функцию f(x) заменяют такой приближающей $\phi(x)$, так что она, во-первых, близка f(x), а, во-вторых, интеграл от $\phi(x)$ легко вычисляется. Рассмотрим наиболее простой и часто применяемый способ, когда подынтегральную функцию заменяют на интерполяционный многочлен $P_n(x) = \sum_{j=0}^{n} a_j x^j$, причем коэффициенты многочлена a_j, вообще говоря, различны на каждом отрезке $[x_i, x_{i+k}]$ и определяются из условия $\phi(x_j) = f(x_j), j = i, ..., i + k$, т.е. многочлен P_n зависит от параметров $a_j - P_n(x, \widetilde{a_i})$, тогда

$$f(x) = P_n(x, \widetilde{a_i}) + R_n(x, \widetilde{a_i}), x \in [x_i, x_{i+k}],$$

где $R_n(x, \widetilde{a_i})$ – остаточный член интерполяции. Тогда $F = \sum_{i=1}^{N} \int_{x_{-1}}^{x_i} P_n(x, \widetilde{a_i})dx + R$, где $R = \sum_{i=1}^{N} \int_{x_{-1}}^{x_i} R_n(x, \widetilde{a_i})dx$ – остаточный член формулы численного интегрирования или её погрешность.

При использовании интерполяционных многочленов различной степени, получают формулы численного интегрирования различного порядка точности.

Заменим подынтегральную функцию, интерполяционным многочленом Лагранжа нулевой степени, проходящим через середину отрезка – точку $\widetilde{x_i}= (x_{i-1} + x_i)/2$, получим формулу прямоугольников.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{N} h_i f(\frac{x_{i-1} + x_i}{2})$$

В случае постоянного шага интегрирования $h_i = h, i = 1, 2, ..., N$ и существования $f''(x), x \in [a, b]$, имеет место оценка остаточного члена формулы прямоугольников

$$R \leq \frac{1}{24} h^2 M_2 (b - a),$$

где $M_2 = max|f''(x)|_{[a,b]}$.

В случае таблично заданных функций удобно в качестве узлов интерполяции выбрать начало и конец отрезка интегрирования, т.е. заменить функцию f (x) многочленом Лагранжа первой степени.

$$F = \int_a^b f(x)dx \approx \frac{1}{2} \sum_{i=1}^{N} (f_i + f_{i-1}) h_i$$

Эта формула носит название формулы трапеций.

В случае постоянного шага интегрирования величина остаточного члена оценивается

$$R \leq \frac{b - a}{12} h^2 M_2$$

, где $M_2 = max|f''(x)|_{[a,b]}$

Для повышения порядка точности формулы численного интегрирования заменим подынтегральную кривую параболой – интерполяционным многочленом второй степени, выбрав в качестве узлов интерполяции концы и середину отрезка интегрирования: $x_{i-1}, x_{i-\frac{1}{2}} = (x_{i-1} + x_i)/2, x_i$.

Для случая $h_i = \frac{x_i - x_{i-1}}{2}$, получим формулу Симпсона (парабол)

$$F = \int_a^b f(x)dx \approx \frac{1}{3}\sum_{i=1}^{N}(f_{i-1} + 4f_{i-\frac{1}{2}} + f_i)h_i$$

В случае постоянного шага интегрирования $h_i = h, i = 1, 2, ..., N$, формула Симпсона принимает вид.

$$F \approx \frac{h}{3}[f_0 + 4f_{\frac{1}{2}} + 2f_1 + 4f_{\frac{3}{2}} + 2f_2 + ... + 2f_{N-1} + 4f_{N-\frac{1}{2}} + f_N],$$

при этом количество интервалов на которое делится отрезок интегрирования, равно 2N.

В том случае если существует $f^{IV}(x), x \in [a, b]$, для оценки величины погрешности справедлива мажорантная оценка

$$R \leq \frac{(b-a)}{180}h^4 M_4,$$

где $M_4 = max|f^{IV}(x)|_{[a,b]}$

Метод Рунге-Ромберга-Ричардсона позволяет получать более высокий порядок точности вычисления. Если имеются результаты вычисления определенного интеграла на сетке с шагом $h - F = F_h + O(h^p)$ и на сетке с шагом $kh - F = F_{kh} + O((kh)^p)$,то

$$F = \int_a^b f(x)dx = F_h = \frac{F_h - F_{kh}}{k^p - 1} + O(h^{p+1})$$

2 Исходный код

1 Многочлен Лагранжа и многочлен Ньютона.

```
import argparse
import numpy as np


def lagrange(points):
    def inner(x):
        cnt_points = len(points)
        l = []
        for i in range(cnt_points):
            l.append(np.prod([(x - points[j][0]) / (points[i][0] - points[j][0])
                              for j in range(cnt_points) if i != j]))
        return sum([points[idx][1] * i for idx, i in enumerate(l)])
    return inner


def newton(points):
    def inner(x):
        cnt_points = len(points)
        x_point, y_point = zip(*points)
        coef = [y_point[0]]
        for j, shift in zip(reversed(range(1, cnt_points)), range(cnt_points)):
            tmp = []
            for l, r in zip(range(j), range(1, j + 1)):
                num1 = y_point[l] - y_point[r]
                num2 = x_point[l] - x_point[r + shift]
                tmp.append(num1 / num2)
            y_point = tmp
            coef.append(y_point[0])

        res = 0
        for i in range(cnt_points):
            res += coef[i] * np.prod([x - x_point[j] for j in range(i)])
        return res
    return inner


def get_points(file_name):
    with open(file_name) as f:
        x = [float(num) * np.pi for num in f.readline().split()]
        points = [(i, np.sin(i)) for i in x]
        x_test = float(f.readline()) * np.pi
    return points, x_test


def main():
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument('--input', required=True, help='Input test file')
    parser.add_argument('--output', required=True, help='File for answer')
    args = parser.parse_args()

    points, x_test = get_points(args.input)
    lg = lagrange(points)
    nt = newton(points)

    with open(args.output, 'w') as f:
        tmp = round(x_test, 3)
        res1 = lg(tmp)
        res2 = nt(tmp)
        res3 = np.sin(tmp)
        eps1 = abs(res1 - res3)
        eps2 = abs(res2 - res3)
        f.write(f'Lagrange:\nL({tmp}) = {res1}\nEps = {round(eps1, 5)}\n')
        f.write(f'Newton:\nN({tmp}) = {res2}\nEps = {round(eps2, 5)}\n')
        f.write(f'Orig:\nSin({tmp}) = {res3}\n')

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

2 Кубический сплайн.

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


def tma(matrix, d, shape):
    a, b, c = zip(*matrix)
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    x = [0] * (shape + 1)
    for i in range(1, shape):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))
    for i in reversed(range(shape)):
        x[i] = p[i] * x[i + 1] + q[i]
    return x[:-1]


def spline(x, y):
    size = len(x)
    h = [x[i] - x[i - 1] for i in range(1, size)]
    mtrx = [[0, 2 * (h[0] + h[1]), h[1]]]
    b = [3 * ((y[2] - y[1]) / h[1] - (y[1] - y[0]) / h[0])]
    for i in range(1, size - 3):
        tmp = [h[i], 2 * (h[i] + h[i + 1]), h[i + 1]]
        mtrx.append(tmp)
```

```
        b.append(3 * ((y[i + 2] - y[i + 1]) / h[i + 1] - (y[i + 1] - y[i]) / h[i]))
    mtrx.append([h[-2], 2 * (h[-2] + h[-1]), 0])
    b.append(3 * ((y[-1] - y[-2]) / h[-1] - (y[-2] - y[-3]) / h[-2]))
    c = tma(mtrx, b, size - 2)
    a = []
    b = []
    d = []
    c.insert(0, 0)
    for i in range(1, size):
        a.append(y[i - 1])
        if i < size - 1:
            d.append((c[i] - c[i - 1]) / (3 * h[i - 1]))
            b.append((y[i] - y[i - 1]) / h[i - 1] -
                    h[i - 1] * (c[i] + 2 * c[i - 1]) / 3)
    b.append((y[-1] - y[-2]) / h[-1] - 2 * h[-1] * c[-1] / 3)
    d.append(-c[-1] / (3 * h[-1]))
    return a, b, c, d


def get_points(file_name):
    with open(file_name) as f:
        x = [float(num) for num in f.readline().split()]
        y = [float(num) for num in f.readline().split()]
        x_test = float(f.readline())
    return x, y, x_test


def polyval(x0, x, k, coef):
    a, b, c, d = coef
    tmp = (x0 - x[k])
    return a[k] + b[k] * tmp + c[k] * tmp**2 + d[k] * tmp**3


def pol(x, x_test, coef):
    k = 0
    for i, j in zip(x, x[1:]):
        if i <= x_test <= j:
            break
        k += 1
    return polyval(x_test, x, k, coef)


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input', required=True, help='Input test file')
    parser.add_argument('--output', required=True, help='File for answer')
    args = parser.parse_args()

    x, y, x_test = get_points(args.input)
```

```
76
77      '''
78      Test Runge's phenomenon
79      f = lambda x: 1 / (1 + 25*x**2)
80      x_test = 1
81      x = list(range(6))
82      y = [f(i) for i in x]
83      # '''
84
85      coef = spline(x, y)
86
87      x1 = np.linspace(x[0], x[-1], 50)
88      y1 = [pol(x, i, coef) for i in x1]
89
90      plt.plot(x1, y1, color='b')
91      plt.scatter(x, y, color='r')
92      plt.show()
93
94      res = pol(x, x_test, coef)
95
96      for i in x:
97          print(pol(x, i, coef))
98
99      with open(args.output, 'w') as f:
100         f.write(f'x = {x}\ny = {y}\n\n')
101         f.write(f'a = {coef[0]}\nb = {coef[1]}\nc = {coef[2]}\nd = {coef[3]}\n')
102         f.write(f'\nf({x_test}) = {round(res, 4)}\n')
103
104 if __name__ == "__main__":
105     main()
```

3 Метод наименьших квадратов.

```
1  import argparse
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from numpy.linalg import inv
5
6
7  def get_points(file_name):
8      with open(file_name) as f:
9          x = [float(num) for num in f.readline().split()]
10         y = [float(num) for num in f.readline().split()]
11     return x, y
12
13
14 def mls(k, x, y):
15     x = np.array(x)
16     t = np.array([[i**j for i in x] for j in reversed(range(k))])
17     t_trans = np.transpose(t)
```

```
    g = t t_trans
    a = inv(g) t y
    return a


def cal_err(x, y, coef2, coef3):
    y_err2 = []
    y_err3 = []
    for i in x:
        y_err2.append(np.polyval(coef2, i))
        y_err3.append(np.polyval(coef3, i))
    err1 = sum([(y_err2[idx] - i)**2 for idx, i in enumerate(y)])
    err2 = sum([(y_err3[idx] - i)**2 for idx, i in enumerate(y)])
    return err1, err2


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input', required=True, help='Input test file')
    parser.add_argument('--output', required=True, help='File for answer')
    args = parser.parse_args()

    x, y = get_points(args.input)

    coef2 = mls(2, x, y)
    coef3 = mls(3, x, y)

    # print(coef2, coef3)

    area = np.linspace(x[0], x[-1], 50)
    y_coef2 = [np.polyval(coef2, i) for i in area]
    y_coef3 = [np.polyval(coef3, i) for i in area]



    plt.scatter(x, y, color='r')
    plt.plot(area, y_coef2, color='b')
    plt.plot(area, y_coef3, color='g')
    plt.show()

    with open(args.output, 'w') as f:
        err1, err2 = cal_err(x, y, coef2, coef3)
        f.write(f'ax + b:\n Coefficients: {coef2},\n Error: {round(err1, 5)}\n')
        f.write(f'ax^2 + bx + c:\n Coefficients: {coef3},\n Error {round(err2, 5)}\n')

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

4 Численное дифференцирование.

```
import argparse


def get_points(file_name):
    with open(file_name) as f:
        x = [float(num) for num in f.readline().split()]
        y = [float(num) for num in f.readline().split()]
        x_test = float(f.readline())
    return x, y, x_test


def first_der(x, y, x0, k):
    num1 = (y[k + 1] - y[k]) / (x[k + 1] - x[k])
    num2 = (y[k + 2] - y[k + 1]) / (x[k + 2] - x[k + 1]) - num1
    num2 = num2 / (x[k + 2] - x[k])
    return num1 + num2 * (2 * x0 - x[k] - x[k + 1])


def second_der(x, y, k):
    num1 = (y[k + 2] - y[k + 1]) / (x[k + 2] - x[k + 1])
    num2 = (y[k + 1] - y[k]) / (x[k + 1] - x[k])
    return 2 * (num1 - num2) / (x[k + 2] - x[k])


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input', required=True, help='Input test file')
    parser.add_argument('--output', required=True, help='File for answer')
    args = parser.parse_args()

    x, y, x_test = get_points(args.input)

    k = 0
    for i, j in zip(x, x[1:]):
        if i <= x_test <= j:
            break
        k += 1

    res1 = first_der(x, y, x_test, k)
    res2 = second_der(x, y, k)
    with open(args.output, 'w') as f:
        f.write(f"f'({x_test}) = {round(res1, 5)}\n")
        f.write(f"f''({x_test}) = {round(res2, 5)}\n")

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

5 Численное интегрирование.

```
import argparse
```

```
import numpy as np


# x0, xk, h1, h2
def get_points(file_name):
    with open(file_name) as f:
        a, b, h1, h2 = [float(num) for num in f.readline().split()]
    return a, b, h1, h2

def f(x):
    return x / (2 * x + 5)


def simpson(a, b, h, n):
    s = f(a) + f(b)
    for i in range(1, n, 2):
        s += 4 * f(a + i * h)
    for i in range(2, n-1, 2):
        s += 2 * f(a + i * h)
    return s * h / 3


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--input', required=True, help='Input test file')
    parser.add_argument('--output', required=True, help='File for answer')
    args = parser.parse_args()

    a, b, h1, h2 = get_points(args.input)

    x1 = np.linspace(a, b, int((b - a) / h1 + 1))
    x2 = np.linspace(a, b, int((b - a) / h2 + 1))

    y_trap1 = [f(i) for i in x1]
    y_trap2 = [f(i) for i in x2]

    rect1 = h1 * sum([f((i + j) / 2) for i, j in zip(x1, x1[1:])])
    rect2 = h2 * sum([f((i + j) / 2) for i, j in zip(x2, x2[1:])])

    trap1 = h1/2 * sum([i + j for i, j in zip(y_trap1[1:], y_trap1)])
    trap2 = h2/2 * sum([i + j for i, j in zip(y_trap2[1:], y_trap2)])

    simps1 = simpson(a, b, h1, int((b - a) / h1))
    simps2 = simpson(a, b, h2, int((b - a) / h2))

    with open(args.output, 'w') as fi:
        fi.write(f'Method of rectangles = {rect1}\tStep = {h1}\n')
        fi.write(f'Method of rectangles = {rect2}\tStep = {h2}\n')
        fi.write(f'Error: {round(abs(rect1 - rect2) / 3, 5)}\n')
```

```
51          fi.write('='*10)
52          fi.write('\n')
53          fi.write(f'Method of trapeziums = {trap1}\tStep = {h1}\n')
54          fi.write(f'Method of trapeziums = {trap2}\tStep = {h2}\n')
55          fi.write(f'Error: {round(abs(trap1 - trap2) / 3, 5)}\n')
56          fi.write('='*10)
57          fi.write('\n')
58          fi.write(f"Simpson's method = {simps1}\tStep = {h1}\n")
59          fi.write(f"Simpson's method = {simps2}\tStep = {h2}\n")
60          fi.write(f'Error: {round(abs(simps1 - simps2) / 15, 5)}\n')
61
62  if __name__ == "__main__":
63      main()
```

3 Вывод программы

1 Многочлен Лагранжа и многочлен Ньютона.

1. $y = sin(x), \quad a) X_i = 0.1\pi, 0.2\pi, 0.3\pi, 0.4\pi; \quad b) X_i = 0.1\pi, \frac{\pi}{6}, 0.3\pi, 0.4\pi; \quad X^* = \frac{\pi}{4}$

Входные данные: Имя файла с входными точками, имя файла для результата.

Выходные данные: В выходном файле значение для многочлена лагранжа, погрешность, значение для многочлена ньютона, погрешность и значение оригинальной функции.

```
art@mars:~/study/NM/lab_3/p_1 python3 Lw3_1.py --input input1 --output output1
art@mars:~/study/NM/lab_3/p_1 python3 Lw3_1.py --input input2 --output output2
art@mars:~/study/NM/lab_3/p_1 cat output1
Lagrange:
L(0.785) = 0.7066650831763222
Eps = 0.00016
Newton:
N(0.785) = 0.706665083176322
Eps = 0.00016
Orig:
Sin(0.785) = 0.706825181105366
art@mars:~/study/NM/lab_3/p_1 cat output2
Lagrange:
L(0.785) = 0.7065637262572506
Eps = 0.00026
Newton:
N(0.785) = 0.7065637262572506
Eps = 0.00026
Orig:
Sin(0.785) = 0.706825181105366
```

2 Кубический сплайн.

1. $X^* = 1.5$

i	0	1	2	3	4
x_i	0.0	1.0	2.0	3.0	4.0
f_i	0	0.5	0.86603	1.0	0.86603

Входные данные: Имя файла с входными данными, имя файла для результата.

Выходные данные: В выходном файле значения коэффициентов для каждого отрезка и вычисленное значение.

```
art@mars:~/study/NM/lab_3/p_2 python3 Lw3_2.py --input input --output output
0.0
0.5
0.86603
1.0
0.86603
art@mars:~/study/NM/lab_3/p_2 cat output
x = [0.0,1.0,2.0,3.0,4.0]
y = [0.0,0.5,0.86603,1.0,0.86603]

a = [0.0,0.5,0.86603,1.0]
b = [0.52409375,0.45181249999999995,0.26674624999999996,-0.01879749999999998]
c = [0,-0.07228125000000005,-0.11278499999999991,-0.17275875000000007]
d = [-0.024093750000000014,-0.013501249999999956,-0.019991250000000054,0.057586250000
f(1.5) = 0.7061
```

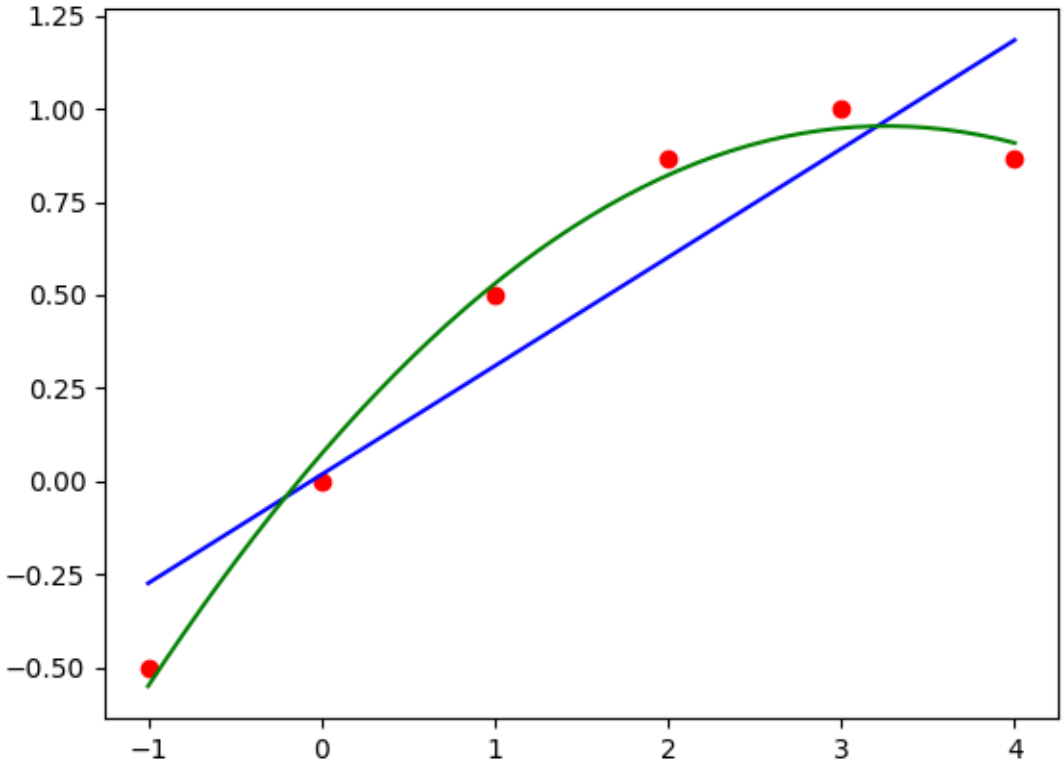
3 Метод наименьших квадратов. 1.

i	0	1	2	3	4	5
x_i	-1.0	0.0	1.0	2.0	3.0	4.0
f_i	-0.5	0	0.5	0.86603	1.0	0.86603

Входные данные: Имя файла с входными данными, имя файла для результата.

Выходные данные: В выходном файле значения коэффицентов для многочлена 1-ой и 2-ой степени степени и ошибки. График приближаемой функции и приближающих многочленов.

```
art@mars:~/study/NM/lab_3/p_3 python3 Lw3_3.py --input input --output output
art@mars:~/study/NM/lab_3/p_3 cat output
ax + b:
Coefficients: [0.29131943 0.01836419],
Error: 0.27082
ax^2 + bx + c:
Coefficients: [-0.08274946  0.53956782  0.0735305 ],
Error 0.01518
```



4 Численное дифференцирование.

1. $X^* = 1.0$

i	0	1	2	3	4
x_i	-1.0	0.0	1.0	2.0	3.0
y_i	-0.5	0	0.5	0.86603	1.0

Входные данные: Имя файла с входными данными, имя файла для результата.

Выходные данные: В выходном файле значения первой и второй производной в точке $x = X^*$.

```
art@mars:~/study/NM/lab_3/p_4 python3 Lw3_4.py --input input --output output
art@mars:~/study/NM/lab_3/p_4 cat output
f'(1.0) = 0.43301
f''(1.0) = -0.13397
```

5 Численное интегрирование.

1. $y = \dfrac{x}{2x+5}, \quad X_0 = -1, X_k = 1, h_1 = 0.5, h_2 = 0.25;$

Входные данные: Имя файла с входными данными, имя файла для результата.

Выходные данные: В выходном файле значения значения интеграла методами прямоугольников, трапеции, Симпсона с шагами h1, h2 и погрешность вычисления методом Рунге-Ромберга.

```
art@mars:~/study/NM/lab_3/p_5 python3 Lw3_5.py --input input --output output
art@mars:~/study/NM/lab_3/p_5 cat output
Method of rectangles = -0.05450105450105448 Step = 0.5
Method of rectangles = -0.05794799368631646 Step = 0.25
Error: 0.00115
==========
Method of trapeziums = -0.06845238095238095 Step = 0.5
Method of trapeziums = -0.06147671772671773 Step = 0.25
Error: 0.00233
==========
Simpson's method  = -0.05952380952380953 Step = 0.5
Simpson's method  = -0.05915149665149664 Step = 0.25
Error: 2e-05
```

4 Выводы

Благодаря этой лабораторной работе, я узнал, что можно быстро с помощью компьютера находить многочлены Лагранжа и Ньютона по заданными точкам, вычислять кубический сплайн, решать нормальную систему МНК, вычислять первую и вторую производную и вычислять определенный интеграл.

Список литературы

[1] *Численные методы. Учебник*
Пирумов Ульян Гайкович, Гидаспов Владимир Юрьевич
(ISBN 978-5-534-03141-6)