AULA 2

Números e operações. Erros de arredondamento e de truncatura. Gráficos simples. Cópia de objetos.

Números inteiros

Plain int: 32 bits, representa os números inteiros:

$$-2^{31} < N < +2^{31} - 1$$

A representação é **exata**, dentro do **alcance** representado.

Operações entre inteiros dão, em geral, resultado inteiro. Em PYTHON3, a divisão é tratada de forma especial:

2/3 >> 0.66(6) (convertido em float)

2//3 >> 0 (divisão inteira)

Números float

$$x = \pm m \times 2^{e-E}$$

m mantissa (**algarismos sinificativos**)

e expoente, E viés (**alcance**)

$$m, e, E \in \mathbb{Z}$$

Os números ***float*** representam um sub-conjunto dos racionais

Em 8 bytes (64 bits, 16 algarismos significativos): (para além do "0" e do sinal)

- Min float: 2.2250738585072014e-308
- Max float: 1.7976931348623157e+308

Operações aritméticas

Operações elementares: `+ - * /`

Potência: `**`

Divisão inteira: `//`

As operações envolvem objetos (números) que podem ser de **tipo** diferente. O tipo de resultado depende do tipo dos números operados. Em geral o python dá o resultado no tipo de maior **rank**:

```
X=10 #inteiro
Y=2.5 #float
Z=X*Y #float
c=1+4j #complex (1+4i)
```

BigInt (só para inteiros escalares)

In[102]: x=2**128

In[103]: x

Out[104]: 340282366920938463463374607431768211456

(inteiro de 128 bit)

Y=2**1280

(inteiro de 1280 bit)

Mas:

z=2.0**1280 #OVERFLOW error

a=np.array([10]);b=a**10000 #OVERFLOW error em numpy

Erros

Overflow: fora do alcance

Erro de arredondamento:

In[116]: 1/3

Out[117]: 0.3333333333333333

Erro de truncatura: depende das aproximações do Código.

Instabilidade numérica resulta da interação entre o erro de arredondamento e o erro de truncatura.

Erros em funções: $\sqrt{-2} = i\sqrt{2}$

```
import math; import numpy
print((-2.)**0.5)
```

>>(8.659560562354934e-17+1.4142135623730951j)

```
math.sqrt(-2.)
```

>>ValueError: math domain error

```
numpy.sqrt(-2.)
```

__main__:1: RuntimeWarning: invalid value encountered in sqrt

```
numpy.sqrt(-2+0j)
```

>>1.4142135623730951j

Exemplo: Resolver $ay^2 + by + c = 0$

Parece fácil:

$$y = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Mas se $b^2 \gg 4ac$ há problemas… Exemplo:

$$-y^2 + 88550000y - 1 = 0$$

Solução exata (8 algarismos):

$$\begin{cases} y_1 = 88550000 \\ y_2 = 1.1293055 \times 10^{-8} \end{cases}$$

$$-y^2 + 88550000y - 1 = 0$$

```
from math import sqrt
a=-1.;b=88550000.;c=-1.;
y1=(-b-sqrt(b*b-4*a*c))/(2*a)
y2=(-b+sqrt(b*b-4*a*c))/(2*a)
print('y1=',y1,'y2=',y2)
```

```
y2= 88549999.99999999
y1= 1.4901161193847656e-08
```

Solução:

$$\begin{cases} y_1 = 88550000 \\ y_2 = 1.1293055 \times 10^{-8} \end{cases}$$

Erro de 30%!

Fórmula alternativa

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

```
y1a=2*c/(b+sqrt(b*b-4*a*c))
y2a=2*c/(b-sqrt(b*b-4*a*c))
print('y1=',y1a,'y2=',y2a)
```

```
y1= 67108864.0
y2= 1.1293054771315643e-08
```

Erro na outra raíz

Solução:

$$\begin{cases} y_1 = 88550000 \\ y_2 = 1.1293055 \times 10^{-8} \end{cases}$$

Solução exata

```
from sympy import Symbol,solve
y=Symbol('y')
a=-1.;b=88550000.;c=-1.;
solution=solve(a*y**2+b*y+c,y) #y: f(y)=0
print('sympy solution=',solution)
```

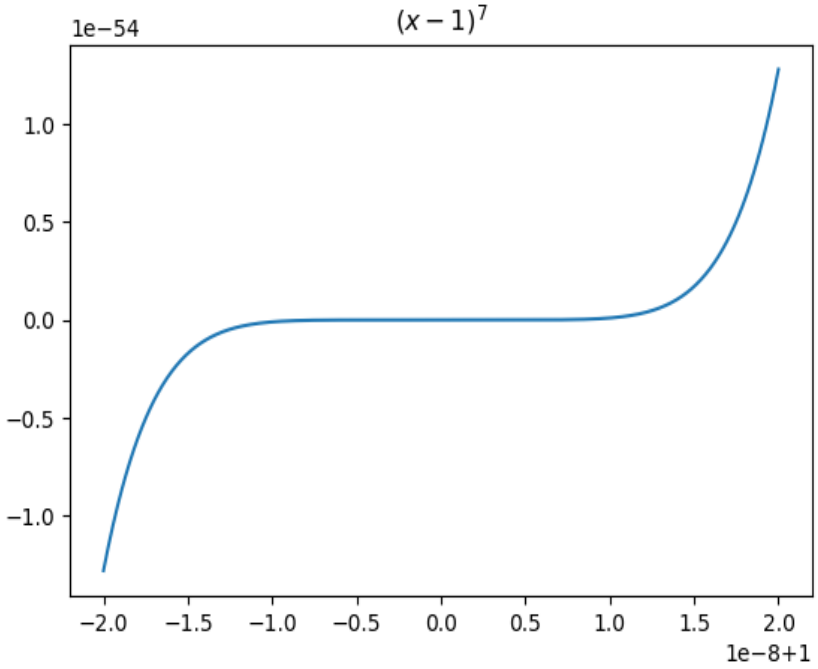
```
sympy solution= [1.12930547713156e-8,
88550000.0000000]
```

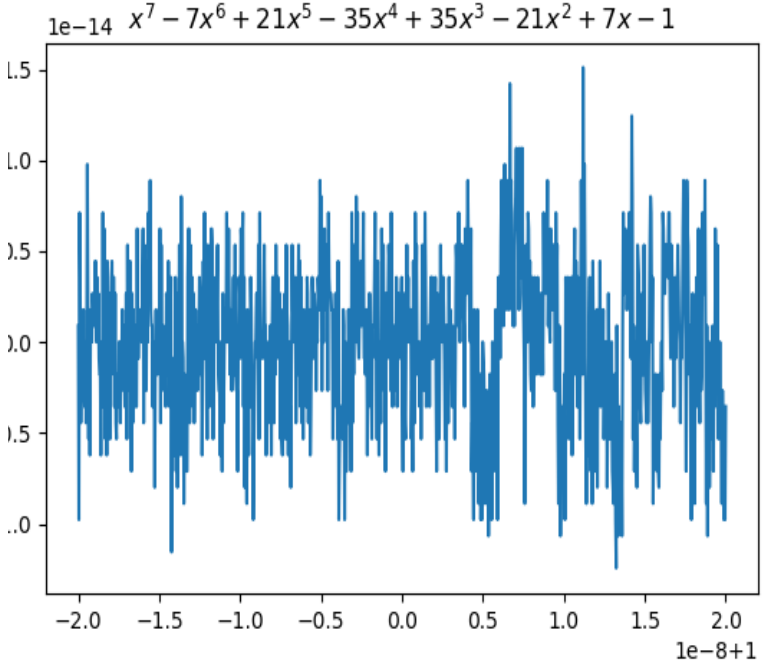
sympy.solve; print formatado

```
from math import sqrt;from sympy import Symbol,solve;
y=Symbol('y')
a=-1.;b=88550000.;c=-1.;
y1=(-b-sqrt(b*b-4*a*c))/(2*a)
y2=(-b+sqrt(b*b-4*a*c))/(2*a)
print('y1=%18.9e y2=%18.9e' % (y1,y2))
solution=solve(a*y**2+b*y+c,y)
print('sympy solution=%18.9e %18.9e' \
% (solution[0],solution[1]))
>>y1=   8.855000000e+07 y2=   1.490116119e-08
sympy solution=   1.129305477e-08 8.855000000e+07
```

Mais erros… perda de algarismos significativos em operações sucessivas

$$(x-1)^7 = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$





Erro de 40 ordens de grandeza!

Perda de algarismos significativos …

```
(1-x)**7
9.9777777777777635e-01   1.0000000000000000e+00   2.676162024048363282e-19

x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x-1
x**7-7*x**6
9.8454776491274775 82e-01 -6.9071836513923283 72e+00 -5.9226358864795809 47e+00
x**7-7*x**6+21*x**5
-5.9226358864795809 47e+00   2.0767701401736399 45e+01   1.4845065515256818 50e+01
…
x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x
-5.9844444444444384 60e+00   6.9844444444444446 77e+00   1.0000000000000062 17e+00
x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x-1
1.0000000000000062 17e+00   1.0000000000000000 00e+00   6.2172489379008766 26e-15
```

Gráficos e anotações

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0.99999998,1.00000002,1001)
# x=np.arange(0.99999998,1.00000002+1e-11,4e-11)
f1=(x-1)**7
f2=x**7-7*x**6+21*x**5-35*x**4+35*x**3-21*x**2+7*x-1
plt.close('all')
plt.subplot(1,2,1)
plt.plot(x,f1)
plt.title(r"$(x-1)^{7}$")
plt.subplot(1,2,2)
plt.plot(x,f2)
plt.title(r"$x^{7}-7x^{6}+21x^{5}-35x^{4}+35x^{3}-
21x^{2}+7x-1$")
plt.savefig('func.png')
```

Cópia de objetos
a=b *vs* a=copy(b)

```
import copy
a=[1,2,3]
d=copy.copy(a)
x=a;x[0]=10;d[0]=100
print('a=',a,'d=',d,'\n','x=',x)
```

```
>>a= [10, 2, 3] d= [100, 2, 3]
  x= [10, 2, 3]
```

list *vs* array

```
import numpy as np
b=[1.,2.,3.]
y=b*2 #b é uma list
c=np.array(b)
z=c*2 #c é um array
print('y=',y,'z=',z)
```

```
>>y= [1.0, 2.0, 3.0, 1.0, 2.0, 3.0]
  z= [2. 4. 6.]
```

list *vs* array

```
import numpy as np
a=[1,2,3]
b=np.array(a)
a.append(7.) #só para list
c=np.array(a)
print(a,b,c)
>> [1, 2, 3, 7.0] [1 2 3] [1. 2. 3. 7.]
```

list array(int) array(float)