Національний технічний університет України

«Київський Політехнічний Інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Чисельні методи

Лабораторна робота №5

Інтерполяційні поліноми

Виконав:

Студент  
IІІ курсу ФІОТ  
групи ІП-22  
Сочка О. О.

Київ - 2015

**Постановка задачі**

Створити програму, яка для заданої функції по заданим точкам будує інтерполяційний поліном Pn(x) у формі Лагранжа або Ньютона, а також здійснює інтерполяцію кубічними сплайнами. Програма має розрахувавати значення похибки | P (x) y(x) | ε = n − , для чого потрібно вивести на графік із кроком (графік можна будувати допоміжними засобами, наприклад, у Mathcad), меншим у 5-6 разів, ніж крок інтерполяції, відповідні значення поліному та точної функції. Якщо похибка дуже мала, застосувати масштабування. Знайти кубічний інтерполяційний сплайн для заданої функції у Mathcad. Вивести графік результатів

**Варіант 17**

|  |  |  |
| --- | --- | --- |
| № | Функція y(x) | Вузли інтерполяції xi |
| 17 | ½ \* x \* cos(2x) | 0, 2, 4, 6, 8 |

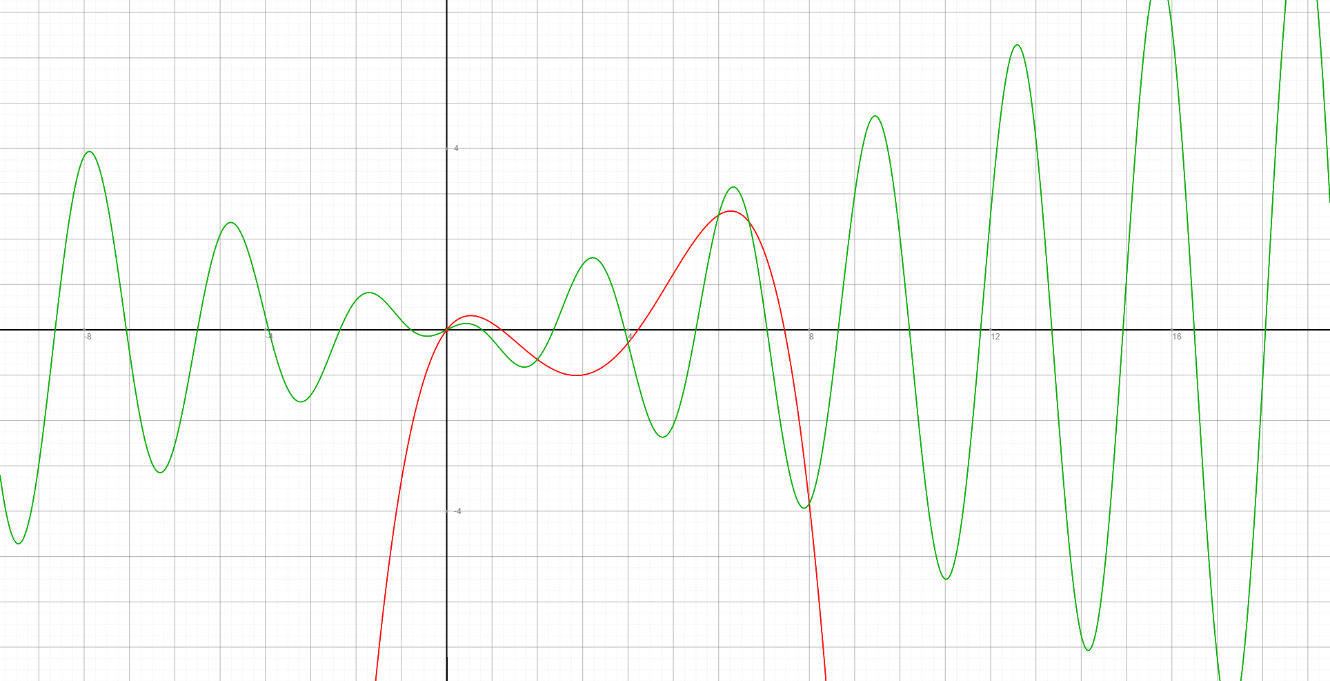
**Метод Лагранжа**

Метод Лагранжа (коефіціенти):

[-0.034084144136282184, 0.43908537997826796, -1.5531203472899566, 1.2957505173252932, 0.0]

Поліном

-0.0340841x^4 + 0.439085x^3 + -1.55312x^2 + 1.29575x + 0

****

Червоним – поліном Лагранжа, зеленим – функція

**Метод кубічних сплайнів**

Кубічний сплайн:

0.0 + 0(x - 0) + 0.0(x - 0)^2 + 0(x - 0)^3

-0.6536436208636119 -0.3943240906448245(x - 2) -0.10125342031952778(x - 2)^2 -0.05062671015976389(x - 2)^3

-0.2910000676172271 + 1.433866931478754(x - 4) + 1.9294444424431063(x - 4)^2 + 1.015348931381317(x - 4)^3

2.5315618761974763 -0.5633353896785598(x - 6) -3.92664676360042(x - 6)^2 -2.928045603021763(x - 6)^3

-3.8306379212935386 -4.489982153278981(x - 8) + 0.0(x - 8)^2 + 1.96332338180021(x - 8)^3

Помаранчевим– кубічний сплайн інтерполянт, синім – функція 1/2xcos(2x)

**Висновок**

В лабораторній роботі було досліджено 3 методи знаходження інтерполяційних поліномів – метод Лагранжа, метод Ньютона та кубічна сплайн-інтерполяція. Для двох з них (Лагранжа та кубічної сплайн-інтерполяції) було розроблене програмне забезпечення. Метод Лагранжа, на відміну від метода Ньютона, вимагає менше операцій, але при додаванні нових x вимагає повного перерахунку, тому його використання ускладняється в системах з real-time отриманням даних.

Лістинг програми

import math

def y(x):

return x \* math.cos(2\*x) / 2

xs = [0, 2, 4, 6, 8]

ys = [y(x) for x in xs]

n = len(xs)

def lagrange(x, y):

result = [0.] \* len(x)

for i in range(len(x)):

temp = lagrange\_compute(x, i)

for j in range(len(result)):

result[j] += y[i] \* temp[j]

print('Метод Лагранжа (коефіціенти): ')

print(result)

print('Поліном')

for i in range(len(x) - 2):

print('{:g}x^{} + '.format(result[i], len(x) - i - 1), end='')

print('{:g}x + {:g}'.format(result[-2], result[-1]))

def lagrange\_compute(x, k):

s = [i if i < k else i + 1 for i in range(len(x) - 1)]

div = 1

for i in range(len(x)):

if i == k: continue

div \*= x[k] - x[i]

result = [0] \* len(x)

result[0] = 1

for i in range(1, len(s) + 1):

st = list(range(i))

while True:

a = -1 if (i % 2 > 0) else 1

for j in st:

a \*= x[s[j]]

result[i] += a

temp = st.pop()

n = 1

while st and temp == len(s) - n:

temp = st.pop()

n += 1

if n == i and temp == len(s) - i:

break

else:

st.append(temp + 1)

temp += 1

while len(st) != i:

st.append(st[-1] + 1)

for i in range(len(result)):

result[i] /= div

return result

lagrange(xs, ys)

splines = []

def cubic\_spline(x, y):

global splines

n = len(x)

print()

print('Кубічний сплайн:')

splines = [{'x': x[i], 'a': y[i], 'b':0, 'c':0, 'd':0} for i in range(n)]

splines[0]['c'] = splines[n - 1]['c'] = 0.0

alpha = [0] \* (n - 1)

beta = [0] \* (n - 1)

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

hi = x[i] - x[i - 1]

hi1 = x[i + 1] - x[i]

C = 2.0 \* (hi + hi1)

F = 6.0 \* ((y[i + 1] - y[i]) / hi1 - (y[i] - y[i - 1]) / hi)

z = (hi \* alpha[i - 1] + C)

alpha[i] = -hi1 / z

beta[i] = (F - hi \* beta[i - 1]) / z

for i in range(n - 2, 0, -1):

splines[i]['c'] = alpha[i] \* splines[i + 1]['c'] + beta[i]

for i in range(n - 1, 0, -1):

hi = x[i] - x[i - 1]

splines[i]['d'] = (splines[i]['c'] - splines[i - 1]['c']) / hi

splines[i]['b'] = hi \* (2. \* splines[i]['c'] + splines[i - 1]['c']) / 6. + (y[i] - y[i-1]) / hi

for spline in splines:

[a,b,c,d,x] = [spline['a'], spline['b'], spline['c'], spline['d'], spline['x']]

print(str(a) + " " + ("+ " if (b >= 0) else "") +

str(b) + "(x - " + str(x) + ") " +

("+ " if (c >= 0) else "") +

str(c) + "(x - " + str(x) + ")^2 " +

("+ " if d >= 0 else "") +

str(d) + "(x - " + str(x) + ")^3")

def interpolate(x):

global splines

n = len(splines)

if x <= splines[0]['x']:

s = splines[0]

elif x >= splines[n - 1]['x']:

s = splines[0]

else:

i = 0

j = n - 1

while i + 1 < j:

k = i + (j - 1) // 2

if x <= splines[k]['x']:

j = k

else:

i = k

s = splines[j]

dx = x - s['x']

return s['a'] + (s['b'] + (s['c'] / 2.0 + s ['d'] \* dx / 6.) \* dx) \* dx

cubic\_spline(xs, ys)

xxx = list(map(lambda x: x / 10, range(0, 81)))

print('xxx')

print(' '.join(map(str, xxx)))

real = [y(x) for x in xxx]

print('real')

print(' '.join(map(str, real)))

yyy = [interpolate(x) for x in xxx]

print('yyy')

print(' '.join(map(str, yyy)))