數值分析

Numerical Analysis

108 學年度　第二次學習成果報告

學 號 ： S07240018

姓 名 ： 溫宏岳

主題	使用Richardson extrapolation驗證 給定的值可近似Euler's number之研究與探討
講評	
評分	
☐ 需大幅修改　☐ 少許修正　☐ 無需修改	

I. 前提：Richardson extrapolation 是什麼？

這種方法將漸進式地利用低階的近似結果得到高階精度的近似結果。如果我們知道兩個同樣步長參數 h 下的近似值，並且同時知道當步長 h 趨近於零時，近似誤差的漸進精度函數形式。那麼，就可以根據這些近似值和函數形式，以外推插值的形式計算當 h = 0（也就是外推到極限情況）時的結果，以獲得高階的計算精度。

也就是說在數值分析中，理察森外推法用以改善級數序列收斂效率，它是在 20 世紀前期由英國數學家，物理學家，氣象學家 Lewis Fry Richardson 提出的。在數值分析領域，Richardson 外推法有很多實際應用，如勒貝格積分方法，是在梯形公式的基礎上應用 Richardson 外推法導出的；還有用於求解常微分方程的 Bulirsch-Stoer 算法。

From Wikipedia：

示例 [編輯]

應用理察森方法，改善用於近似微分的中心差分公式

$$f'(x_n) = \underbrace{\frac{f(x_n+h)-f(x_n-h)}{2h}}_{D(h)} - \underbrace{\frac{f'''(x_n)}{6}}_{a}h^2 - \underbrace{\frac{f^{(5)}(x_n)}{120}}_{b}h^4$$

則由式(1)可知$p=2, q=4, h_2=2h, r=\left(\frac{1}{2}\right)^p=\frac{1}{4}$ 代入公式：

$$D^* = \frac{D(h)-rD(h_2)}{1-r} = \frac{\frac{f(x_n+h)-f(x_n-h)}{2h} - \frac{1}{4}\frac{f(x_n+2h)-f(x_n-2h)}{4h}}{1-\frac{1}{4}}$$

$$D^* = \frac{8[f(x_n+h)-f(x_n-h)]-f(x_n+2h)+f(x_n-2h)}{12h}$$

由此，中心差分公式精度由2階變為4階。

II. 過程：算法及其運作方式(Python)

```
#11.a
import matplotlib.pyplot as plt
import numpy as np
import math

def f(x):
    return (1+x)**(1/x)

print('題目: lim h=>0 (1+h)^(1/h) = e')
print()

print('h = 0.04  :',f(0.04))

print('h = 0.02  :',f(0.02))

print('h = 0.01  :',f(0.01))

print('h = 0.005 :',f(0.005))
print()

print('  O(h)        O(h^2)      O(h^3)      O(h^4)')

N1 = [2.66583633, 2.69158803, 2.70481383, 2.71151712]

n = len(N1)
O = np.zeros((n,n))
O[:,0]=np.array(np.array(N1))

for i in range(1,n):
    for j in range(1,n):
        if (i>=j):
```

```
                O[i][j] = O[i][j-1]+(O[i][j-1]-O[i-1][j-
1])/(math.pow(2,j)-1)


            else:
                continue
print(O)


x = np.linspace(-10,10,100)
y = (1+x)**(1/x)
plt.plot(x,y)
plt.show()
```

Output

```
In [1]: runfile('D:/willy/Desktop/大學報告/01.應數系/17.數值分析(7必選5)/
05.報告/數值分析報告2/數值分析報告2_1(a).py', wdir='D:/willy/Desktop/大學報
告/01.應數系/17.數值分析(7必選5)/05.報告/數值分析報告2')
題目: lim h=>0 (1+h)^(1/h) = e

h = 0.04  : 2.665836331487422
h = 0.02  : 2.691588029073608
h = 0.01  : 2.7048138294215285
h = 0.005 : 2.711517122929317

  O(h)       O(h^2)     O(h^3)     O(h^4)
[[2.66583633 0.         0.         0.        ]
 [2.69158803 2.71733973 0.         0.        ]
 [2.70481383 2.71803963 2.71827293 0.        ]
 [2.71151712 2.71822041 2.71828067 2.71828178]]
```

由此可知，O(h^3) = 2.71827293 ≈ e with h = 0.04

O(h^4) = 2.71828178 ≈ e with h = 0.04

III. 過程：算法及其運作方式(Python)

#11.a 另解

```
import numpy as np

def f(x):
    return (1+x)**(1/x)

def richardson( x, n, h ):

    d = np.zeros((n+1,n+1))

    for i in range( n + 1 ):
        d[i,0] = 0.5 * ( f( x + h ) - f( x - h ) ) / h

        powerOf4 = 1

        for j in range( 1, i + 1 ):
            powerOf4 = 4 * powerOf4
            d[i,j] = d[i,j-1] + ( d[i,j-1] - d[i-1,j-1] )
/ ( powerOf4 - 1 )

        h = 0.5 * h

    return d

print(richardson(0.04,3,0.02))
```

Output

```
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/willy/Desktop/大學報告/01.應數系/17.數值分析(7必選5)/
05.報告/數值分析報告2/數值分析報告2_2(a).py', wdir='D:/willy/Desktop/大學報
告/01.應數系/17.數值分析(7必選5)/05.報告/數值分析報告2')
[[-1.26530582  0.          0.          0.        ]
 [-1.26499931 -1.26489714  0.          0.        ]
 [-1.2649227  -1.26489717 -1.26489717  0.        ]
 [-1.26490355 -1.26489717 -1.26489717 -1.26489717]]
```

由此可知，O(h^3) = -1.26489717 ≠ e with h = 0.04

為什麼會不接近尤拉數 e？因為 Richardson extrapolation 是將 f(x) 函數取導數才外推得到答案的方法，所以必須假設尤拉數 e 的定義 (1+x)**(1/x) = f(x) 取反導數，也就是 ∫ f(x)dx = ∫ (1+x)**(1/x)dx。但是∫ (1+x)**(1/x)dx 無法算出解答。

再由 WolframAlpha 證實此程式碼為 (1+x)**(1/x) 取導數後得 (-(math.log(1+x))/(x**2) + 1/(x+(x**2)))*((1+x)**(1/x)) 帶入 x = 0.04 的值為 -1.26489717，並不是 ∫ f(x)dx = ∫ (1+x)**(1/x)dx 的程式碼。

因此，此題目 11.a 逼近尤拉數 e 的此程式碼(另解)不適合用 Richardson extrapolation。

IV. 過程：算法及其運作方式(Python)

```
#11.b
import numpy as np
import math

def f(x):
    return ((2+x)/(2-x))**(1/x)

print('題目: lim h=>0 ((2+h)/(2-h))^(1/h) = e')
print()

print('h = 0.04  :',f(0.04))

print('h = 0.02  :',f(0.02))

print('h = 0.01  :',f(0.01))

print('h = 0.005 :',f(0.005))
print()

print('  O(h^2)     O(h^4)     O(h^6)     O(h^8)')

N1 = [2.71864438, 2.71837244, 2.71830448, 2.71828749]

n = len(N1)
O = np.zeros((n,n))
O[:,0]=np.array(np.array(N1))

for i in range(1,n):
    for j in range(1,n):
        if (i>=j):
            O[i][j] = O[i][j-1]+(O[i][j-1]-O[i-1][j-
1])/(math.pow(4,j)-1)
```

```
            else:
                continue
    print(O)
```

Output

```
In [1]: runfile('D:/willy/Desktop/大學報告/01.應數系/17.數值分析(7必選5)/
05.報告/數值分析報告2/數值分析報告2_1(b).py', wdir='D:/willy/Desktop/大學報
告/01.應數系/17.數值分析(7必選5)/05.報告/數值分析報告2')
題目: lim h=>0 ((2+h)/(2-h))^(1/h) = e

h = 0.04  : 2.7186443772212376
h = 0.02  : 2.7183724448006217
h = 0.01  : 2.7183044812417467
h = 0.005 : 2.7182874915732635

  O(h^2)     O(h^4)     O(h^6)     O(h^8)
[[2.71864438 0.         0.         0.        ]
 [2.71837244 2.71828179 0.         0.        ]
 [2.71830448 2.71828183 2.71828183 0.        ]
 [2.71828749 2.71828183 2.71828183 2.71828183]]
```

由此可知，O(h^6) = 2.71828183 ≈ e with h = 0.04

O(h^8) = 2.71828183 ≈ e with h = 0.04

V. 過程：算法及其運作方式(Python)

#11.b 另解

```
import numpy as np

def f(x):
    return ((2+x)/(2-x))**(1/x)

def richardson( x, n, h ):

    d = np.zeros((n+1,n+1))

    for i in range( n + 1 ):
        d[i,0] = 0.5 * ( f( x + h ) - f( x - h ) ) / h

        powerOf4 = 1

        for j in range( 1, i + 1 ):
            powerOf4 = 4 * powerOf4
            d[i,j] = d[i,j-1] + ( d[i,j-1] - d[i-1,j-1] ) /
( powerOf4 - 1 )

        h = 0.5 * h

    return d

print(richardson(0.04,3,0.02))
```

Output

```
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/willy/Desktop/大學報告/01.應數系/17.數值分析(7必選5)/
05.報告/數值分析報告2/數值分析報告2_2(b).py', wdir='D:/willy/Desktop/大學報
告/01.應數系/17.數值分析(7必選5)/05.報告/數值分析報告2')
[[0.01813578 0.         0.         0.        ]
 [0.0181337  0.018133   0.         0.        ]
 [0.01813317 0.018133   0.018133   0.        ]
 [0.01813304 0.018133   0.018133   0.018133  ]]
```

由此可知，O(h^3) = 0.018133 ≠ e with h = 0.04

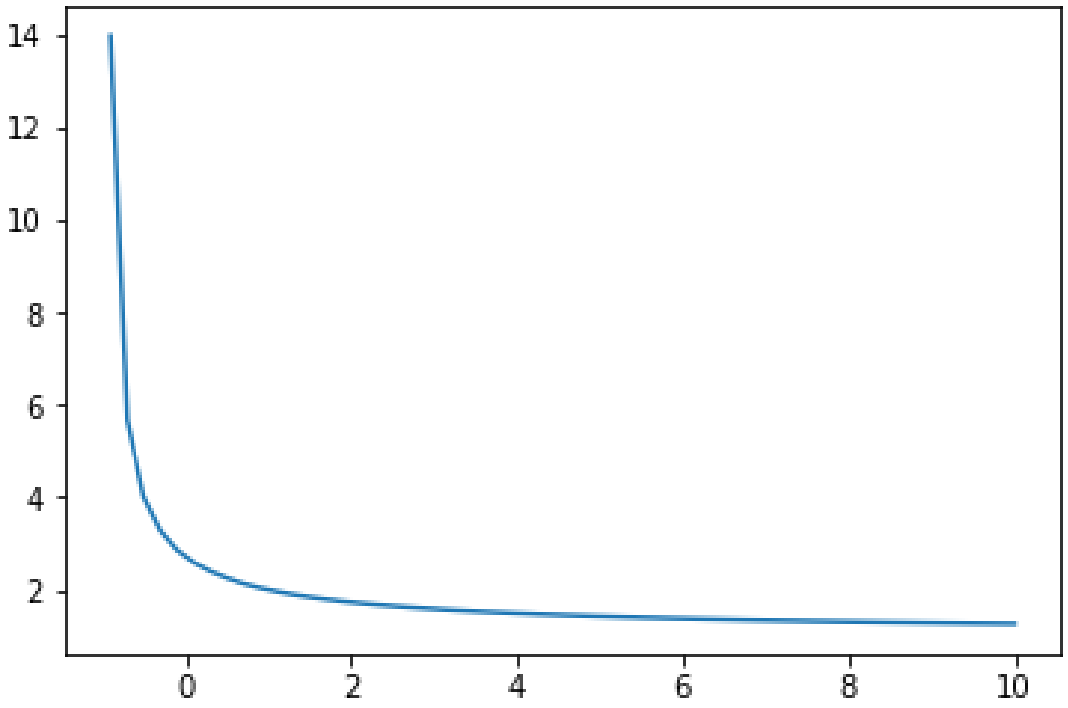
為什麼會不接近尤拉數 e？因為 Richardson extrapolation 是將 f(x) 函數取導數才外推得到答案的方法，所以必須假設尤拉數 e 的定義 (1+x)**(1/x) = f(x) 取反導數，也就是 ∫ f(x)dx = ∫ (1+x)**(1/x)dx。但是∫ (1+x)**(1/x)dx 無法算出解答。

再由 WolframAlpha 證實此程式碼為 (1+x)**(1/x) 取導數後得 (-(math.log(1+x))/(x**2) + 1/(x+(x**2)))*((1+x)**(1/x)) 帶入 x = 0.04 的值為 0.018133，並不是 ∫ f(x)dx = ∫ (1+x)**(1/x)dx 的程式碼。

因此，此題目 11.b 逼近尤拉數 e 的此程式碼(另解)不適合用 Richardson extrapolation。

Image

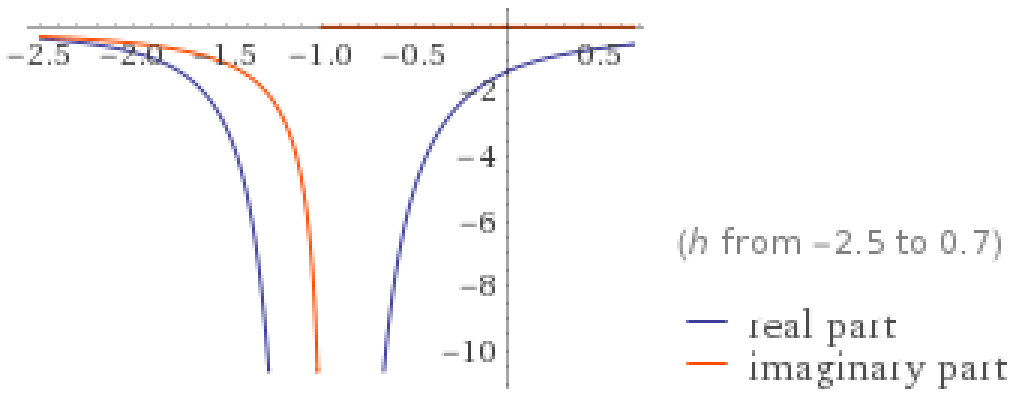
From Python 11.a：




```
y = (1+x)**(1/x)
```

y(0)= e

From WolframAlpha 11.a 另解：




```
y' = (-(math.log(1+x))/(x**2) +
1/(x+(x**2))) * ((1+x)**(1/x))
```

y'(0)= -1.26489717

#同樣地，11.b 亦是如此

VI. 結論：為什麼要使用 Richardson extrapolation？

Euler's number (e)

$\lim_{h\to 0}(1+h)^{\frac{1}{h}}$ by $N_4(h)$	2.71828178
$\lim_{h\to 0}\left(\frac{2+h}{2-h}\right)^{\frac{1}{h}}$ by $N_4(h)$	2.71828183
Exact of e	2.7182818284..

我們可以從 $\lim_{h\to 0}(1+h)^{\frac{1}{h}}$ by $N_1(h)$ = 2.71151712 及 $\lim_{h\to 0}\left(\frac{2+h}{2-h}\right)^{\frac{1}{h}}$ by $N_1(h)$ = 2.71828749 看出後者已經較為接近尤拉數 e，再用 Richardson extrapolation，雖然除法的運算在電腦程式上較為複雜、運算較長，但我們可以得到較好的精度！

也因為可以用 Richardson extrapolation 方式進行任意高階的推導。然而，我們需要做的是盡可能地減少 h 所導致的捨入誤差。

通過將參數的有限值處的多個計算外推到零值處的結果來提取精確的分析結果的想法，在科學計算中是一種非常強大的工具。（在統計力學中也有同樣的想法，通常被稱為有限大小縮放。）

$$N_{j+1}(h) = N_j(h/2) + \frac{N_j(h/2) - N_j(h)}{2^j - 1}$$

$\mathcal{O}(h)$	$\mathcal{O}(h^2)$	$\mathcal{O}(h^3)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^5)$
$N_1(h)$				
$N_1(h/2)$	$N_2(h)$			
$N_1(h/4)$	$N_2(h/2)$	$N_3(h)$		
$N_1(h/8)$	$N_2(h/4)$	$N_3(h/2)$	$N_4(h)$	
$N_1(h/16)$	$N_2(h/8)$	$N_3(h/4)$	$N_4(h/2)$	$N_5(h)$
↑ *Measurements*	↑	*Extrapolations*		↑

參考文獻:

http://moocs.nccu.edu.tw/course/132/intro

http://muchomas.lassp.cornell.edu/P480/Notes/integ/node10.html

https://jmahaffy.sdsu.edu/courses/f16/math541/beamer/richard.pdf

http://www.math-cs.gordon.edu/courses/mat342/python/richardson.py

https://sites.math.washington.edu/~greenbau/Math_498/lecture04_richardson.pdf

https://zh.wikipedia.org/wiki/%E7%90%86%E6%9F%A5%E5%BE%B7%E6%A3%AE%E5%A4%96%E6%8E%A8%E6%B3%95