log(2)= 0.6931471805599453

N	201	301	501	701	801	901
S_N	0.69190	0.69232	0.69265	0.69279	0.69284	0.69287

Table 1: Sub-sums from series expansion Eq.(3) for log(2)

$ Supplement Note 16: Polynomial Fit: A Simple Trick Can Go a Long Way

A polynomial function of *order k* may be written in general as

$$p_k(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} + a_k x^k \tag{1}$$

where k is the order of the polynomial and the $a's$ are constants.

Polynomials have many applications in science and engineering. They are used extensively for extrapolating, interpolating and curve fitting of functions. As a first example, let us consider the evaluation of log(2) by using the series expansion **Example 7.2** (p.7-2):

$$\ln 2 = (1 - \frac{1}{2}) + (\frac{1}{3} - \frac{1}{4}) + (\frac{1}{5} - \frac{1}{6}) \cdots = \lim_{N\to\infty} S_N \tag{2}$$

where for numerical stability, consecutive alternating terms have been explicitly summed together. The sub-sums S_N are given explicitly by

$$S_N = \sum_{k=1}^{N} \frac{1}{2k(2k-1)} \tag{3}$$

Some of the results from Eq.(3) are listed in Table 1. The convergence of the results as a function of N is rather slow. The problem may be approached by using the polynomials for extrapolation. To monitor the errors of the sub-sums S_N versus N, they are expanded as polynomial functions of order k for the variable $x \equiv \frac{1}{N}$:

$$S_N = p_k\left(\frac{1}{N}\right) = a_0 + a_1\left(\frac{1}{N}\right) + a_2\left(\frac{1}{N}\right)^2 + \cdots + a_{k-1}\left(\frac{1}{N}\right)^{k-1} + a_k\left(\frac{1}{N}\right)^k \tag{4}$$

so that the value of log(2) may be obtained by taking the limit $N \to \infty$:

$$\log(2) = \lim_{N\to\infty} S_N = p_k(0) = a_0 \tag{5}$$

The coefficients a_i of the polynomial $p_k(x)$ can be fitted with $k+1$ calculated values of S_N. This is easily done by using the *numpy.polyfit* sub-package. The results of the polynomials may then be evaluated by using the *numpy.poly1d* sub-package.

Here sub-sums S_N with N from 13 to 20 in the series expansion Eq.(2) for log(2) are used for such extrapolations using polynomials of orders $k = 1, \ldots, 7$. The values for these sub-sums S_N are listed in Table 2 and the extrapolated results for $log(2)$ are listed in Table 3. Extremely satisfactory results can be obtained using polynomials of order $k = 4$. Even results obtained by linear extrapolation ($k = 1$) exceed the brute force results in Table 1 with $N = 900$. As is always the case, due to numerical cancellations and intrinsic oscillation behaviors, *extrapolation using polynomials of higher orders should always be taken with caution.*

Supplement Problem 1 *As an exercise, calculate the following series*

$$\sigma = \sum_{k=1}^{\infty} \frac{1}{k^2} \quad ; \quad \sigma_1 = \sum_{k=0}^{\infty} \frac{1}{(2k+1)^2}$$

by using polynomial expansion and compare with the exact results $\frac{\pi^2}{6}$ *and* $\frac{\pi^2}{8}$ *(Eq. (13.18) and Eq. (13.19) in Lecture Note.)*

```
# PolyfitLog2.py  calculate log2 by Polynomials
# using numpy.polyfit and numpy.poly1d sub-package
import numpy as np
Log2 = np.log(2)
N= 20
nn = np.arange(1, N+1)   # print('nn=',nn)
xx = 1/nn                # print('1/n=',xx)
nn2 = 2*nn
np1 = nn2*(nn2-1)        # print('2n*(2n-1)',np1)
b = 1/np1                # print('1/2n*(2n-1)',b)
```

sub-sum	value
S_{13}	0.6742859610812901
S_{14}	0.6756087124040414
S_{15}	0.6767581376913977
S_{16}	0.6777662022075267
S_{17}	0.6786574678046746
S_{18}	0.6794511185983254
S_{19}	0.6801623561516682
S_{20}	0.6808033817926938

Table 2: Sub-sums S_N used in the polynomial extrapolations

log(2)= 0.6931471805599453

k	fitted value for log(2)	error	sub-sums used
1	0.6929828689721763	-1.6431e-04	S_{19}, S_{20}
2	0.6931470007152927	-1.7984e-07	S_{18}, S_{19}, S_{20}
3	0.6931472467682191	6.6208e-08	$S_{17}, S_{18}, S_{19}, S_{20}$
4	0.6931471811315769	5.7163e-10	$S_{16}, S_{17}, S_{18}, S_{19}, S_{20}$
5	0.6931471804272558	-1.3269e-10	$S_{15}, S_{16}, S_{17}, S_{18}, S_{19}, S_{20}$
6	0.6931471806794017	1.1946e-10	$S_{14}, S_{15}, S_{16}, S_{17}, S_{18}, S_{19}, S_{20}$
7	0.6931471807176897	1.5774e-10	$S_{13}, S_{14}, S_{15}, S_{16}, S_{17}, S_{18}, S_{19}, S_{20}$

Table 3: Extrapolated results for log(2) using polynomials of order k

```
Sn = np.add.accumulate(b)
nnAll = nn[12:N]          # print('nnAll=',nnAll)
SnAll = Sn[12:N]          # print('SnAll=',SnAll)
print('log2=', Log2)
print('N,  S_N   used in this program')
for i,t in zip(nnAll,SnAll): print(i,t)
print('******************************')
Xn,yy =[],[]
for n in range(1,8):
    N1= N - n - 1
    xn, x, y = nn[N1:N], xx[N1:N], Sn[N1:N] # print(x)
    Xn.append(xn)                           # print(y)
    z = np.polyfit(x,y,n)
    p = np.poly1d(z)
    y0, ydif= p(0), p(0)-Log2
    yy.append(y0)
    print('N which participating in polyfit=',xn)
    print('Values of S_N used which participating in polyfit=\n',y)
    print('n= order of the fitted polynomial=',n,'; p_n(x)=')
    print('\n', np.poly1d(p),'\n')
    print('extrapolated value of log(2)=',y0,', error={:12.4e}'.format(ydif))
    print('___________________________________')
for i,x in enumerate(zip(Xn,yy)):
    xn,y0 = x
    n, ydif = i + 1, y0-Log2
    print(n,y0,'{:12.4e}'.format(ydif),xn)
```