Санкт-Петербургский государственный университет
Факультет прикладной математики - процессов управления
Кафедра математической теории игр и статистических решений

Численные методы: интерполирование функций

Сардарян Армен, группа 301
5 семестр, 2019 год

1 Полином Лагранжа

Пусть на отрезке $[a, b] \subset \mathbb{R}$ рассматривается функция $f(x)$ и известны её значения в $(n+1)$ различных узлах $x_0, x_1, \ldots, x_n$, принадлежащих $[a, b]$. Искомый интерполяционный полином может быть представлен в виде:

$$L_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i), \tag{1}$$

где $l_i(x) = \frac{(x-x_o)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x_k-x_n)}{(x_k-x_o)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$. Многочлены $l_i(x)$ называют множителями Лагранжа, а формулу (1) формулой Лагранжа для интерполяционного многочлена $L_n(x)$.

1.1 Полином Лагранжа с равноотстоящими узлами интерполирования

Рассмотрим функцию $f(x) = x - sin(x) - 0.25$ на отрезке $[-15, 15]$ с 6 равноотстоящими узлами интерполирования.

1.2 Выбор узлов интерполяции

Рассмотрим множество F_n всевозможных функций f, которые $(n + 1)$ раз непрерывно дифференцируемы на $[a, b]$ и производная которых порядка $(n+1)$ ограничена по модулю числом $M_{n+1} : |f^{(n+1)}(x)| \leq M_{n+1}, x \in [a, b]$. В этом классе функций остаток интерполирования (методическая погрешность интерполирования) имеет оценку:

$$|r_n(x)| \leq \frac{M_{n+1}}{(n+1)!} |x - x_0||x - x_1| \ldots |x - x_n|. \tag{2}$$

Множитель $\frac{M_{n+1}}{(n+1)!}$ не зависит от выбора узлов, поэтому при фиксированном значении $\overline{x}$ необходимо выбрать $\overline{x_{i_k}}$ так, чтобы произведение

$$|\overline{x} - x_0||\overline{x} - x_1| \ldots |\overline{x} - x_n| \tag{3}$$

имело наименьшее значение. Эта величина принимает наименьшее возможное значение, если узлы интерполяции являются корнями полинома Чебышева степени $n + 1$:

$$x_i = \frac{1}{2}[(b - a) cos(\frac{2i + 1}{2(n + 1)})\pi + (b + a)], \ i = \overline{0, n} \tag{4}$$

1.2.1 Реализация алгоритма интерполяции на языке Python

```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt

def f(x):
    return x - np.sin(x) - 0.25

def lagrange(X, Y):
    x = symbols('x')
    L = 0
    for j in range(np.prod(X.shape)):
```

```
        l_num = 1
        l_denum = 1
        for i in range(np.prod(X.shape)):
            if i == j:
                l_num *= 1
                l_denum *= 1
            else:
                l_num *= (x - X[i])
                l_denum *= (X[j] - X[i])
        L += Y[j] * l_num / l_denum
    return collect(expand(L), x)

def nodes(f, p, n, a=-15, b=15):
    if p == 0:
        X = np.arange(a, b + 1, (b - a) / n)
        Y = np.array(f(X))
        return X, Y
    else:
        X = np.array([(((b - a) * np.cos((2 * i + 1) * np.pi / (2 * n + 2)) +
        (b + a)) / 2) for i in range(n + 1)])
        Y = np.array(f(X))
        return X, Y

Lf1 = lagrange(nodes(f, 0, 5)[0], nodes(f, 0, 5)[1])
print("L(x) = ", Lf1)
Lf2 = lagrange(nodes(f, 1, i)[0], nodes(f, 1, i)[1])
print("L(x) = ", Lf2)
```

1.2.2 Результат работы программы

Lf1 = -1.91935559627895e-9*x**5 + 1.75207959200903e-5*x**3 + 1.38777878078145e-17*x**2 + 0.9528024656179*x - 0.25
Lf2 = -3.44779736737544e-6*x**5 - 0.000448783500766511*x**3 - 2.77555756156289e-17*x**2 + 1.18136237339343*x - 0.25

1.3 Увеличим количество узлов

Увеличим количество интерполяционных узлов до 8 и 11. Это можно сделать в цикле.

```
for i in [10, 7, 5]:
    Lf1 = lagrange(nodes(f, 0, i)[0], nodes(f, 0, i)[1])
    print(Lf1)
    Lf2 = lagrange(nodes(f, 1, i)[0], nodes(f, 1, i)[1])
    print(Lf2)
```

1.3.1 Результат работы программы (для 8 узлов)

Lf1 = 2.84065982898414e-7*x**7 + 1.67289007082724e-20*x**6 - 0.000123150101420238*x**5 - 4.98732999343332e-18*x**4 + 0.0151774345792085*x**3 + 4.16333634234434e-16*x**2 + 0.540509494948253*x - 0.25
Lf2 = -1.39395935272614e-7*x**7 + 1.6940658945086e-21*x**6 + 4.53067236030277e-5*x**5 - 4.33680868994202e-19*x**4 - 0.00331883345355203*x**3 + 1.38777878078145e-16*x**2 + 0.952201084106592*x - 0.25

1.3.2 График

```
arr = np.arange(-15, 16, 0.01)
fig1, ax1 = plt.subplots()
ax1.set_ylim([-30, 30])
line = ax1.plot(arr, f(arr), label='f(x)')
ax1.plot(arr, [Lf1.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='random with %d nodes' %r)
ax1.plot(arr, [Lf2.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='minerr with %d nodes' %r)
```

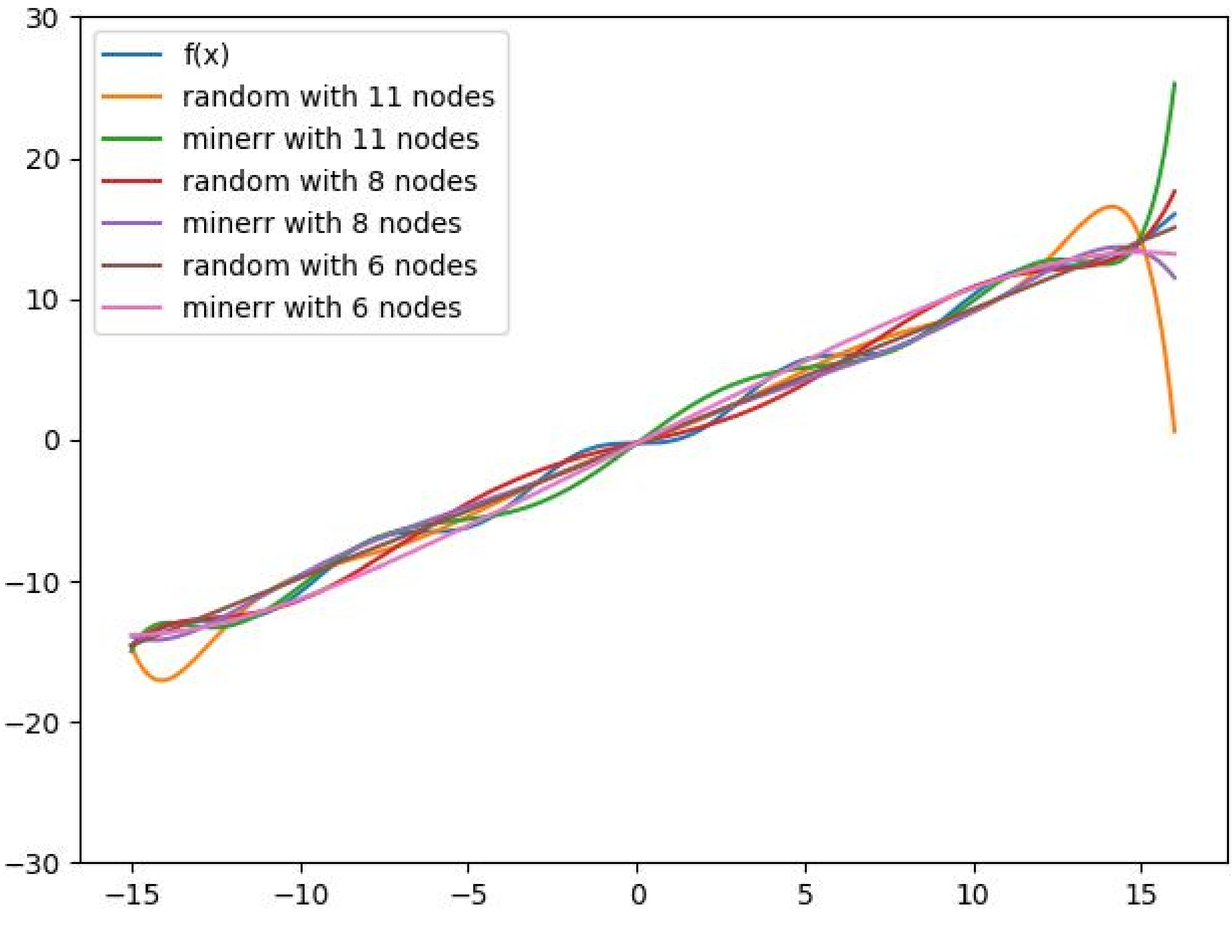

Рис. 1:

1.4 Аналогичная задача для $h(x) = |x|f(x)$

1.4.1 Реализация алгоритма интерполяции на языке Python

```
Lh1 = lagrange(nodes(h, 0, 5)[0], nodes(h, 0, 5)[1])
print(Lh1)
Lh2 = lagrange(nodes(h, 1, 5)[0], nodes(h, 1, 5)[1])
print(Lh2)
```

1.4.2 Результат работы программы

Lh1 = -0.000183133114729667*x**5 + 4.82253086419751e-5*x**4 + 0.0960514459203 66*x**3 - 0.0251736111111111*x**2 + 2.00925076094994*x - 0.527343749999999
Lh2 = -0.00026108895341774*x**5 + 3.74259142679528e-5*x**4 + 0.104897218830352*x**3 - 0.0220291070590848*x**2 + 3.03537085476482*x - 0.647047612756303

1.4.3 График

```
fig2, ax2 = plt.subplots()
ax2.set_ylim([-30, 30])
ax2.plot(arr, f(arr), label='f(x)')
ax2.plot(arr, [Lf1.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='random for f with 6 nodes')
ax2.plot(arr, [Lf2.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='minerr for f with 6 nodes')
ax2.plot(arr, [Lh1.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='random for h with 6 nodes')
ax2.plot(arr, [Lh2.subs(x, arr[j]) for j in range(np.prod(arr.shape))],
label='minerr for h with 6 nodes')
```

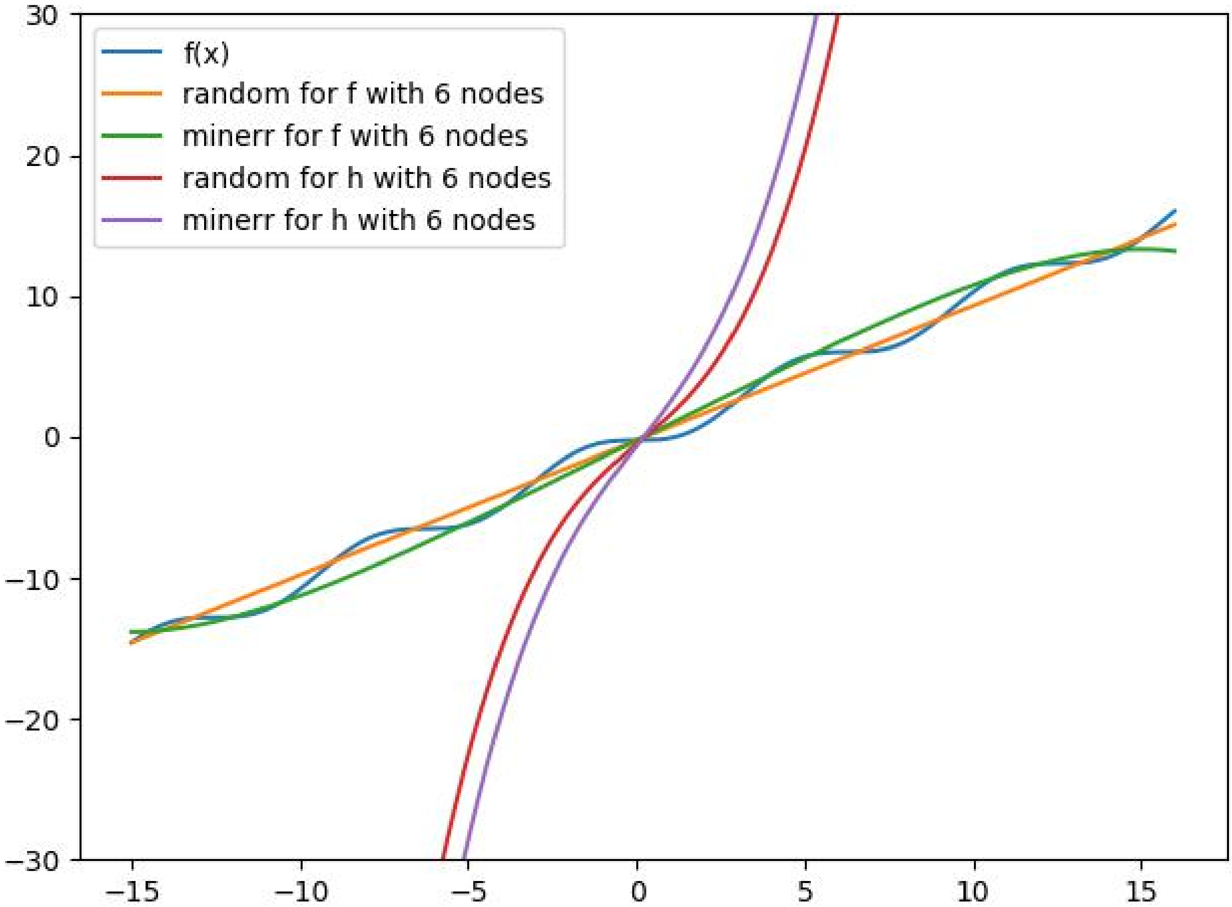

Рис. 2: