



Sheet 1 - Error Analysis

CSE 213 - Numerical Analysis

120200033

CSE Section 01

Ahmad Mongy Saad Aboelnaga

Ahmad.Aboelnaga@ejust.edu.eg

Question 1: Absolute and Relative Errors: If x is the exact value, and x^* is the approximate value, calculate the absolute and relative errors when:

$$a. \quad x = 10.147, x^* = 10.159$$

$$b. \quad x = 0.0047, x^* = 0.0045$$

$$c. \quad x = 671000, x^* = 669000$$

Solution:

$$\text{Absolute Error} = | \text{Exact value } (x) - \text{Approximate Value } (x^*) |$$

$$\text{Relative Error} = \frac{| \text{Exact value } (x) - \text{Approximate Value } (x^*) |}{\text{Exact value } (x)}$$

a.

$$\text{Absolute Error} = |10.147 - 10.159| = 0.012 \quad (1)$$

$$\text{Relative Error} = \frac{|10.147 - 10.159|}{10.147} \approx 1.183 \times 10^{-3} \quad (2)$$

b.

$$\text{Absolute Error} = |0.0047 - 0.0045| = 0.0002 \quad (3)$$

$$\text{Relative Error} = \frac{|0.0047 - 0.0045|}{0.0047} \approx 4.255 \times 10^{-2} \quad (4)$$

c.

$$\text{Absolute Error} = |671000 - 669000| = 2000 \quad (5)$$

$$\text{Relative Error} = \frac{|671000 - 669000|}{671000} \approx 2.98 \times 10^{-3} \quad (6)$$

Question 2: Truncation Error: Use only the first 3 terms of the series

$$\sin(x) = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

to evaluate $\sin(\frac{\pi}{2})$ and then find the truncation error.

Solution:

The Exact Value of $\sin(\frac{\pi}{2}) = 1$

Using only the first 3 terms of the series we get:

$$\begin{aligned} \sin\left(\frac{\pi}{2}\right) &= (-1)^0 \frac{\frac{\pi^{2 \times 0 + 1}}{2}}{(2 \times 0 + 1)!} + (-1)^1 \frac{\frac{\pi^{2 \times 1 + 1}}{2}}{(2 \times 1 + 1)!} + (-1)^2 \frac{\frac{\pi^{2 \times 2 + 1}}{2}}{(2 \times 2 + 1)!} \\ &= + 1.5707963 \quad - 0.6459641 \quad + 0.0796926 \\ &= + 1.0045248 \end{aligned}$$

Truncation Error:

$$R_3(X) = |1 - 1.0045248| = 0.0045248 \quad (\text{Absolute Error})$$

$$\therefore \text{Truncation Error} = \frac{0.0045248}{1} = 0.45248 \times 10^{-2} \quad (\text{Relative Error})$$

Question 3: Hypothetical Machine: Consider a hypothetical digital computer in which the binary representation of a real number is: $(1)(001)(11010)$, where (001) corresponds to the Exponent, and (11010) corresponds to the Mantissa. Find:

- The values of the parameters L, U, p, and b.
- The representation (in base 10) of the given binary representation.
- The absolute error resulting from the operation: $1.75 - 0.1875$
- The Maximum and Minimum decimal values that can be represented by the given machine.
- How many different numbers are in this floating point system?

Solution:

a)

$$L = -(11)_2 = -(3)_{10} \quad (\text{one bit for sign and two bit for the value}),$$

$$U = +(11)_2 = +(3)_{10} \quad (\text{Upper limit}),$$

$$b = 2 \quad (\text{Binary Base}),$$

$$p = 5 \quad (\text{Length of mantissa})$$

b)

$$(\textcolor{red}{1}\textcolor{green}{00}\textcolor{blue}{111010})$$

$$\begin{array}{ccccccc} \textcolor{red}{\downarrow} & \textcolor{green}{\downarrow} & \textcolor{green}{\downarrow} & \textcolor{green}{\downarrow} & \textcolor{blue}{\downarrow} & \textcolor{blue}{\downarrow} & \textcolor{blue}{\downarrow} & \textcolor{blue}{\downarrow} \end{array}$$

$$= \textcolor{green}{2}^1 \times (\textcolor{blue}{2}^{-1} + \textcolor{blue}{2}^{-2} + \textcolor{blue}{2}^{-4})$$

$$= (-1.625)_{10}$$

c)

$$x = 1.75 - 0.1875 = 1.5625 \quad (\text{The exact value})$$

$$x^* = \text{fl}(\text{fl}(1.75) - \text{fl}(0.1875)) = 1.5625 \quad (\text{The machine value})$$

$$\therefore \text{Absloute Error} = |x - x^*| = 0$$

$\text{fl}(1.75) \rightarrow (0)(001)(11100)$, has an exact representation

$\text{fl}(0.1875) \rightarrow (0)(110)(11000)$, has an exact representation

$\text{fl}(1.5625) \rightarrow (0)(001)(11001)$, has an exact representation

d)

Min -ve Value at (1)(011)(11111)=-7.75,

Min +ve Value at (0)(111)(10000)=+0.0625,

Max +ve Value at (0)(011)(11111)=+7.75

e) Possible combinations in

- sign-bit =2 [(0),(1)]
- exponent =7 [(000),(001),(010)...
- mantissa =16 [(10000),(10001)...

Total possible numbers that can be represented in this machine

$$= 2 \times 7 \times 16$$

= 224 different numbers

Question 4: Verify that the chopping error resulting from calculating $(1.0/6.0)$ to 2 significant digits will cause that $(1.0/6.0) \times 6.0 \neq 1.0$. Verify that when $(1.0/6.0)$ is approximated to 5 decimal digits, the result of $(1.0/6.0) \times 6.0$ will be 0.99996

Solution:

$$\frac{1.0}{6.0} \approx 0.16 \text{ chopped at 2 significant digits}$$

$$\therefore (1.0/6.0) \times 6.0 \rightarrow 0.16 \times 6.0 = 0.96 \neq 1.0$$

$$\frac{1.0}{6.0} \approx 0.16666 \text{ chopped at 5 significant digits}$$

$$\therefore (1.0/6.0) \times 6.0 \rightarrow 0.16666 \times 6.0 = 0.99996$$

Question 5: The critical part of floating-point operations is the potential loss of correct digits in the significand. Consider a computer model that uses 4 digits for the significand, 1 digit for the exponent plus an optional sign for both the significand and the exponent, any real number 'a' may be written as a normalised decimal number $a \times 10^n$, where the number 'a' is in the range $[0.1, 1)$ and is called the significand, while the integer n is called the exponent.

Convert the following numbers to normal form and perform the required operation:

- 1) $a = 42.34$ and $b = 0.0033$, calculate $a + b$
- 2) $a = 10.34$ and $b = -10.27$, calculate $a + b$
- 3) $a = 10/7$ and $b = -1.42$, calculate $a + b$
- 4) $a = 23.57$ and $b = -6.759$, calculate $a * b$ and a/b

Solution:

$$\begin{aligned} 1) \quad a &= 42.34, \quad fl(a) \rightarrow 0.4234 \times 10^2 \\ b &= 0.0033, \quad fl(b) \rightarrow 0.3300 \times 10^{-2} \\ a + b &= 0.423433 \times 10^2, \quad fl(a + b) \rightarrow 0.4234 \times 10^2 \text{ (Normalized)} \end{aligned}$$

$$\begin{aligned} 2) \quad a &= 10.34, \quad fl(a) \rightarrow 0.1034 \times 10^2 \\ b &= -10.27, \quad fl(b) \rightarrow -0.1027 \times 10^2 \\ a + b &= 0.0007 \times 10^2 \quad fl(a + b) \rightarrow 0.7000 \times 10^{-1} \text{ (Normalized)} \end{aligned}$$

$$\begin{aligned} 3) \quad a &= 10/7, \quad fl(a) \rightarrow 0.1428 \times 10^1 \text{ chopped or } 0.1429 \times 10^1 \text{ rounded} \\ b &= -1.42, \quad fl(b) \rightarrow -0.1420 \times 10^1 \\ a + b &\approx 8.571428 \times 10^{-3}, \quad fl(a + b) \rightarrow 0.8 \times 10^{-2} \text{ chopped or } 0.9 \times 10^{-2} \text{ rounded} \end{aligned}$$

$$\begin{aligned} 4) \quad a &= 23.57, \quad fl(b) \rightarrow 0.2375 \times 10^2 \\ b &= -6.759, \quad fl(b) \rightarrow -0.6759 \times 10^1 \\ a * b &= -1.5931 \times 10^2 \quad fl(a * b) \rightarrow -0.1593 \times 10^3 \text{ (Normalized by rounding)} \\ a/b &= -0.34872 \times 10^1 \quad fl(a/b) \rightarrow -0.3487 \times 10^1 \text{ (Normalized)} \end{aligned}$$

Question 6: Consider the simple algorithm:
 $x = 0.0;$
while $x \neq 1.0$
print x
 $x = x + 0.1;$

Implement the algorithm in a program and check what values of x will be printed? Explain what happened.

Solution:

The screenshot shows a Python IDE with a file named 'untitled0.py' containing the following code:

```

1 x=0
2 while x!=1:
3     print(x)
4     x=x+0.1

```

The console output shows the values of x printed by the program. The output is as follows:

```

Desktop')
0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
1.5000000000000002
1.6000000000000003
1.7000000000000004
1.8000000000000005
1.9000000000000006
2.0000000000000004
2.1000000000000005
2.2000000000000006
2.3000000000000007
2.4000000000000001
2.5000000000000001
2.6000000000000001
2.7000000000000001
2.8000000000000001
2.90000000000000012
3.00000000000000013
3.10000000000000014
3.20000000000000015

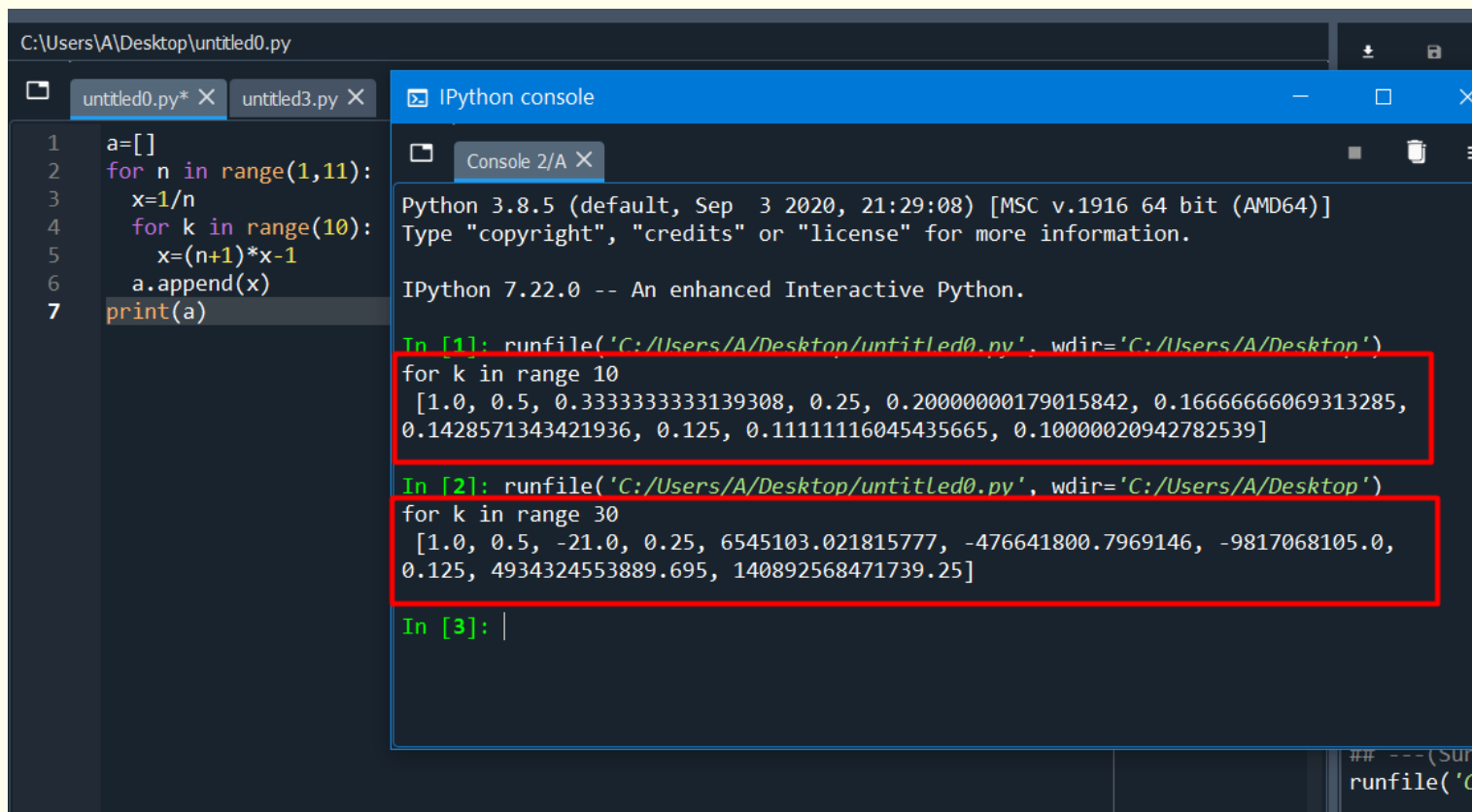
```

As 0.1 does not have a finite representation in binary, there is an approximation error and it shows for example in 0.3 as the machine prints 0.30000000000000004 instead. Due to the error of finite representation an exact 1 is never reached and there for the program enters an infinite loop and continue printing numbers that is greater than 1.

Question 7: The following Python program shows the effect of the propagation of an initial representation error which results in final (large!) error. Run the program, print its results once for k in $\text{range}(10)$ and in $\text{range}(30)$. Comment on the obtained results.

```
a=[]  
for n in range(1,11)  
    x= 1/n  
    for k in range(30):  
        x=(n+1)*x-1  
    a.append(x)  
print(a)
```

Solution:



```
C:\Users\A\Desktop\untitled0.py  
untitled0.py* X untitled3.py X  
1 a=[]  
2 for n in range(1,11):  
3     x=1/n  
4     for k in range(10):  
5         x=(n+1)*x-1  
6     a.append(x)  
7 print(a)  
  
IPython console  
Console 2/A X  
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]  
Type "copyright", "credits" or "license" for more information.  
IPython 7.22.0 -- An enhanced Interactive Python.  
  
In [1]: runfile('C:/Users/A/Desktop/untitled0.py', wdir='C:/Users/A/Desktop')  
for k in range 10  
[1.0, 0.5, 0.3333333333139308, 0.25, 0.20000000179015842, 0.16666666069313285,  
0.1428571343421936, 0.125, 0.11111116045435665, 0.10000020942782539]  
  
In [2]: runfile('C:/Users/A/Desktop/untitled0.py', wdir='C:/Users/A/Desktop')  
for k in range 30  
[1.0, 0.5, -21.0, 0.25, 6545103.021815777, -476641800.7969146, -9817068105.0,  
0.125, 4934324553889.695, 140892568471739.25]  
  
In [3]: |
```

As there are approximation errors due to machine finite representation, these errors propagate through the iterations and therefore present a solution with larger error, and this can be observed for all numbers that does not have an exact representation in the machine

Question 8: Errors in Computer Arithmetic

In order to reduce the chance of large errors occurring in calculations, here are some tips that you should consider:

i. In general, it is better to add ‘floating-point’ numbers in order of magnitude if possible. For example, suppose we compute $0.273000 + 0.001480 + 0.000862 (= 0.275342)$. If results are stored with mantissas 3 digits long; then,

$$0.273000 + 0.001480 = 0.274480 \rightarrow 0.274000 \text{ and}$$

$$0.27400 + 0.000862 = 0.274862 \rightarrow 0.274000$$

whereas

$$0.000862 + 0.001480 = 0.002342 \rightarrow 0.002340 \text{ and}$$

$$0.002340 + 0.27300 = 0.275340 \rightarrow 0.275000$$

ii. In order to prevent ‘loss of significance’, it is important to avoid a ‘bad subtraction’ that is, a subtraction of a number from another number having almost equal value. Let us consider a simple problem of finding the roots of a second-order equation $ax^2 + bx + c = 0$ by using the formulas:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Assuming the Discriminant: $b^2 - 4ac > 0$, then depending on the sign of b , a “bad subtraction” may be encountered when we try to find the smaller one of the two roots. This implies that it is safe, from the “loss of significance” point of view, to compute the root having the larger absolute value first and then obtain the other root by using the relation (between the roots and the coefficients) $x_1 x_2 = c/a$.

a) Apply this technique to solve the equation $x^2 + 62.10x + 1 = 0$ whose roots are Approximately : $x_1 = -0.01610723$ and $x_2 = -62.08390$. Check the relative errors in the obtained roots (using four-digit rounding arithmetic) in case of finding both roots using the formulas and the case of avoiding ”bad subtraction”.

b) Apply the idea of rationalizing the numerator of the quadratic formula to obtain a more accurate four-digit rounding approximation for x_1

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \left(\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} \right) = \frac{b^2 - (b^2 - 4ac)}{2a(-b - \sqrt{b^2 - 4ac})}$$

$$= \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

iii. In consecutive multiplication/division processes, make the intermediate result as close to one as possible. According to this rule, when computing xy/z , we program the formula as:

- $(xy)/z$ when x and y in the multiplication are very different in magnitude,
- $x(y/z)$ when y and z in the division are close in magnitude, and
- $(x/z)y$ when x and z in the division are close in magnitude.

For instance, when computing y^n/e^{nx} with $x > 1$ and $y > 1$, we would program it as $(y/e^x)^n$ rather than as y^n/e^{nx} , so that overflow/underflow can be avoided. Verify this using Excel/ Python/ Matlab for the case:
 $x = 36$; $y = 1e16$; for $n = -20, -19, 19, 20$.

Solution:

a & b) $a = 1$, $b = 62.10$, $c = 1$

$a = 0.01 \times 10^2$, $b = 0.6210 \times 10^2$, $c = 0.100 \times 10^2$

$x_2 = -62.08$, using root formula

For x_1 :

Method	Using root Formula	Avoid Bad Subtrac	Ration Numerator
X_1 Value	-0.02×10^0	-0.01611	-0.01611
Abs Error(e)	3.89277×10^{-3}	2.77×10^{-6}	2.77×10^{-6}
Relative Error(r)	0.24168	1.7197×10^{-4}	1.7197×10^{-4}

iii) As show in the figure in next page, the result of using both algorithms have yielded close results with one of them having more accurate calculation each time, moreover, in the case with $n=20$ the second method (y^n/e^{nx}) has resulted in Over Flow Error, while the other method ($(y/e^x)^n$) returned a result

```

1  import math
2  e= math.e
3  x=36
4  y=1e16
5  dit={}
6  print(["n"],[" (y/(e**x))**n  "],["(y**n)/(e**n)**x"])
7  for n in [-20,-19,19,20]:
8      print([n],end="")
9      print([(y/(e**x))**n],end="")
10     print([(y**n)/(e**n)**x])

```

IPython console

Console 2/A X

```

In [15]: runfile('C:/Users/A/Desktop/untitled0.py', wdir='C:/
Users/A/Desktop')
['n'] [' (y/(e**x))**n  '] ['(y**n)/(e**n)**x']
[-20][4.9207009302636215e-08][4.920646149013654e-08]
[-19][1.1413678148547262e-07][1.1413678148547305e-07]
[19][8761417.546431169][8761417.54643118]
[20][20322308.024243735]Traceback (most recent call last):

File "C:\Users\A\Desktop\untitled0.py", line 10, in <module>
    print([(y**n)/(e**n)**x])

OverflowError: (34, 'Result too large')

```

Tools used in creating this document:

- Texmaker 5.0.4
- Spyder IDE 5.1.5 with python 3.8.5

©All questions in this file has been adapted from the sheet provided by the course instructor, I have only reorganized the sheet and attempted to answer it. Please do not share without the permission of the owner.

Thanks for my colleagues who helped me reviewing the file

Ahmad.Aboelnaga@ejust.edu.eg

ID:120200033, CSE01