# Numerical Method: Computer Assignment #1

Hong Chenhui
3220101111@zju.edu.cn

Zhejiang University — March 5, 2024

**Info:** It is important to note that the algorithm is valid in theory. The execution, though, might not be accurate. As such, it is imperative to confirm the implementation. Additionally, because the Scarborough approach is just too rigorous, the numbers of actual significant digits in the approximation may not strictly yield six.

## 1 Problem Statement

Two following series may be used to approximate the value of logarithm function.

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots \tag{1}$$

$$\ln(\frac{1+x}{1-x}) = 2(x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots) \tag{2}$$

1. Develop a program that iterates 100 times to approximate both $\ln 2$ and $\ln 1.5$ using the two series mentioned above independently. Analyze the results.
2. To guarantee that the results are correct to six significant digits, modify your program.

### 1.1 Theoretical viewpoint

For the initial question, there isn't much to discuss. All we need to do is create a Matlab program that can provide a result after 100 iterations and compare it with the specified real number.

It would be practical to associate the percent relative errors $\epsilon_a$ to the quantity of significant figures in the estimate for the second question. Scarborough (1966) demonstrated that we may be certain the result is accurate to at least n significant figures if the following condition is satisfied,

$$\frac{\epsilon_s}{(0.5 \times 10^{2-n})\%} < 1 \tag{3}$$

For such cases, the computation is repeated until

$$|\epsilon_a| < \epsilon_s \tag{4}$$

### 1.2 Algorithmic issues

Making it a function that can be called directly from the command line would be a smart idea, especially because there are a lot of repetitive chores to complete.The functions in both queries respond in a manner that is strikingly similar and can be characterized as the pseudocode below. (Problem 2 as an example)

**Algorithm 1:** `IterationMethod`

```
Input: $(x, \epsilon_s)$, two floating-point numbers
Result: $(value, \epsilon_a, iter)$, among which $iter$
        stands for the number of times the
        function iterated

while $|\epsilon_a| \geq \epsilon_s$ do
    add one more item ;
    calculating $\epsilon_a$ ;
    $iter = iter + 1$ ;
end
return $(value, \epsilon_a, iter)$ ;
```

## 2 Implementation

Steven C. Chapra offers a quite elegant iteration code in his book *Numerical Methods for Engineers*. His structure may be used to write similar code with only minor modifications. Hence the following using Matlab.

Command Line

```
>>format long
```

**or else the result won't be displayed properly.*

IterMeth.m

```
function [v,iter] = IterMeth(x,maxit)

% initialization
iter = 1;
sol = 0;

while (1)
 sol = sol + (x ^ iter) * ((-1)^(iter+1)) /iter;
 iter = iter + 1;
 end
 if iter>=maxit,break,end
end

v = sol;
end
```

Command Line

```
>>[v,iter] = IterMeth(1,100)

v =

   0.698172179310195


iter =

   100


>>trueval = log(2)

trueval =

   0.693147180559945

>> et=abs((trueval-v)/trueval)*100

et =

   0.724954077745856

>> [val,iter] = IterMeth(0.5,100)

val =

  0.405465108108164


iter =

  100

>> trueval = log(1.5)

trueval =

  0.405465108108164

>> et=abs((trueval-val)/trueval)*100

et =

2.738146889643058e-14
```

IterMeth2.m

```
function [v,iter] = IterMeth(x,maxit)

% initialization
iter = 1;
sol = 0;

while (1)
 sol = sol + (x ^ (2 * iter-1)) / (2 * iter-1);
 iter = iter + 1;
 end
 if iter>=maxit,break,end
end

v = 2 * sol;
end
```

Command Line

```
>> [val2, iter2] = IterMeth2( 1/3, 100 )

val2 =

   0.693147180559945


iter2 =

   100

>> trueval = log(2)

trueval =

   0.693147180559945

>> et = abs((trueval-val2)/trueval)*100

et =

 3.203426503814917e-14

>> [val2, iter2] = IterMeth2( 1/5, 100 )

val2 =

       0.405465108108164


iter2 =

       100

>> trueval2 = log(1.5)

trueval2 =

       0.405465108108164

>> et = abs((trueval2 - val2)/trueval2)*100

et =

       1.369073444821529e-14
```

To solve the first issue, use the two codes mentioned above. As for the second question, the code is as follows.

IterMeth3.m

```
function [v,ea,iter] = IterMeth3(x,es)

% initialization
iter = 1;
sol = 0;
ea = 100;

% iterative calculation
while (1)
 solold = sol;
 sol = sol + (x ^ iter) * ((-1)^(iter+1)) /iter;
 iter = iter + 1;
 if sol~=0
 ea=abs((sol - solold)/sol)*100;
 end
 if ea<=es,break,end
end

v = sol;
end
```

Command Line

```
>> [val, ea, iter] = IterMeth3(1, 0.5*(1e-4))

val =

   0.693147353846754


ea =

 4.999997159366927e-05


iter =

 2885392

>> [val, ea, iter] = IterMeth3(0.5, 0.5*(1e-4))

val =

   0.405465140417893


ea =

 2.475842191288193e-05


iter =

20
```

IterMeth4.m

```
function [v,ea,iter] = IterMeth4(x,es)

% initialization
iter = 1;
sol = 0;
ea = 100;

% iterative calculation
while (1)
 solold = sol;
 sol = sol + (x ^ (2 * iter-1)) * 2 / (2 * iter-1);
 iter = iter + 1;
 if sol~=0
 ea=abs((sol - solold)/sol)*100;
 end
 if ea<=es,break,end
end

v = sol;
end
```

Command Line

```
>> [val, ea, iter] = IterMeth4(1/3, 0.5*(1e-4))

val =

   0.693147170256012


ea =

 1.392146299193978e-05


iter =

 8

>> [val, ea, iter] = IterMeth4(1/5, 0.5*(1e-4))

val =

   0.405465104253968


ea =

   2.806105298857764e-05


iter =

   6
```

## 3 Analysis

The staggered series provided by Eq. (1) may result in a strong smearing effect, which slowed the pace of convergence in IterMeth1.m. Which, as the chart below illustrates, is noticeably true when taking into account the initial approximation finding regarding $\ln 2$.

| | Value | | | |
|---|---|---|---|---|
| | $\ln 2_1$ | $\ln 1.5_1$ | $\ln 2_2$ | $\ln 1.5_2$ |
| $\epsilon_t$ | 0.72 | 2.74e-14 | 3.20e-14 | 1.37e-14 |

When we learn the number of iterations in the second question—among which the first outcome reached an astounding 2885392—this conclusion becomes more solid. As illustrated by the chart below.

| | Iter | | | |
|---|---|---|---|---|
| | $\ln 2_1$ | $\ln 1.5_1$ | $\ln 2_2$ | $\ln 1.5_2$ |
| $\epsilon_t$ | 2885392 | 20 | 8 | 6 |

We could also see that the second series, as shown in IterMeth2.m, is much more effective than the first, which proposed a better approximation method. This result is constructive regarding the choosing of the series.

Matlab codes are attached to the zip files.