N	10	20	30	40	100	300	1024
sum	0.6687	0.6808	0.6848	0.6869	0.69065	0.69231	0.692903

Table 1: results from Equation (1)

$ Supplement Note 9: Linear and Quadratic Extrapolation

In **Example 7.2**, p.7-2 of **Lecture Note**, it is shown that the value of $\ln 2$ may be obtained by the sum of the *alternating harmonic series*:

$$\begin{aligned}\ln 2 &= \int_1^2 \frac{dx}{x} = \int_0^1 \frac{dt}{1+t} = \int_0^1 (1 - t + t^2 - t^3 + \cdots)dt \\ &= 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} + \cdots = \frac{1}{1*2} + \frac{1}{3*4} + \frac{1}{5*6} + \cdots \\ &= \lim_{N\to\infty} S_N; \quad \text{where} \quad S_N \equiv \sum_{k=1}^{N} \frac{1}{2k(2k-1)} \end{aligned} \tag{1}$$

The convergence of the alternating harmonic series S_N in (1) is slow. Typical results are shown in Table 1. For $N = 1024$, an approximation value 0.692903 for $\ln 2 = 0.6931471805599453$ is obtained, not a very satisfactory result.

Linear Extrapolation

The convergence in the calculation may be improved by making a *linear extrapolation* from two calculated results S_{N_1}, S_{N_2}. Assuming that the result S_N for the alternating harmonic series (1) has an error proportional to $\frac{1}{N}$, S_{N_1}, S_{N_2} may be viewed as the values of a linear function $f_1(x) \equiv a + bx$ at the points $\frac{1}{N_1}, \frac{1}{N_2}$

$$S_{N_i} = f_1(x_i) \equiv a + bx_i; \quad x_i \equiv \frac{1}{N_i},\ i = 1, 2 : \tag{2}$$

As $N_i \to \infty$, $x_i \to 0$, *the constant* $a \equiv S_\infty$ *is then the desired approximation to* $log(2)$. The coefficient b in (2) may be obtained by a *divided difference* of the linear function $f_1(x)$ at $x = x_1, x_2$:

$$b = S_{01} \equiv \frac{f_1(x_2) - f_1(x_1)}{x_2 - x_1} = \frac{S_{N_2} - S_{N_1}}{x_2 - x_1} \tag{3}$$

N_1 , N_2	approximated value for ln 2 from linear extrapolation
10, 20	0.6928353604099597
20, 30	0.6930430825086519
30, 40	0.6930951139425222
40, 100	0.6931315574636133
100, 300	0.6931450972642266
300, 1024	0.6931469771098154

Table 2: Approximated Value for ln 2 from Linear Extrapolation

Some typical results from linear extrapolation of the alternating harmonic series are shown in Table 2. The the convergence of the series is mush faster.

Quadratic Extrapolation

A further improvement may be obtained by using a *quadratic extrapolation* from three calculated results $S_{N_1}, S_{N_2}, S_{N_3}$, where additional error terms proportional to $(\frac{1}{N_i})^2$ are also included. The values $S_{N_i}, i = 1, 2, 3$ are now viewed as the values of a quadratic function $f_2(x) \equiv a_0 + a_1 x + a_2 x^2$ at the points $x_i \equiv \frac{1}{N_i}$:

$$S_{N_i} = f_2(x_i) \equiv a_0 + a_1 x_i + a_2 x_i^2, \quad x_i \equiv \frac{1}{N_i},\ i = 1, 2, 3\ . \tag{4}$$

Here again, *the constant $a_0 \equiv S_\infty$ is the desired approximation to log*(2). The coefficient a_2 of the highest order in (4) may be obtained by repeated use of *divided difference* of the quadratic functions $f_2(x_i)$:

$$S_{01}(x_0, x_1) \equiv \frac{f_2(x_0) - f_2(x_1)}{x_0 - x_1} = \frac{S_{N_0} - S_{N_1}}{x_0 - x_1} = a_1 + a_2(x_0 + x_1) \tag{5}$$

$$S_{12}(x_1, x_2) \equiv \frac{f_2(x_2) - f_2(x_1)}{x_2 - x_1} = \frac{S_{N_2} - S_{N_1}}{x_2 - x_1} = a_1 + a_2(x_1 + x_2) \tag{6}$$

$$a_2 = S_{012} \equiv \frac{S_{01}(x_0, x_1) - S_{12}(x_1, x_2)}{x_0 - x_2} \tag{7}$$

The constant a_0 may then be calculated from Eqs.(6) and (4):

$$a_0 = S_2(x_1) - x_1\Big(S_{12}(x_1, x_2) - a_2 x_2\Big) \tag{8}$$

Some typical results are shown in Table 3, the results from the quadratic extrapolation are very close to the exact value.

$\ln 2 = 0.6931471805599453$

N_1 , N_2 , N_2	results from quadratic extrapolation
10 ,20, 30	0.6931469435579979
20 ,30, 40	0.6931471453763925
30 ,40, 100	0.6931471761155095
40 ,100, 300	0.6931471803104747
100, 300, 1024	0.6931471805563078

Table 3: log(2) from Quadratic Extrapolation

Generalization to extrapolation using higher order of polynomials is straightforward. However, *truncation errors* due to finite degrees of accuracy for real numbers represented by computers (64 bits) have to be carefully handled for such higher order extrapolations. In practical application, considering the simplicity of the approach and the accuracy of the results, *linear extrapolation and quadratic extrapolation are highly recommended.*

We list some of the relevant formulas from divided difference for *cubic extrapolation* using cubic function $f_3(x) \equiv a_0 + a_1 x + a_2 x^2 + a_3 x^3$. Note that *the divided difference is symmetric with respect to all its arguments*.

$$S_{01}(x_0, x_1) \equiv \frac{f_3(x_0) - f_3(x_1)}{x_0 - x_1} = a_1 + a_2(x_0 + x_1) + a_3(x_0^2 + x_1^2 + x_0 x_1) \tag{9}$$

$$S_{012}(x_0, x_1, x_2) \equiv \frac{S_{01}(x_0, x_1) - S_{12}(x_1, x_2)}{x_0 - x_2} = a_2 + a_3(x_0 + x_1 + x_2) \tag{10}$$

$$a_3 = S_{0123} \equiv \frac{S_{012}(x_0, x_1, x_2) - S_{123}(x_1, x_2, x_3)}{x_0 - x_3} \tag{11}$$

```
# log2.py
# calculate log2 by linear extrapolation and quadratic extrapolation
# on the alternating harmonic series log(2)= sum_{n = 1} 1/2n(2n -1)
import numpy as np
import matplotlib.pyplot as plt
nn =[10, 20, 30, 40, 100, 300, 1024]
ni, s, log2s= 0, 0, []
for nf in nn:
```

```
    for i in range(ni+1, nf+1):
        fac = 2*i *(2*i -1)
        s +=  1/fac
    log2s.append(s)
    ni = nf
print(' results from alternating harmonic series sum\n log(2), n ')
for n,s in zip(nn, log2s):
    print(s, n)
print('log(2) =', np.log(2))
print('linear extrapolation')
def LinExtra(n0, s0, n1, s1): #linear extrapolation'
    ninv0 , ninv1 = 1/n0,1/n1
    s01 = (s0-s1)/(ninv0-ninv1)
    print(s0 - s01*ninv0, s1 - s01*ninv1, n0,n1)
s0, n0 = log2s[0], nn[0]
for s1, n1 in zip(log2s[1:], nn[1:]):
    LinExtra(n0, s0, n1, s1)
    s0, n0 = s1, n1
print('quadratic extrapolation')
def QuadExtra(n0, s0, n1, s1, n2, s2): #quadratic extrapolation'
    ninv0 , ninv1 , ninv2 = 1/n0,1/n1,1/n2
    s01 = (s0-s1)/(ninv0-ninv1)
    s12 = (s1-s2)/(ninv1-ninv2)
    s012 = (s12-s01)/(ninv2-ninv0)
    print(n0,n1,n2)
    b= s1 -ninv1*(s12 - s012*ninv2)
    print(b)
    b= s2 -ninv2*(s12 - s012*ninv1)
    print(b)
    b= s0 -ninv0*(s01 - s012*ninv1)
    print(b)
s0, n0 = log2s[0], nn[0]
s1, n1 = log2s[1], nn[1]
```

```
for s2, n2 in zip(log2s[2:], nn[2:]):
    QuadExtra(n0, s0, n1, s1, n2, s2)
    s0, n0 = s1, n1
    s1, n1 = s2, n2
```