Laboratorium 02

Temat: Błędy metod numerycznych - źródła i rodzaje.

Celem ćwiczenia jest zapoznanie się ze źródłami błędów metod numerycznych oraz ich rodzajami. Podstawowy problem dotyczący dokładności obliczeń z wykorzystaniem metod numerycznych wynika z konieczności zapisu liczb w postaci ich reprezentacji komputerowej. Reprezentacja komputerowa liczb zależy przede wszystkim od ich typu:

- liczby całkowite, np. 1, 3, itp: są reprezentowane w postaci zapisu binarnego na 32 lub 64 bitach,
- liczby zmiennoprzecinkowe, np. 1.10, 3.14: są reprezentowane z wykorzystaniem trzech liczb binarnych zapisanych na 64 bitach,
- liczby zespolone: składają się z dwóch liczb zmiennoprzecinkowych.

Liczby w postaci cyfrowej są kodowane z wykorzystaniem dwóch napięć, tj. niskiego (np. 0V) i wysokiego (np. 5V). Zatem do dyspozycji mamy tylko dwie liczby tj, 0 i 1, które określamy nazwą bitu (z ang. bit to **bi**nary **d**igit). Ze względu na to, że ludzie posługują się systemem dziesiętnym (10 cyfr), a w komputerze posługujemy się systemem dwójkowym (2 liczby), istnieje konieczność zamiany liczb pomiędzy systemami. Zamiana liczb z systemu dwójkowego na dziesiętny jest stosunkowo prosta i wymaga jedynie sumowania kolejnych potęg liczby 2 z odpowiednim współczynnikiem odpowiadającym wartości poszczególnych bitów

$$\sum_{i=0}^{n} b_i 2^i$$

gdzie współczynniki b_i są kolejnymi liczbami zapisu dwójkowego, np.

1	**0**	**0**	**1**

$1001 => 1*2^0+0*2^1+0*2^2+1*2^3 = 1+0+0+8 = 9$

W tabeli przedstawiono wartości liczb dla systemu dziesiętnego oraz ich odpowiedniki w systemie dwójkowym.

Binary	1	10	11	100	101	110	111	1000	1001
Decimal	1	2	3	4	5	6	7	8	9

O ile zapis liczb całkowitych jest stosunkowo prosty to zapis liczb zmiennoprzecinkowych sprawia pewne problemy i wymaga zastosowania znaku rozdzielającego części przed i po przecinku:

bn···b0 . b-1b-2···

gdzie m<n. Ponieważ liczby w komputerze są przechowywane zazwyczaj w komórkach o określonej wielkości, to istnieją pewne ograniczania na ich wielkość, jak i dokładność obliczeń. Informację o właściwościach liczb dostępnych w języku Python w wersji 2.7 można uzyskać korzystając z modułu „sys” oraz odpowiednich metod:

- liczby całkowite (int): sys.maxint

```
>>> import sys
>>> print sys.maxint
9223372036854775807
```

- liczby zmiennoprzecinkowe (float): sys.float_info

```
>>> import sys
>>> print sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=
2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

w przypadku Pythona w wersji 3.6:

- liczby całkowite (int): sys.int_info

```
>>> import sys
>>> print(sys.int_info)
sys.int_info(bits_per_digit=30, sizeof_digit=4)
```

- liczby zmiennoprzecinkowe (float): sys.float_info

```
>>> import sys
>>> print(sys.float_info)
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=
2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

W przypadku liczb całkowitych ich maksymalną liczbę całkowitą można wyznaczyć z zależności:

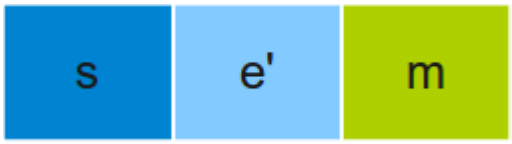
$$liczba\ \ maksymalna = 2^{liczba\ \ bitów - 1} - 1$$

$$2^{63} - 1 = 9223372036854775807$$

gdzie liczba bitów wynosi 64.

Zapis liczb zmiennoprzecinkowych wg standardu IEEE-754 wymaga trzech pól, tzn.:

- bitu określającego znak liczby (sign)
- pola bitów przechowujących wykładnik (exponent)
- pola bitów przechowujących mantysę (mantissa)



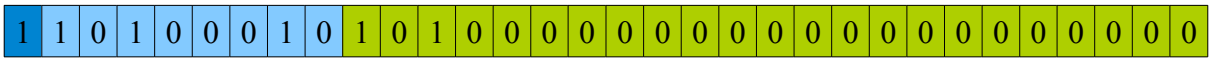
Zamiana zawartości pól na zmiennoprzecinkową liczbę binarną odbywa się wg formuły:

$$liczba = (-1)^s \times (1.\mathrm{m}) \times 2^{e' - 127}$$

Standard IEEE-754 przewiduje między innymi możliwość zapisu liczb zmiennoprzecinkowych z wykorzystaniem:

- 32 bitów [s=>1 bit, e'=>8 bitów, m=>23 bity]
- 64 bitów [s=>1 bit, e'=>11 bitów, m=>52 bity]

np.



Zatem wartość tak zapisanej liczby zmiennoprzecinkowej możemy zapisać jako:

$$(-1)^1*(1.10100000000000000000000)*2^{10100010-(127)}$$

Co w zapisie dziesiętnym odpowiada liczbie:

$$(-1)*(1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} + ...)^{(162-127)} = (-1)*(1 + 0.625)*2^{35} = -5.5834*10^{10}$$

Niestety konsekwencją takiego zapisu jest ograniczona rozdzielczość zapisu liczb zmiennoprzecinkowych określana terminem „machine epsilon". Liczbę tą wyznaczamy jako różnicę pomiędzy wartością 1 a następną najmniejszą liczbą, jaką jesteśmy w stanie zadeklarować/zapisać. I tak, w zależności od przyjętego standardu zapisu liczb, błąd jest następujący:

- liczby zmiennoprzecinkowe 32 bitowe => epsilon= $2^{-23} = 1.19 \times 10^{-7}$
- liczby zmiennoprzecinkowe 64 bitowe => epsilon= $2^{-52} = 2.22 \times 10^{-16}$

Inne rodzaje błędów związane są z problemem zaokrąglania i ucinania. Przykładem błędu obliczeń numerycznych może być np. wyznaczenie sumy liczby 0.1 powtórzonej 1 milion razy. Przykładowy kody w przypadku Pythona w wersji 2.7 (DrPython):

```
1  from decimal import Decimal
2  liczba=0.1
3  suma=0
4  zakres=1000000
5  for i in range(0,zakres):
6      suma=suma+liczba
7  print "Wartosc prawdziwa=", zakres/10
8  print "Wartosc obliczona=" "%5.10f" % (suma)
9  print "Blad obliczen [%]=", 100*(suma-(zakres/10))/(zakres/10)
10 print liczba,"=",Decimal(liczba)
```

lab02a.py

```
/usr/bin/python -u  "/home/artur/Pulpit/Metody numeryczne/Kody 2.7/lab02a.py"
Wartosc prawdziwa= 100000
Wartosc obliczona=100000.0000013329
Blad obliczen [%]= 1.33288267534e-09
0.1 = 0.1000000000000000055511151231257827021181583404541015625
```

oraz w wersji 3.6 (Thonny):

```
1  from decimal import Decimal
2  liczba=0.1
3  suma=0
4  zakres=1000000
5  for i in range(0,zakres):
6      suma=suma+liczba
7  print("Wartosc prawdziwa=", zakres/10)
8  print("Wartosc obliczona=" "%5.10f" % (suma))
9  print("Blad obliczen [%]=", 100*(suma-(zakres/10))/(zakres/10))
10 print(liczba,"=",Decimal(liczba))
```

Shell

```
>>> %Run lab02a.py

  Wartosc prawdziwa= 100000.0
  Wartosc obliczona=100000.0000013329
  Blad obliczen [%]= 1.3328826753422618e-09
  0.1 = 0.1000000000000000055511151231257827021181583404541015625
```

gdzie funkcja Decimal() z biblioteki decimal zwraca dokładną wartość liczby typu float przechowywaną w pamięci.

Ćwiczenia:

1. Napisz program do zamiany liczb z systemu dwójkowego na system dziesiętny, przyjmując założenia, że korzystamy jedynie z komórki pamięci o wielkości 1 bajta tj. 8 bitów.
2. Sprawdź i wyjaśnij zachowanie języka Python w wersji 2.7 oraz 3.6 dla przypadku wykonania następujących operacji:
   - dodanie 1 do wartości maksymalnej typu „int”,
   - dwukrotne dodanie wartości maksymalnej typu „float”.

3. Wykonaj wykres zależności błędu tzw. machine epsilon w zależności od ilości bitów przeznaczonych na zapis mantysy, przyjmując założenie, że liczba bitów dla wykładnika wynosi 8, a liczba bitów dla mantysy jest zmienna i może przyjmować wartości od 8 do 52.

4. Zaproponuj / wyszukaj algorytm, który pozwoli uniknąć błędu obliczeń związanego z tzw. machine epsilon, np. w przypadku banków, itp.
5. Oszacuje ilość liczb typu float pomiędzy w zakresie 0-1.

Uwagi:

- funkcja type(zmienna) – zwraca/wyświetla typ zmiennej, np.

Program:

```
a=1
print type(a) lub print(type(a))
```

Wynik:

```
<type 'int'>
```

- zmiana skali liniowej na logarytmiczną

Program:

```
...
plt.yscale("log")
```

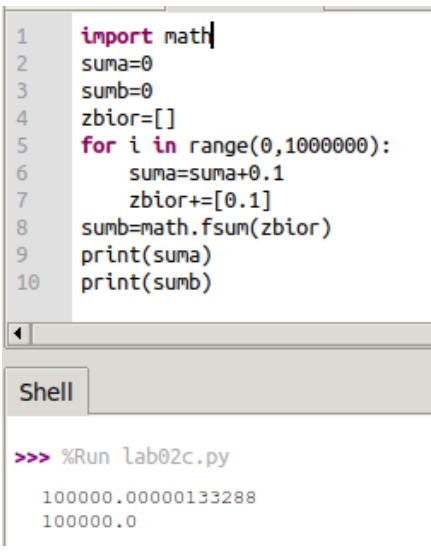
- suma zbioru liczb bez udziału błędu, np. => *math.fsum*

```
1  import math
2  suma=0
3  sumb=0
4  zbior=[]
5  for i in range(0,1000000):
6      suma=suma+0.1
7      zbior+=[0.1]
8  sumb=math.fsum(zbior)
9  print(suma)
10 print(sumb)
```

Shell

```
>>> %Run lab02c.py

  100000.00000133288
  100000.0
```


1 import math
2 suma=0
3 sumb=0
4 zbior=[]
5 for i in range(0,1000000):
6 suma=suma+0.1
7 zbior+=[0.1]
8 sumb=math.fsum(zbior)
9 print(suma)
10 print(sumb)
Shell
>>> %Run lab02c.py
100000.00000133288
100000.0