­­МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

­­­

Контрольна робота

Похибки обчислювальних процесів

з курсу «Чисельні методи»

для студентів базового напрямку 6.08.04 "Комп’ютерні науки"

(заочна форма навчання)

Варіант 14

Виконав студент гр. КНз-2

Чалий Михайло

­­

Львів 2014

## Теоретичне завдання

### Метод Ньютона і його модифікації(42)



Формула називається **першою інтерполяційною формулою** **Ньютона**. Доведено, що існує лише один інтерполяційний многочлен n-го степеня, значення якого у вузлах інтерполяції  дорівнюють значен­ням функції .

Формулу Ньютона можна подати в зручнішому для користування вигляді. Для цього вводять нову безрозмірну змінну t за формулою , або . Тоді формула набирає вигляду:



Цей вигляд першої інтерполяційної формули Ньютона називають **фор­мулою Ньютона для інтерполювання вперед**. Таку назву вона отримала то­му, що в ній використовуються значення функції, дані у вузлах, які містяться вправо від  (вперед, вниз по стовпчику).

Для інтерполювання в кінці таблиці користуються іншою формулою вигляду:



Це є **друга інтерполяційна формула Ньютона для інтерполювання** **назад**. У ній використовуються скінченні різниці, розміщені в діагональній таблиці різниць по діагоналі знизу вгору.

Оскільки інтерполяційні формули Лагранжа і Ньютона – різні форми запису інтерполяційного многочлена, то оцінки залишкових членів формул Ньютона будуть такими, як і для формули Лагранжа, побудованої за такими самими даними. Тому у повну похибку результату, знайденого за формулами Ньютона, крім похибки методу, входитиме неусувна похибка, а також похиб­ка округлення.

#### Реалізація

# coding=utf-8

from numpy import \*

def func\_diff(y, n, i):

def diff(n, i):

if n == 1:

return y[i+1] - y[i]

if n == 0:

return y[i]

return diff(n-1, i+1) - diff(n-1, i)

return diff(n, i)

def newton1(x, y, point, degree):

h = x[1] - x[0]

q = (point - x[0])/h

def item(n):

qq = q

for j in range(1, n):

qq \*= (q - j + 1)

#print(n, "q:", qq, "diff:", func\_diff(y, n, 0))

return (qq/math.factorial(n)) \* func\_diff(y, n, 0)

return sum([y[0]] + [item(j) for j in range(1, degree + 1)])

def newton2(x, y, point, degree):

h = x[1] - x[0]

q = (point - x[-1])/h

def item(n):

qq = q

for j in range(1, n):

qq \*= (q + j - 1)

#print(n, "q:", qq, "diff:", func\_diff(y, n, degree-n))

return (qq/math.factorial(n)) \* func\_diff(y, n, degree-n)

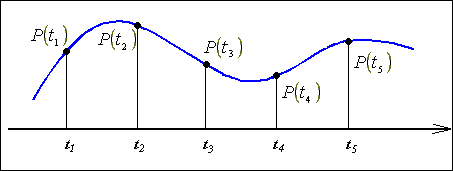
return sum([y[-1]] + [item(j) for j in range(1, degree + 1)])

### Сплайн-iнтерполяцiя. Програмна реалізація (14)

**Сплайн** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) *spline* — планка, рейка) — [функція](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [область визначення](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C_%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) якої розбита на шматки, на кожному зі шматків функція є деяким [поліномом](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BD%D0%BE%D0%BC) (многочленом).

Максимальний степінь поліномів в сплайні називається **степенем сплайна**. Різниця між степенем сплайна і його [гладкістю](https://uk.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F) називається **дефектом сплайна**.

Основна ідея застосування сплайнів полягає в наступному. Інтервал, на якому відновлюють функцію розбивають на підінтервали, на кожному з який функцію задають поліномом достатньо низького степені і забезпечують неперервність кривої в точках “склейки” шляхом прирівняння значень поліномів на межах підінтервалів.



*Відновлення сигналу за допомогою сплайнів.*

При цьому важливою умовою є також неперервність декількох похідних. Таким чином, сплайном *Pn* називають сукупність багаточленів *Pni* ступеня *n*, заданих на *i*-тому кроці дискретизації ,які задовольняють умові

*Pni(ti) = Pn(i-1)(ti)*,

тобто ступеня *n* сплайн- функціями, складеними з «шматочків» багаточленів даного ступеня, що сполучені так, щоб функція, що утворилася, була безперервною і мала декілька неперервних похідних.

Таким чином, сплайн є кусочно-поліноміальною функцією. Сплайн нульового степеня збігається зі східчасто-інтерпольованою функцією, а сплайн першого степеня - із лінійно-інтерпольованою. Слід зазначити, що сплайни другого і більш високого степеня не будуть збігатися з інтерполяційними поліномами відповідних степенів.

#### Приклад

from utils import \*

from pylab import \*

from numpy import \*

from scipy.interpolate import interp1d

x = linspace(0, 10, 10)

y = cos(-x\*\*2/8.0)

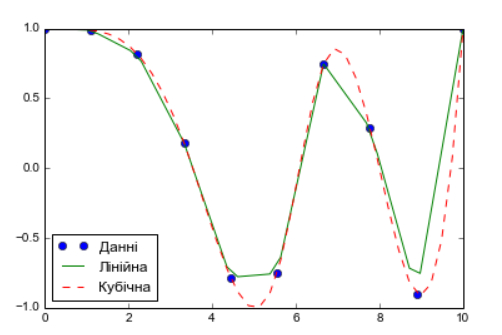
f = interp1d(x, y)

f2 = interp1d(x, y, kind='cubic')

xnew = linspace(0, 10, 40)

plot(x,y,'o',xnew,f(xnew),'-', xnew, f2(xnew),'--')

legend(['Данні', 'Лінійна', 'Кубічна'], loc='best')



#### Реалізація

from pylab import arange

from numpy import \*

from math import e

from math import pi

from math import sin

from math import cos

from numpy import poly1d

from numpy import polyval

def cubic(n, x, y):

zeroV = lambda m: ([0]\*m)

h = zeroV(n-1)

# alpha will be values in a system of eq's that will allow us to solve for c

# and then from there we can find b, d through substitution.

alpha = zeroV(n-1)

# l, u, z are used in the method for solving the linear system

l = zeroV(n+1)

u = zeroV(n)

z = zeroV(n+1)

# b, c, d will be the coefficients along with a.

a = y

b = zeroV(n)

c = zeroV(n+1)

d = zeroV(n)

for i in range(n-1):

# h[i] is used to satisfy the condition that

# Si+1(xi+l) = Si(xi+l) for each i = 0,..,n-1

# i.e., the values at the knots are "doubled up"

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

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

# Sets up the linear system and allows us to find c. Once we have

# c then b and d follow in terms of it.

alpha[i] = (3./h[i])\*(a[i+1]-a[i])-(3./h[i-1])\*(a[i] - a[i-1])

# I, II, (part of) III Sets up and solves tridiagonal linear system...

# I

l[0] = 1

u[0] = 0

z[0] = 0

# II

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

l[i] = 2\*(x[i+1] - x[i-1]) - h[i-1]\*u[i-1]

u[i] = h[i]/l[i]

z[i] = (alpha[i] - h[i-1]\*z[i-1])/l[i]

l[n] = 1

z[n] = 0

c[n] = 0

# III... also find b, d in terms of c.

for j in range(n-2, -1, -1):

c[j] = z[j] - u[j]\*c[j+1]

b[j] = (a[j+1] - a[j])/h[j] - h[j]\*(c[j+1] + 2\*c[j])/3.

d[j] = (c[j+1] - c[j])/(3\*h[j])

def polinome(a, b, c, d, x\_i):

root = poly1d(x\_i,True)

# print(root)

return d\*root\*\*3 + c\*root\*\*2 + b\*root + a

splines = [(x[j], x[j+1], polinome(a[j],b[j],c[j],d[j],x[j])) for j in range(n-1)]

return lambda v: (p(v) for (xa, xb, p) in splines if xa < v <= xb)

x = linspace(0, 10, 10).tolist()

y = [cos(-x\_val\*\*2/8.0) for x\_val in x]

cubic(10,x,y)