БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Методы интерполяции

Выполнил: Аквух Джеймс
2 курс 3 группа
Преподаватель: Будник А.М.

Минск 2016

1. Постановка задачи

Для некоторой функции $f(x)$ известны значения в некоторых точках x_i . Необходимо найти значения данной функции в других точках, принадлежащих заданному отрезку.

2. Методы интерполяции

A. метод Лагранжа

Многочлен:

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

Погрешность:

$$|f(x) - P_n(x)| \leq \frac{M}{(n+1)!}|\omega(x)|$$

$$\forall x \in [a, b] \Rightarrow |f^{(n+1)}(x)| \leq M = const$$

B. метод Ньютона

Многочлен:

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_1) + (x - x_0)(x - x_1)f(x_0, x_1, x_2) +$$

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

Погрешность:

$$|f(x) - P_n(x)| = \omega(x)f(x, x_0, \dots, x_n),$$

$$\omega(x) = (x - x_0) * \dots * (x - x_n).$$

C. метод Лагранжа на узлах, найденных с помощью метода Чебышева

Многочлен:

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

Узлы интерполирования:

$$x_m = \frac{a+b}{2} + \frac{b-a}{2} cos\left(\frac{\pi(m + 1/2)}{n}\right), m = \overline{0, n-1}$$

Погрешность:

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

D. метод Эрмита

Многочлен:

$$P_n(x) = f(x_0) + (x - x_0)f(x_0, x_0) + (x - x_0)^2 f(x_0, x_0, x_1) +$$
$$+ (x - x_0)^2 \dots (x - x_n) f(x_0, \dots, x_n),$$

Погрешность:

$$\frac{f^{(K)}(c)}{K!} \prod (x - x_i)^{k_i},$$

где K - общее число данных значений для интерполирования, k_i - число производных в точке x_i плюс 1.

E. метод наименьших квадратов

Многочлен:

$$\phi(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_m x^m$$

$$\sum_{i=0}^{m} \alpha_i (g_i, g_j) = (f, g_j), j = \overline{0, m}$$

$$g_i(x) = x^i$$

F. интерполирование кубическим сплайном

Многочлен:

$$s_i(x) = M_i \frac{(x_{i+1} - x)^3}{6h} + M_{i+1} \frac{(x - x_i)^3}{6h} + C_i(x - x_i) + B_i,$$

$$C_i = \frac{f_{i+1} - f_i}{h} - \frac{h}{6}(M_{i+1} - M_i), \quad B_i = f_i - \frac{h_i^2}{6} M_i, \quad h = (b - a)/n$$

$$\begin{cases} 2M_0 + \alpha_0 M_1 = d_0 \\ \phi_1 M_0 + 2M_1 + \alpha_1 M_2 = d_1 \\ \dots \\ \phi_{n-1} M_{n-2} + 2M_{n-1} + \alpha_{n-1} M_n = d_{n-1} \\ \phi_n M_{n-1} + 2M_n = d_n \end{cases}$$

$$\alpha_i = \phi_i = 1/2, d_i = 3/2 * (f_{i+1} - 2f_i + f_{i-1}), i = \overline{1, n-1};$$
$$\alpha_0 = d_0 = \phi_n = d_n = 0.$$

3. Исходные данные:

$f(x) = 0.9 * e^x + 0.1 * e^{-x}$
$x \in [0, 2]$
$n = 5,\ 10$

4. Листинг

```
require('console.table')
const {sum, prod, factorial, subtract, mean, lusolve} = require('mathjs')
const {take, takeRight, isArray, last, first, ary, uniq, initia} = require('lodash')
const {exp, abs, floor, cos, pow, PI} = Math
const {log, table} = console

const a = 0, b = 2
const M = 0.1 * exp(2) + 0.9

const w = v => x => prod(v.map(vi => x - vi))
const wd = v => (x, i) => w(v.filter((_, ind) => ind !== i))(x)
const range = (a, b, n = 1) => [...Array(n + 1).keys()].map(i => a + (b - a) / n * i)
const R = v => x => M / factorial(v.length) * abs(w(v)(x))
const f = x => 0.1 * exp(x) + 0.9 * exp(-x)
const fd = x => 0.1 * exp(x) - 0.9 * exp(-x)
const array = n => [...Array(n).keys()]

const diffFactory = x => function diff(...v) {
        if (uniq(v).length === 1) {
                if (v.length === 1) return f(x[v[0]])
                else return fd(x[v[0]])
        } else {
                return (diff(...v.slice(1)) - diff(...v.slice(0, v.length - 1))) / (x[last(v)] - x[first(v)])
        }
}

const print = ({name, n, x, p, Rn, fn, r}) => {
        log(`${name} for n = ${n}`)
        table([,
                ['x', ...x],
                ['f(x)', ...fn],
                ['p(x)', ...p],
                ['|p(x) - f(x)|', ...r],
                ['R(x, n)', ...Rn]
        ])
}

const lagrange = v => x => {
        return w(v)(x) * sum(v.map((xi, i) => f(xi) / (x - xi) / wd(v)(xi, i)))
}
const lagrangeTest = n => {
        const v = range(a, b, n)
        const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
        const p = x.map(lagrange(v))
        const fn = x.map(ary(f, 1))
        const r = subtract(p, x.map(ary(f, 1))).map(abs)
        const Rn = x.map(R(v))
        print({name: 'Lagrange', n, x, p, Rn, fn, r})
}

const newton = v => x => {
        const diff = diffFactory(v)
        return f(v[0]) + sum([...Array(v.length - 1).keys()].map(i => {
                return prod(take(v, i + 1).map(xi => x - xi)) * diff(...[...Array(i + 2).keys()])
        }))
}
const newtonTest = n => {
        const v = range(a, b, n)
        const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
        const p = x.map(newton(v))
        const fn = x.map(ary(f, 1))
        const r = subtract(p, x.map(ary(f, 1))).map(abs)
        const Rn = x.map(R(v))
        print({name: 'Newton', n, x, p, Rn, fn, r})
}
```

```
const chebyshevTest = n => {
	const v = [...Array(n + 1).keys()].map(i => mean(a, b) + mean(b, -a) * cos(PI * (i + 1 / 2) / (n + 1)))
	const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
	const p = x.map(lagrange(v))
	const fn = x.map(ary(f, 1))
	const r = subtract(p, x.map(ary(f, 1))).map(abs)
	const Rn = x.map(() => M / factorial(n + 1) * pow(b - a, n) * pow(2, 1 - 2 * n))
	print({name: 'Chebyshev, Lagrange', n, x, p, Rn, fn, r})
}

const ermit = (z, v) => x => {
	const diff = diffFactory(v)
	return f(v[z[0]]) + sum([...Array(z.length - 1).keys()].map(i => {
		return prod(take(z, i + 1).map(zi => x - v[zi])) * diff(...take(z, i + 2))
	}))
}
const ermitTest = n => {
	const v = range(a, b, n)
	const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
	const z = [...Array(2 * (n + 1)).keys()].map(i => floor(i / 2))
	const p = x.map(ermit(z, v))
	const fn = x.map(ary(f, 1))
	const r = subtract(p, x.map(ary(f, 1))).map(abs)
	const Rn = x.map(R(v))
	print({name: 'Ermit', n, x, p, Rn, fn, r})
}

const lsm = (v, m) => x => {
	const n = v.length - 1
	const phi = i => x => prod(array(i).map(k => x).concat(1))
	const A = array(m + 1).map(i => array(n + 1).map(j =>
		sum(array(n + 1).map(k => phi(i)(v[k]) * phi(j)(v[k])))))
	const B = array(m + 1).map(i => sum(array(n + 1).map(k => f(v[k]) * phi(i)(v[k]))))
	const a = lusolve(A, B)
	return sum(a.map((c, i) => c[0] * phi(i)(x)))
}
const lsmTest = n => {
	const v = range(a, b, n)
	const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
	const p = x.map(lsm(v, n))
	const fn = x.map(ary(f, 1))
	const r = subtract(p, x.map(ary(f, 1))).map(abs)
	const Rn = x.map(R(v))
	print({name: 'LSM', n, x, p, Rn, fn, r})
}

const spline = (v, h) => x => {
	const i = v.findIndex((_, i) => v[i] <= x && x <= v[i + 1])
	const n = v.length - 1

	let A = array(n + 1).map(() => array(n + 1))
	A[0][0] = A[n][n] = 2
	A[0][1] = A[n][n - 1] = 0
	array(n - 1).map(i => {
		A[i + 1][i] = A[i + 1][i + 2] = 1 / 2
		A[i + 1][i + 1] = 2
	})
	let D = array(n + 1)
	D[0] = D[n] = 0
	array(n - 1).map(i => {
		D[i + 1] = 3 / 2 * (f(v[i + 1]) - 2 * f(v[i]) + f(v[i - 1]))
	})
	const M = lusolve(A, D).map(v => v[0])

	const B = f(v[i]) - h * h / 6 * M[i]
	const C = (f(v[i + 1]) - f(v[i])) / h - h / 6 * (M[i + 1] - M[i])
	return (M[i] * pow(v[i + 1] - x, 3) + M[i + 1] * pow(x - v[i], 3)) / 6 / h + C * (x - v[i]) + B
}
```

```
const splineTest = n => {
	const v = range(a, b, n)
	const h = (b - a) / n
	const x = [mean(take(v, 2)), mean(take(v.slice(floor(n / 2)), 2)), mean(takeRight(v, 2))]
	const p = x.map(spline(v, h))
	const fn = x.map(ary(f, 1))
	const r = subtract(p, x.map(ary(f, 1))).map(abs)
	const Rn = x.map(R(v))
	print({name: 'Spline', n, x, p, Rn, fn, r})
}

lagrangeTest(5)
lagrangeTest(10)
newtonTest(5)
newtonTest(10)
chebyshevTest(5)
chebyshevTest(10)
ermitTest(3)
ermitTest(5)
lsmTest(5)
lsmTest(10)
splineTest(5)
splineTest(10)
```

5. Результаты

Lagrange for n = 5

0	1	2	3
x	0.2	1	1.8
f(x)	0.8589979535862005	0.6029196799002027	0.7537337458407226
p(x)	0.8590510579402804	0.6029320434806754	0.7537856796871295
|p(x) - f(x)|	0.0000531043540799025 5	0.00001236358047274066 5	0.00005193384640689302 5
R(x, n)	0.00013766807123101758	0.00003277811219786132	0.00013766807123101737

Lagrange for n = 10

0	1	2	3
x	0.1	1.1	1.9
f(x)	0.9248707680399283	0.6000005777229149	0.8032012015282987
p(x)	0.9248707680573781	0.6000005777227796	0.8032012015259311
|p(x) - f(x)|	1.7449819367243435e-11	1.3533618670180658e-13	2.3675506000131463e-12
R(x, n)	2.688189318726947e-10	4.033244274346223e-12	2.68818931872694e-10

Newton for n = 5

0	1	2	3
x	0.2	1	1.8
f(x)	0.8589979535862005	0.6029196799002027	0.7537337458407226
p(x)	0.8590510579402806	0.6029320434806753	0.7537856796871296
|p(x) - f(x)|	0.00005310435408012459 4	0.00001236358047262964 3	0.00005193384640700405
R(x, n)	0.00013766807123101758	0.00003277811219786132	0.00013766807123101737

Newton for n = 10

0	1	2	3
x	0.1	1.1	1.9
f(x)	0.9248707680399283	0.6000005777229149	0.8032012015282987
p(x)	0.924870768057379	0.6000005777227795	0.8032012015259298
|p(x) - f(x)|	1.7450707545663136e-11	1.354472090042691e-13	2.3688828676426965e-12
R(x, n)	2.688189318726947e-10	4.033244274346223e-12	2.68818931872694e-10

Chebyshev, Lagrange for n = 5

0	1	2	3
x	1.8365163037378078	1	0.16348369626219217
f(x)	0.7708986161269546	0.6029196799002027	0.8820229248970497
p(x)	0.7709239922632937	0.6029465580577821	0.8820488990115263
|p(x) - f(x)|	0.00002537613633901703	0.00002687815757940193	0.00002597411447657904 7
R(x, n)	0.00014226611196988413	0.00014226611196988413	0.00014226611196988413

Chebyshev, Lagrange for n = 10

0	1	2	3
x	1.9497267186177256	0.8591337215792852	0.05027328138227455
f(x)	0.8307583653360365	0.617287378094183	0.9610283988985046
p(x)	0.8307583653357625	0.6172873780958346	0.961028398901007
|p(x) - f(x)|	2.7400304247748863e-13	1.6515677714323829e-12	2.502442697505103e-12
R(x, n)	8.01914862745108e-11	8.01914862745108e-11	8.01914862745108e-11

Ermit for n = 3

0	1	2	3
x	0.3333333333333333	1	1.6666666666666665
f(x)	0.7844394220250193	0.6029196799002027	0.6994370476008085
p(x)	0.7844388899984687	0.6029194906779913	0.6994365232248136
|p(x) - f(x)|	5.320265505925903e-7	1.892222113442088e-7	5.243759948481141e-7
R(x, n)	0.012645876619545254	0.007587525971727153	0.012645876619545257

Ermit for n = 5

0	1	2	3
x	0.2	1	1.8
f(x)	0.8589979535862005	0.6029196799002027	0.7537337458407226
p(x)	0.8589979535814805	0.6029196798999377	0.7537337458360589
|p(x) - f(x)|	4.7200021668913905e-12	2.65010235978024 87e-13	4.663713859542895e-12
R(x, n)	0.00013766807123101758	0.00003277811219786132	0.00013766807123101737

LSM for n = 5

0	1	2	3
x	0.2	1	1.8
f(x)	0.8589979535862005	0.6029196799002027	0.7537337458407226
p(x)	0.8590510579396596	0.6029320434806915	0.7537856796877127
|p(x) - f(x)|	0.00005310435345917 6856	0.00001236358048883 8899	0.00005193384699009318
R(x, n)	0.00013766807123101758	0.00003277811219786132	0.00013766807123101737

LSM for n = 10

0	1	2	3
x	0.1	1.1	1.9
f(x)	0.9248707680399283	0.6000005777229149	0.8032012015282987
p(x)	0.9248706482850594	0.6000005748122748	0.8032010875597239
|p(x) - f(x)|	1.19754868910249e-7	2.9106401733258735e-9	1.1396857479972766e-7
R(x, n)	2.688189318726947e-10	4.033244274346223e-12	2.68818931872694e-10

Spline for n = 5

0	1	2	3
x	0.2	1	1.8
f(x)	0.8589979535862005	0.6029196799002027	0.7537337458407226
p(x)	0.8733686076783594	0.6156369685379494	0.7689682211956317
|p(x) - f(x)|	0.014370654092158985	0.012717288637746682	0.015234475354909072
R(x, n)	0.00013766807123101758	0.00003277811219786132	0.00013766807123101737

Spline for n = 10

0	1	2	3
x	0.1	1.1	1.9
f(x)	0.9248707680399283	0.6000005777229149	0.8032012015282987
p(x)	0.9299665085218027	0.6030328443987073	0.8071950774591228
|p(x) - f(x)|	0.005095740481874 4445	0.003032266675792372	0.00399387593082412
R(x, n)	2.688189318726947e-10	4.033244274346223e-12	2.68818931872694e-10

6. Вывод

Погрешности метода Лагранжа и метода Ньютона практически в точности совпадают. Это объясняется тем, что оба метода строят один и тот же полином и отличаются лишь способами его построения. Поскольку погрешность обусловлена только машинной ошибкой, а ее абсолютное значение имеет порядок намного больший, чем машинный эпсилон (ср. 10^{-4} и 10^{-15}), то первые несколько значащих цифр полностью совпадают.

При увеличении числа узлов интерполяции в 2 раза, точность метода Лагранжа и метода Ньютона увеличивается на 6 порядков. Это объясняется тем, что погрешность методов имеет факториальную зависимость от числа узлов интерполяции.

Рассмотренные выше метода строят полином, который в точности проходит через все узлы интерполирования. В то же время для интерполяции можно минимизироват и другие функционалы.

Метод Чебышева используется для определения узлов интерполирования, при которых максимальная погрешность на отрезке $[-1,1]$ будет наименьшей среди всех полиномов той же степени. Как видно из результатов, это обеспечивает более точное приближение функции по сравнения с методами Лагранжа и Ньютона. Тем не менее для использования метода Чебышева необходимо знать значения функции в точках, которые являются корнями полинома Чебышева. На практике эти значение получить не всегда возможно, что ограничивает использование метода Чебышева.

Эрмитова интерполяция используется, когда кроме значений самой функции известны также значения ее производных. Как и в методе Ньютона, при эрмитовой интерполяции значения построенного полинома и его производных в точности совпадают с заданными интерполирующими значениями.

Метод наименьших квадратов минимизирует функционал общего квадратичного отклонения интерполяционных значений от построенного полинома. Это означает, что построенная функция может не совпадать с интерполирующими значениями в узлах. Несмотря на то, что погрешность метода наименьших квадратов больше соответствующей погрешности рассмотренных выше методов, у этого метода есть существенный плюс: в качестве базиса можно выбрать любые линейно независимые функции. Это означает, что если перед интерполяций имеется предварительная информация о виде функций (к примеру, ее тригонометрический характер - исходя из физических соображений задачи), то можно подобрать подходящий базис тем самым существенно повысив точность метода.

Идея метода интерполяции сплайном заключается в разбиении всего отрезка интерполирования на несколько и приближении каждого из них полиномами заданной степени.