Московский Авиационный Институт
(Государственный Технический Университет)

Факультет прикладной математики и физики.
Кафедра вычислительной математики и программирования.

Лабораторная работа №3
по курсу «Численные методы»

VI семестр

Студент Баскаков О.А.
Группа 08-306
Вариант 1

Москва, 2011.

Постановка задачи

1 Интерполяция многочленом

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^*.

$$y = \sin x$$

$$X_i = 0.1\pi, 0.2\pi, 0.3\pi, 0.4\pi$$

$$X^* = \pi/4$$

2 Интерполяция сплайнами

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^*.

$X_i = [0.0, \quad 1.0, \quad 2.0, \quad 3.0, \quad 4.0]$ $\quad X^* = 1.5$

$f_i = [0.0, \quad 0.5, \quad 0.86603, \quad 1.0, \quad 0.86603]$

3 Метод наименьших квадратов

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.

$X_i = [-1.0, \quad 0.0, \quad 1.0, \quad 2.0, \quad 3.0, \quad 4.0]$

$Y_i = [-0.5, \quad 0.0, \quad 0.5, \quad 0.86603, \quad 1.0, \quad 0.86603]$

4 Численное дифференцирование

Вычислить первую и вторую производную от таблично заданной функции в точке X^*.

$X_i = [-1.0, \quad 0.0, \quad 1.0, \quad 2.0, \quad 3.0]$ $\quad X^* = 1.0$

$Y_i = [-0.5, \quad 0.0, \quad 0.5, \quad 0.86603, \quad 1.0]$

5 Численное интегрирование

Вычислить определенный интеграл методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

$$y = \frac{x}{2x + 5}$$

$$X_0 = -1, \quad X_k = 1, \quad h_1 = 0.5, \quad h_2 = 0.25$$

Теоретическая часть ИНТЕРПОЛЯЦИЯ

Пусть на отрезке $[a, b]$ задано множество несовпадающих точек x_i (интерполяционных узлов), в которых известны значения функции f = $f(x_i)$, i = 0,... n .

Приближающая функция p(x, a) такая, что выполняются равенства

$$\varphi(x_i, a_0, \ldots, a_n) = f(x_i) = f_i, \; i = 0, \ldots n .$$

называется интерполяционной.

Наиболее часто в качестве приближающей функции используют многочлены степени n.

Для нахождения интерполяционного многочлена не обязательно решать систему

$$\sum_{i=0}^{n} a_i x^i = f_k , \; k = 0, \ldots, n ,$$

(3.3). 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 многочлен

влияния, которые удовлетворяют условию $l_i(x_j) = \begin{cases} 1, \text{при } i = j, \\ 0, \text{при } i \neq j. \end{cases}$ и, соответственно

$l_i(x) = \prod_{j=0,\ j\neq i}^{n} \frac{(x - x_i)}{(x_i - x_j)}$, а интерполяционный многочлен (3.4) запишется в виде

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

Интерполяционный многочлен, записанный в форме (3.5), 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 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 $f(x_t, x_j)$ и определяются через разделенные разности нулевого порядка.

Пусть известны значения аппроксимируемой функции f (x) в точках $x_0, x_1, \ldots, x_n$.

Интерполяционный многочлен, значения которого в узлах интерполяции совпадают со значениями функции $f(x)$ может быть записан в виде:

$$P_n(x) = f(x_0) + (x - x_0) f(x_1, x_0) + (x - x_0)(x - x_1) f(x_0, x_1, x_2) + \ldots +$$
$$+ (x - x_0)(x - x_1)...(x - x_n) f(x_0, x_1, ..., x_n).$$

Запись многочлена в формуле (3.8) есть так называемый интерполяционный многочлен Ньютона. Если функция f (x) не есть многочлен n-й степени, то формула (3.8) 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 $\mathbf{x_0}$, $\mathbf{x_1}$,... , $\mathbf{x_n}$, то при построении интерполяционного

многочлена Ньютона удобно пользоваться таблицей, называемой таблицей разделенных разностей, пример которой для $n = 4$ приведен в табл. 3.1.

Таблица 3.1

x_0	$f(x_0)$				
x_1	$f(x_1)$	$f(x_0,x_1)$			
x_2	$f(x_2)$	$f(x_1,x_2)$	$f(x_0,x_1,x_2)$		
x_3	$f(x_3)$	$f(x_2,x_3)$	$f(x_1,x_2,x_3)$	$f(x_0,x_1,x_2,x_3)$	$f(x_0,x_1,$
x_4	$f(x_4)$	$f(x_3,x_4)$	$f(x_2,x_3,x_4)$	$f(x_1,x_2,x_3,x_4)$	

Для повышения точности интерполяции в сумму (3.8) 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

$$\left|\varepsilon_n(x)\right| = \left|f(x) - P_n(x)\right| \le \frac{M_{n-1}}{(n+1)!}\left|\omega_{n-1}(x)\right|,$$

где $M_{n1} = \max\left|f^{(n-1)}(\xi)\right|, \ \xi \in [x_0, x_n]$.

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 в [1].

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

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:

$$S(x) = \sum_{k=0}^{n} a_{ik} x^k, \quad x_{i-1} \le x \le x_i, i = 1,\ldots,n, \tag{3.10}$$

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

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

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

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.

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

В (3.12) 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 (3.12) a_i, b_i, d_i, то можно получить систему из $n-1$ линейных алгебраических уравнений относительно c_i, $i = 2,\ldots,n$ с трехдиагональной матрицей:

$$
\begin{aligned}
&2(h_1 + h_2)c_2 + h_2 c_3 = 3[(f_2 - f_1)/h_2 - (f_1 - f_0)/h_1]\\
&h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i-1} = 3\left[(f_i - f_{i-1})/h_i - (f_{i-1} - f_{i-2})/h_{i-1}\right],\ i = \quad 3,\ldots,n-1\\
&h_{n-1}c_{n-1} + 2(h_{n-1} + h_n)c_n = 3[(f_n - f_{n-1})/h_n - (f_{n-1} - f_{n-2})/h_{n-1}]
\end{aligned}
\qquad (3.13)
$$

Остальные коэффициенты сплайнов могут быть восстановлены по формулам:

$$
\begin{aligned}
&a_i = f_{i-1},\ i = 1,..n; \quad b_i = (f_i - f_{i-1})/h_i - \frac{1}{3}h_i(c_{i-1} + 2c_i), \quad d_i = \frac{c_{i-1} - c_i}{3h_i},\ i = 1,\ldots,n-1\\
&c_1 = 0, \quad b_n = (f_n - f_{n-1})/h_n - \frac{2}{3}h_n c_n, \quad d_n = -\frac{c_n}{3h_n}
\end{aligned}
\qquad (3.14)
$$

3.2. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Пусть задана таблично в узлах x_j функция $y_j = f(x_j)$, $j = 0,1,...,N$. значения функции y_j определены с некоторой погрешностью, также из соображений известен вид функции, которой должны приближенно уд табличные точки, например: многочлен степени n, у которого неизвестны коэ a_i, $F_n(x) = \sum_{i=0}^{n} a_i x^i$. Неизвестные коэффициенты будем находить из условия

квадратичного отклонения многочлена от таблично заданной функции.

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

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

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

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 (3.17) называется нормальной системой метода наименьших квадратов (МНК) представляет собой систему линейных алгебраических уравнений относительно коэффициентов *a*.. Решив систему, построим многочлен F_n (X), приближающий функцию / (*x)* и минимизирующий квадратичное отклонение.

Необходимо отметить, что система (3.17) 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.

3.3. 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 $\left[x_j, x_{j+k}\right]$ заменяется некоторой приближающей, легко вычисляемой функцией $\varphi(x,\overline{a})$, $y = \varphi(x,\overline{a}) + R(x)$, где $R(x)$ – остаточный член приближения, $\overline{a}$ - набор коэффициентов, вообще говоря, различный для каждого из рассматриваемых отрезков, и полагают, что $y'(x) \approx \varphi'(x,\overline{a})$. Наиболее часто в качестве приближающей функции $\varphi(x,\overline{a})$ берется интерполяционный многочлен $\varphi(x,\overline{a}) = P_n(x) = \sum_{i=0}^{n} a_i x^i$, 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 $y(x) \approx \varphi(x) = y_i + \frac{y_{i-1} - y_i}{x_{i-1} - x_i}(x - x_i)$, $x \in \left[x_i, x_{i+1}\right]$. В этом случае:

$$y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = const,\ x \in \left[x_i, x_{i-1}\right], \tag{3.18}$$

производная является кусочно-постоянной функцией и рассчитывается, по формуле (3.18) с первым порядком точности в крайних точках интервала, и со вторым порядком точности в средней точке интервала [1].

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

$$y(x) \approx \varphi(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + \frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}(x - x_i)(x - x_{i+1}),\ x \in \left[x_i, x_{i+1}\right] \tag{3.19}$$

$$y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{\frac{y_{i+2} - y_{i-1}}{x_{i+2} - x_{i-1}} - \frac{y_{i-1} - y_i}{x_{i-1} - x_i}}{x_{i-2} - x_i}(2x - x_i - x_{i-1}),\ x \in \left[x_i, x_{i+1}\right] \tag{3.20}$$

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

3.4. 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 $x_0 \ldots x_N$, так что $a = x_0 \le x_1 \le \ldots \le x_N = b$ 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 $\varphi(x)$, так что она, во-первых, близка $f(x)$, а, во-вторых, интеграл от $\varphi(x)$ 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 $\varphi(x_j) = f(x_j),\ j = i,\ldots,i+k$, т.е. многочлен P_n зависит от параметров a_j - $P_n(x,\bar{a}_i)$, тогда

$$f(x) = P_n(x,\bar{a}_i) + R_n(x,\bar{a}_i), \quad x \in [x_i, x_{i-k}], \tag{3.22}$$

где $R_n(x,\bar{a}_i)$ – остаточный член интерполяции. Тогда $F = \sum_{i=1}^{N} \int_{x_{i-1}}^{x_i} P_n(x,\bar{a}_i)dx + R$,

где $R = \sum_{i\ 1}^{N} \int_{x_{i-1}}^{x_i} R_n(x, \overline{a}_i)\, dx$ – 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 $\overline{x}_i = (x_{i-1} + x_i)/2$, получим формулу прямоугольников.

$$\int_a^b f(x)\, dx \approx \sum_{i\ 1}^{N} h_i\, f\left(\frac{x_{i\ 1} + x_i}{2}\right) \tag{3.23}$$

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

$$R \le \frac{1}{24} h^2 M_2 (b-a), \tag{3.24}$$

где $M_2 = \max \left| f''(x) \right|_{[a.b]}$.

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.

$$F = \int_a^b f(x)\, dx \approx \frac{1}{2} \sum_{i=1}^{N} (f_i + f_{i-1}) h_i \tag{3.25}$$

Эта формула носит название формулы трапеций.

В случае постоянного шага интегрирования величина остаточного члена оценивается

$$R \le \frac{b-a}{12} h^2 M_2, \tag{3.26}$$

где $M_2 = \max \left| f''(x) \right|_{[a,b]}$.

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)

$$F = \int_a^b f(x)\,dx \approx \frac{1}{3} \sum_{i=1}^{N} (f_{i-1} + 4 f_{i-\frac{1}{2}} + f_i) h_i \tag{3.27}$$

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

$$F \approx \frac{h}{3} \left[f_0 + 4 f_{\frac{1}{2}} + 2 f_1 + 4 f_{\frac{3}{2}} + 2 f_2 + ... + 2 f_{N-1} + 4 f_{N-\frac{1}{2}} + f_N \right], \tag{3.28}$$

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

$$R \le \frac{(b-a)}{180} h^4 M_4, \tag{3.29}$$

где $M_4 = \max \left| f^{IV}(x) \right|_{[a,b]}$.

Исходный код на языке Python

1 Интерполяция многочленом

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy
from functools import reduce

Xi = [0.1*pi, 0.2*pi, 0.3*pi, 0.4*pi]
Yi = [sin(xx) for xx in Xi]

Y1_global = []

#Xi = [0.0, 1.0, 2.0, 3.0]
#Yi = [round(sin(xx*pi/6), 5) for xx in Xi]

X0 = pi/4
Y0 = sin(X0)

def product(L):
     mul = lambda x, y: x*y
     return reduce(mul, L)


def l_i(x, i, X):
     """ OK """
     n = len(X)
     S = 1
     for j in range(n):
          if (i!=j):
               S *= (x - X[j])/(X[i] - X[j])
     return S

def L_n(x, Y, X):
     return sum( [(Y[i]*l_i(x, i, X)) for i in range(len(Y))] )


def L(Y, X):
     return lambda x: L_n(x, Y, X)

def Y_m(Y, X):
     m = len(Y)
     if   (m == 1):
          return Y[0]
     elif (m == 2):
          return(Y[0] - Y[1])/(X[0] - X[1])
     else:
          return(Y_m(Y[0:m-1],X[0:m-1])-Y_m(Y[1:m],X[1:m]) )/(X[0]-X[m-1])

def P_m(X):
     if (len(X) == 0): return lambda x: 1
     return lambda x:product( [(x - x_i) for x_i in X])

def N_1(x, Y1, P1):
     return sum( [(Y*P(x)) for (Y,P) in zip(Y1,P1)] )

def N(Y, X):
     m = len(X)
     Y1 = [Y_m(Y[0:i],X[0:i]) for i in range(1,m+1)]
     P1 = [P_m(X[0:i])        for i in range(0,m)]
```

```
     print("polynom koef:")
     for yy in Y1:  print(round(yy, 5))

     return lambda x: N_1(x,Y1,P1)

def main():
     print("table function: ")
     print(Xi)
     print(Yi)

     lagranje = L(Yi, Xi)
     newton   = N(Yi, Xi)

     print("Test lagranje", Yi[2], "=", lagranje(Xi[2]) )
     print("Test newton  ", Yi[2], "=", newton  (Xi[2]) )

     eps_n = abs( Y0 - lagranje(X0) )
     print( "|sin(x) - P_n(x)| = ", eps_n)

     return 0

main()
```

2 Интерполяция сплайнами

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy
from functools import reduce

class Tridiagonal_Matrix:
     def __init__(self):
               self.a = []
               self.b = []
               self.c = []
               self.d = []
               self.n = 0

     def solve(self):
          """Method progonki"""

          a = self.a
          b = self.b
          c = self.c
          d = self.d
          n = len(d)

          P = []
          Q = []
          P.append(-c[0]/b[0])
          Q.append( d[0]/b[0])

          for i in range(1, n):
               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]) )

          x = [0]*n
          x[n-1] = Q[n-1]
          for i in range(n-2, -1, -1):
```

```
                x[i] = P[i]*x[i+1] + Q[i]

          return x

Xi = [0.0, 1.0, 2.0, 3.0, 4.0]
#Yi = [0.0, 0.5, 0.86603, 1.0, 0.86603]
Yi = [0.0, 1.8415, 2.9093, 3.1411, 3.2432]

Hi = [0.0] + [(Xi[i] - Xi[i-1]) for i in range(1,len(Xi))]

X0 = 1.5
Y0 = 2.4969

def roundx(v):
     return [round(xx, 5) for xx in v]


def build_ci(Y, X, H):
     n = len(X)
     M = Tridiagonal_Matrix()

     for i in range(2, n): # numerating from zero
          a = H[i-1]
          b = 2*(H[i-1]+H[i])
          c = H[i]
          d = 3*( (Y[i]-Y[i-1])/H[i] - (Y[i-1]-Y[i-2])/H[i-1] )
          if (i==2)  : a = 0
          if (i==n-1): c = 0
          M.a.append(a)
          M.b.append(b)
          M.c.append(c)
          M.d.append(d)

     print("Tridiagonal_Matrix:")
     print(roundx(M.a))
     print(roundx(M.b))
     print(roundx(M.c))
     print(roundx(M.d))

     x = [0.0] + M.solve()

     print ("x = ", roundx(x) )

     return x


def spline1(A, B, C, D, X, x):
     """ calculate spline"""
     if (x < X[0]) : return 0.0
     if (x > X[-1]) : return 0.0

     i = 0
     while (X[i+1] < x): i+=1
     # segment X[i]..X[i+1]
     x1 = X[i]
     return A[i] + B[i]*(x-x1) + C[i]*(x-x1)**2 + D[i]*(x-x1)**3

def cr_spline(A, B, C, D, X):
     """ create spline"""
     return lambda x: spline1(A, B, C, D, X, x)
```

```
def main():
     print("table function: ")
     print(Xi)
     print(Yi)

     Ai = Yi[0 : -1]
     Ci = build_ci(Yi, Xi, Hi)

     n = len(Ci)-1
     Bi = [( (Yi[i+1]-Yi[i])/Hi[i+1] - Hi[i+1]*(Ci[i+1]+2*Ci[i])/3 ) for i in
range(0,n)]
     Bi.append( (Yi[n+1]-Yi[n])/Hi[n] - Hi[n]*Ci[n]*2/3 )

     Di = [( (Ci[i+1]-Ci[i])/(3*Hi[i+1]) ) for i in range(0,n)]
     Di.append( -Ci[n]/(3*Hi[n]) )

     print("proverka:")
     print("A = ", roundx(Ai))
     print("B = ", roundx(Bi))
     print("C = ", roundx(Ci))
     print("D = ", roundx(Di))

     sp = cr_spline(Ai, Bi, Ci, Di, Xi)

     print("\ntest1 ", Y0,"=", sp(X0) )

     return 0

main()
```

3 Метод наименьших квадратов

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy
from functools import reduce
from matrix_2 import *


#Xi = [-1.0, 0.0, 1.0, 2.0    , 3.0, 4.0    ]
#Yi = [-0.5, 0.0, 0.5, 0.86603, 1.0, 0.86603]

Xi = [0.0, 1.7   , 3.4   , 5.1   , 6.8   , 8.5   ]
Yi = [0.0, 1.3038, 1.8439, 2.2583, 2.6077, 2.9155]

X0 = 1.5
Y0 = 2.4969

def roundx(v):
     return [round(xx, 4) for xx in v]

def P_n(A):
     return lambda x: sum([ A[i]*x**i for i in range(0,len(A)) ] )


def cr_MNK(Y, X, m = 2):
     n = m + 1

     A = Matrix("", n, n)
     A.b = [sum(Y)] + [ sum([(y_j * x_j**k) for (y_j, x_j) in zip(Y, X)])
```

```
                                                                  for      k      in
range(1,n)    ]

     A.M = [ [sum([ x_j**(i+k) for x_j in X ])
                                          for k in range(n) ]
                                               for i in range(n) ]

#     A.M[0][0] = len(X) #~ does not matter
     A.pr()

     print("b = ", roundx(A.b) )

     A.n = len(A.M)
     A.m = len(A.M)
     A.build_LU()

     print("p = ", A.p)

     Ai = A.solve(A.shift_b(A.b))

     print("Ai = ", roundx(Ai) )
     return Ai

def F_eps(f, Y, X):
     return sum( [(f(xx)-yy)**2 for (xx,yy) in zip(X, Y)] )


def main():
     print("table function: ")
     print(Xi)
     print(Yi)


     Ai = cr_MNK(Yi, Xi, 2)
     polynom = P_n(Ai)

     print("\ntest1 ", Yi[1],"=", polynom (Xi[0]) )
     print("F_bolshoe     =", F_eps(polynom, Yi, Xi) )

     return 0

main()
```

4 Численное дифференцирование

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy
from functools import reduce
from matrix_2 import *


#Xi = [-1.0, 0.0, 1.0, 2.0    , 3.0, 4.0    ]
#Yi = [-0.5, 0.0, 0.5, 0.86603, 1.0, 0.86603]

Xi = [0.0, 0.1   , 0.2   , 0.3   , 0.4   ]
Yi = [1.0, 1.1052, 1.2214, 1.3499, 1.4918]

X0 = 0.2
Y0 = 2.4969
```

```
def roundx(v):
     return [round(xx, 4) for xx in v]

def P_n(A):
     return lambda x: sum([ A[i]*x**i for i in range(0,len(A)) ] )

def Differetiation( Y, X, n, x):
     if (x<X[0])  : return None
     if (x>=X[-2]-0.001) : return None
     i = 0
     while (x > X[i]+0.0001) : i+=1
     i-=1
     print("segment", [X[i], X[i+1] ], ", x =", x )

     if   (n==1):
          return  (Y[i+1]-Y[i])/(X[i+1]-X[i])  +  (  ((Y[i+2]-Y[i+1])/(X[i+2]-
X[i+1]) - (Y[i+1]-Y[i])/(X[i+1]-X[i]))/(X[i+2]-X[i]) ) * (2*x-X[i]-X[i+1])
     elif (n == 2):
          return 2*(((Y[i+2] - Y[i+1]) / (X[i+2] - X[i+1]) - (Y[i+1] - Y[i])
/ (X[i+1] - X[i])) / (X[i+2] - X[i]));
     else:
          return None



def diff1(Y, X, n):
     return lambda x: Differetiation( Y, X, n, x)

def main():
     print("table function: ")
     print(Xi)
     print(Yi)

     d1 = diff1(Yi, Xi, 1)
     d2 = diff1(Yi, Xi, 2)

     print("diff1", d1(X0) )
     print("diff1", d2(X0) )

     return 0

main()
```

5 Численное интегрирование

```
#! /usr/bin/python3.1
# -*- coding: utf-8 -*-

from math import sqrt,sin,cos,atan,pi,log
from copy import copy, deepcopy

segment = [-1.0, 1.0]

def func(x):
     return x/(3*x + 4)**2

def func2(x):
     return x/(2*x + 5)

def roundx(v):
     return [round(xx, 4) for xx in v]
```

```
def P_n(A):
     return lambda x: sum([ A[i]*x**i for i in range(0,len(A)) ] )

def MethodOfRectangles(f, seg, h):
     n = round((seg[1] - seg[0])/h )
     return sum( [ h * f( seg[0] + h*i + h/2) for i in range(n)] )

def MethodOfTrapezoids(f, seg, h):
     n = round((seg[1] - seg[0])/h )
     y = [ f(seg[0] + h*i) for i in range(n+1)]
     return sum( [ (y[i] + y[i+1])*h/2  for i in range(n)] )

def SimpsonMethod(f, seg, h):
     n = round((seg[1] - seg[0])/h )
     n2 = round((seg[1] - seg[0])/h)//2
     y = [ f(seg[0] + h*i) for i in range(n+1)]
     return sum( [( (y[2*i] + 4*y[2*i+1]+y[2*(i+1)] )*h/3 )
                                                    for i in range(n2)] )

def RRR(f1, f2, k, p):
     """ Runge-Romberga-Richardsona """
     # k = h1/h2, p - interpolation level
     return f1 + (f1 - f2) / (k**p - 1)

def main():
     r1 = MethodOfRectangles(func, segment, 0.5)
     print("Rect 0.5:\t", r1)
     t1 = MethodOfTrapezoids(func, segment, 0.5)
     print("Trap 0.5:\t", t1)
     s1 = SimpsonMethod (func, segment, 0.5)
     print("Simspson 0.5:\t", s1)
     r2 = MethodOfRectangles(func, segment, 0.25)
     print("Rect 0.25:\t", r2)
     t2 = MethodOfTrapezoids(func, segment, 0.25)
     print("Trap 0.25:\t", t2)
     s2 = SimpsonMethod (func, segment, 0.25)
     print("Simspson 0.25:\t", s2)

     print("RRR of Rect:\t", RRR(r2, r1, 2, 2) )
     print("RRR of Trap:\t", RRR(t2, t1, 2, 2) )
     print("RRR of Simpson:\t", RRR(s2, s1, 2, 2) )
     print("answer\t", SimpsonMethod(func, segment, 0.0001) )
     return 0
main()
```

6 Построение графика табличной функции

```
#! /usr/bin/python2.6
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt,sin,cos,atan,pi,log

func = lambda x: cos(x)
Xi = [0.1*pi, 0.2*pi, 0.3*pi, 0.4*pi]
Yi = [sin(xx) for xx in Xi]

X = np.arange(Xi[0], Xi[-1], 0.01)
Y = [ func(xx) for xx in X]
plt.plot(X,Y,color="red")
plt.plot(Xi,Yi,color="grey")
plt.show()
```

Протокол тестирования

1 Интерполяция многочленом

```
oleg@debian:~/lab3_release$ ./lab3_1.py
table function:
[0.3141592653589793,0.6283185307179586,0.9424777960769379,1.2566370614359172]
[0.3090169943749474,0.5877852522924731,0.8090169943749475,0.9510565162951535]
polynom koef:
0.30902
0.88735
-0.29148
-0.1164
Test lagranje 0.809016994375 = 0.809016994375
Test newton   0.809016994375 = 0.809016994375
|sin(x) - P_n(x)| = 0.000160111853005
```

2 Интерполяция сплайнами

```
oleg@debian:~/lab3_release$ ./lab3_2.py
table function:
[0.0, 1.0, 2.0, 3.0, 4.0]
[0.0, 1.8415, 2.9093, 3.1411, 3.2432]
Tridiagonal_Matrix:
[0, 1.0, 1.0]
[4.0, 4.0, 4.0]
[1.0, 1.0, 0]
[-2.3211, -2.508, -0.3891]
x =  [0.0, -0.44953, -0.52299, 0.03347]
proverka:
A =  [0.0, 1.8415, 2.9093, 3.1411]
B =  [1.99134, 1.54181, 0.5693, 0.07979]
C =  [0.0, -0.44953, -0.52299, 0.03347]
D =  [-0.14984, -0.02449, 0.18549, -0.01116]

test1  2.4969 = 2.49696428571
```

3 Метод наименьших квадратов

```
oleg@debian:~/lab3_release$ ./lab3_3.py
table function:
[0.0, 1.7, 3.4, 5.1, 6.8, 8.5]
[0.0, 1.3038, 1.8439, 2.2583, 2.6077, 2.9155]
Matrix 3x3 :
    6.0     25.5      158.9
    25.5    158.9     1105.4
    158.9   1105.4    8176.7
-----
b =  [10.9292, 62.5172, 415.0468]
p =  [2, 1, 0]
Ai =  [0.1294, 0.6193, -0.0355]

test1  1.3038 = 0.129442857143
F_bolshoe     = 0.0945576948571
```

4 Численное дифференцирование

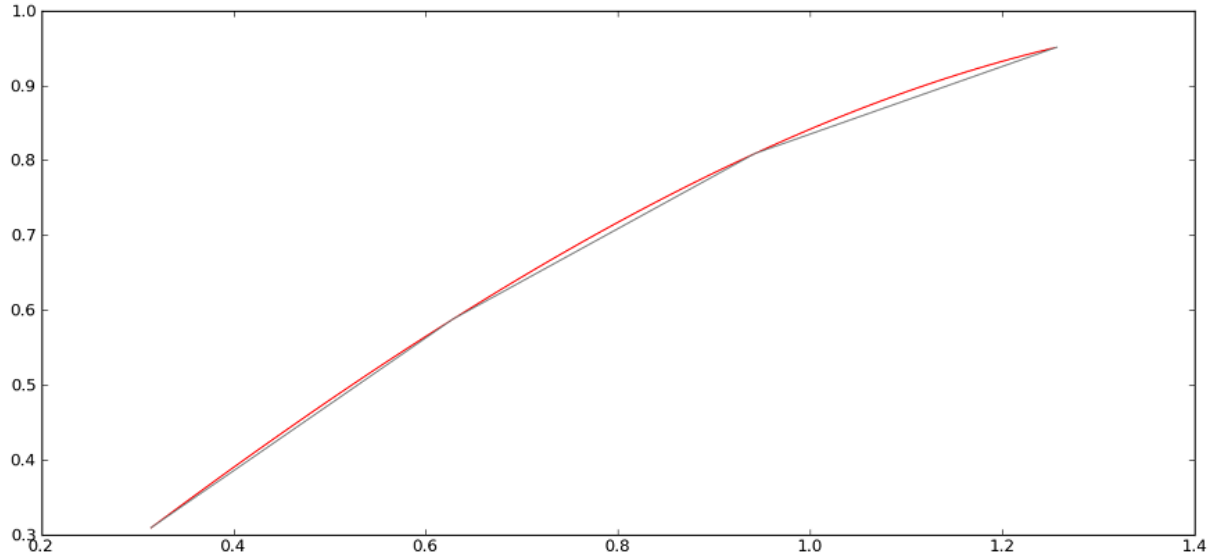
```
oleg@debian:~/lab3_release$ ./lab3_4.py
table function:
[0.0, 0.1, 0.2, 0.3, 0.4]
[1.0, 1.1052, 1.2214, 1.3499, 1.4918]
segment [0.1, 0.2] , x = 0.2
diff1 1.2235
segment [0.1, 0.2] , x = 0.2
diff1 1.23
```

5 Численное интегрирование

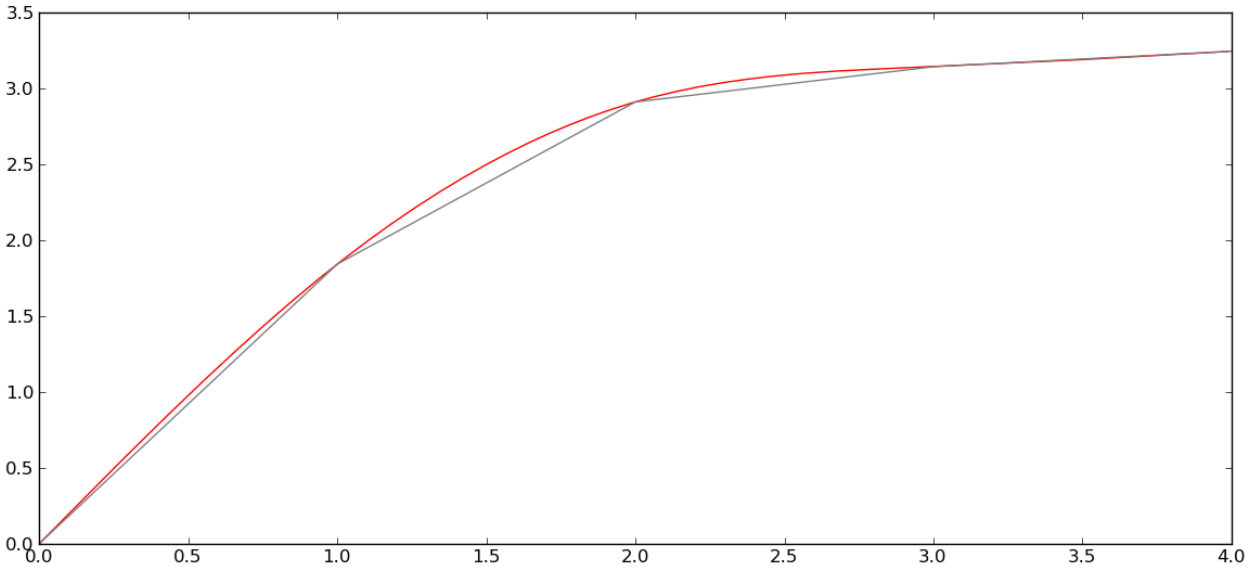
```
oleg@debian:~/lab3_release$ ./lab3_5.py
Rect 0.5:                           -0.119143132913
Trap 0.5:                           -0.276633496374
Simspson 0.5:                       -0.205579355709
Rect 0.25:                          -0.149311959381
Trap 0.25:                          -0.197888314644
Simspson 0.25:                      -0.171639920734
RRR of Rect:                        -0.15936823487
RRR of Trap:                        -0.171639920734
RRR of Simpson:                     -0.160326775742
answer                              -0.164740142168
```

Графики

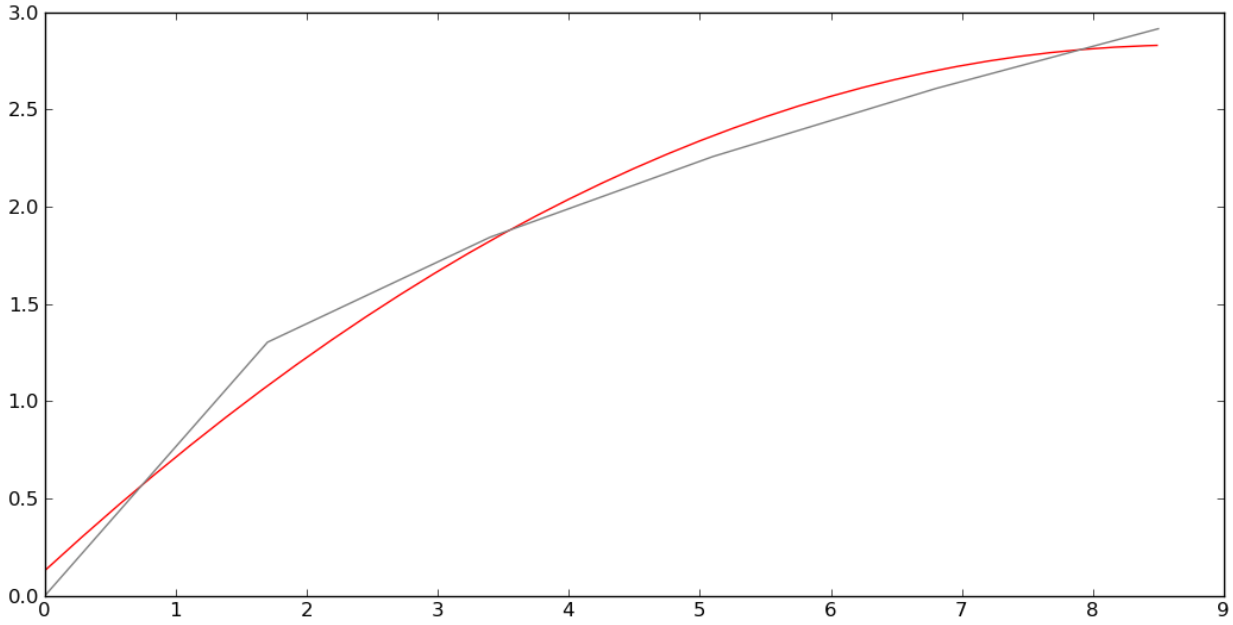
1 Интерполяция многочленом



2 Интерполяция сплайнами



3 Метод наименьших квадратов



Выводы

1 Интерполяция многочленом

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^*.

2 Интерполяция сплайнами

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^*.

3 Метод наименьших квадратов

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.

4 Численное дифференцирование

Вычислить первую и вторую производную от таблично заданной функции в точке X^*.

5 Численное интегрирование

Вычислить определенный интеграл методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.