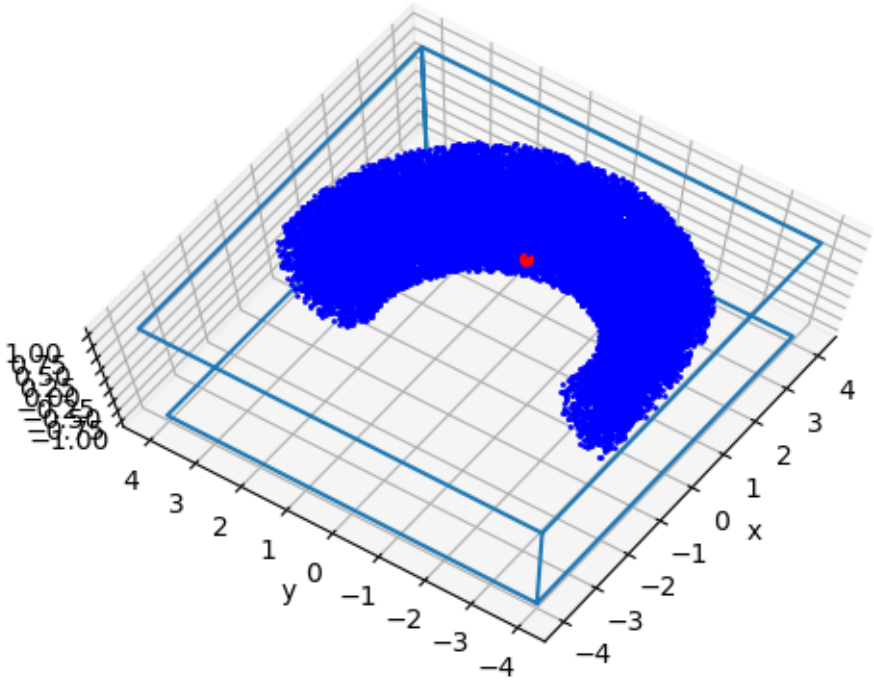
Aula 9

Integração numérica: método de Monte Carlo



Notas

Os métodos do ponto médio, do trapézio, e de Simpson, mimetizam o método analítico utilizando diferenças finitas

- Tal como o método analítico exigem uma definição de um elemento de integração (linear, área, volume) e de um domínio constituído por um conjunto desses elementos

Estes métodos, utilizando uma malha de pontos regularmente espaçada para avaliar numericamente o integral, são designados por métodos de quadratura de Newton-Cotes.

Uma outra família de métodos utiliza pontos de amostragem colocados de forma a otimizar o cálculo, podendo dar origem a cálculos exatos com um pequeno número de pontos: são os métodos de quadratura de Gauss.

Por vezes o mapeamento do domínio é muito díficil, especialmente se a geometria do domínio for complicada: nesse caso…

Teorema de Monte Carlo

Dado um volume V e N pontos ($\vec{x}_k$) aleatoriamente distribuídos nesse volume, com uma distribuição uniforme, tem-se:

$$\iiint_V f(x,y,z)\,dV \approx V\langle f\rangle \pm V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}}$$

Com

$$\langle f\rangle = \frac{1}{N}\sum_{k=1}^{N} f(x_k,y_k,z_k),\quad \langle f^2\rangle = \frac{1}{N}\sum_{k=1}^{N}[f(x_k,y_k,z_k)]^2$$

É aplicável com qualquer número de dimensões.

Se o volume V não for um paralelipípedo…

Escolhe-se um paralelipípedo P que contenha V. Define-se:

$$\iiint_V f\,dV = \iiint_P g\,dP,\ g(x) = \begin{cases} f(x), x \in V \\ 0, x \notin V \end{cases}$$

Cálculo de π a partir de $A = \pi R^2$

```
import numpy as np;import matplotlib.pyplot as plt
from math import pi;from mpl_toolkits.mplot3d import Axes3D
R=1;vol=[];errvol=[]; nns=np.arange(10,23);ns=2**nns
for n in ns:
    xmin=-R;xmax=R;xlen=xmax-xmin;
    ymin=-R;ymax=R;ylen=ymax-ymin;
    V=xlen*ylen; #volume do "paralelepípedo"
    Svol=0;Svol2=0;
    for k in range(n):
        X=xmin+xlen*np.random.rand();
        Y=ymin+ylen*np.random.rand();
        if X**2+Y**2<=R**2:
            Svol=Svol+1;
            Svol2=Svol2+1**2;
    Svol=Svol/n;Svol2=Svol2/n;
    volume=V*Svol;delvol=V*np.sqrt((Svol2-Svol**2)/n);
    print('n=%10i,area=%15.7e m^{2} +-%15.7e m^{2} (Err:%8.5f %%) ' \%
        (n,volume,delvol,delvol/volume*100))
```

$$A = \iint dA \ (f \equiv 1)$$

$$\langle f \rangle = \frac{1}{N} \sum_{k=1}^{N} f(x_k, y_k, z_k)$$

$$\langle f^2 \rangle = \frac{1}{N} \sum_{k=1}^{N} [f(x_k, y_k, z_k)]^2$$

Série 2^n

Como o método de Monte Carlo converge muito lentamente, vamos explorar o seu comportamento para números de amostra da série 2^n

```
nns=np.arange(10,23)
ns=2**nns
>> array([   1024,    2048,    4096,    8192,
16384,   32768,   65536, 131072,  262144,  524288,
1048576, 2097152, 4194304], dtype=int32)
```

π

```
n=      1024,area=  3.2070312e+00 m^2 +-  4.9834448e-02 m^2 (Err: 1.55391 %)
n=      2048,area=  3.1738281e+00 m^2 +-  3.5781779e-02 m^2 (Err: 1.12740 %)
n=      4096,area=  3.1494141e+00 m^2 +-  2.5573726e-02 m^2 (Err: 0.81202 %)
n=      8192,area=  3.1494141e+00 m^2 +-  1.8083355e-02 m^2 (Err: 0.57418 %)
n=     16384,area=  3.1145020e+00 m^2 +-  1.2974127e-02 m^2 (Err: 0.41657 %)
n=     32768,area=  3.1491699e+00 m^2 +-  9.0426245e-03 m^2 (Err: 0.28714 %)
n=     65536,area=  3.1322632e+00 m^2 +-  6.4399600e-03 m^2 (Err: 0.20560 %)
n=    131072,area=  3.1411133e+00 m^2 +-  4.5368538e-03 m^2 (Err: 0.14443 %)
n=    262144,area=  3.1416779e+00 m^2 +-  3.2072737e-03 m^2 (Err: 0.10209 %)
n=    524288,area=  3.1440659e+00 m^2 +-  2.2655885e-03 m^2 (Err: 0.07206 %)
n=   1048576,area=  3.1424866e+00 m^2 +-  1.6030875e-03 m^2 (Err: 0.05101 %)
n=   2097152,area=  3.1413460e+00 m^2 +-  1.1341018e-03 m^2 (Err: 0.03610 %)
n=   4194304,area=  3.1413069e+00 m^2 +-  8.0194433e-04 m^2 (Err: 0.02553 %)
```

Converge muito lentamente!

(notar que o erro é estimado… sem usar π)

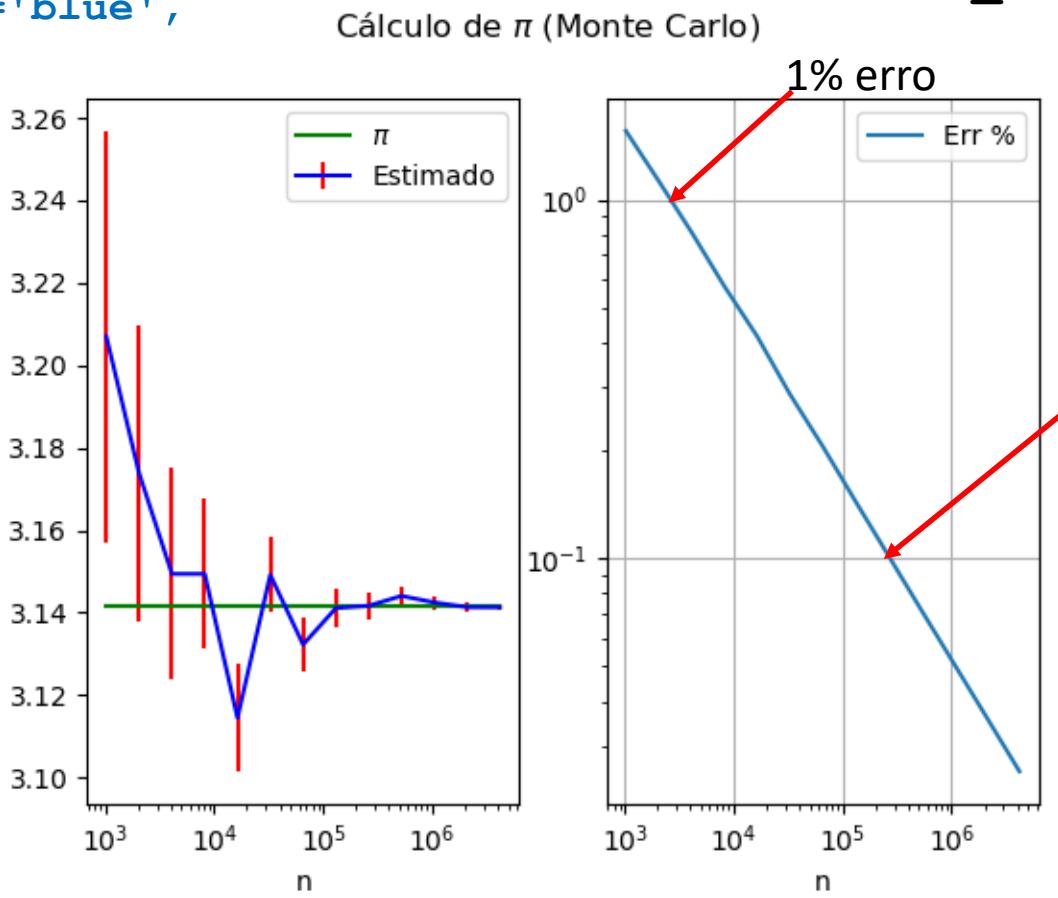
Cálculo de π: evolução do erro

```
…
    Svol=Svol/n;Svol2=Svol2/n;volume=V*Svol;
    delvol=V*np.sqrt((Svol2-Svol**2)/n);
    vol.append(volume)
    errvol.append(delvol)
plt.figure()
ax=plt.subplot(1,2,1); ax.set_xscale('log')
plt.errorbar(ns,vol,yerr=errvol,ecolor='red',color='blue',label='Estimado')
plt.plot([2**nns[0],2**nns[-1]],[pi*R**2,pi*R**2],\
        color='green',label=r'$\pi$')
plt.xlabel('n');plt.legend()
ax=plt.subplot(1,2,2);ax.set_xscale('log');ax.set_yscale('log')
plt.plot(ns,np.array(errvol)/np.array(vol)*100,label='Err %')
plt.grid();plt.legend();plt.xlabel('n')
plt.suptitle(r'Cálculo de $\pi$ (Monte Carlo)')
```

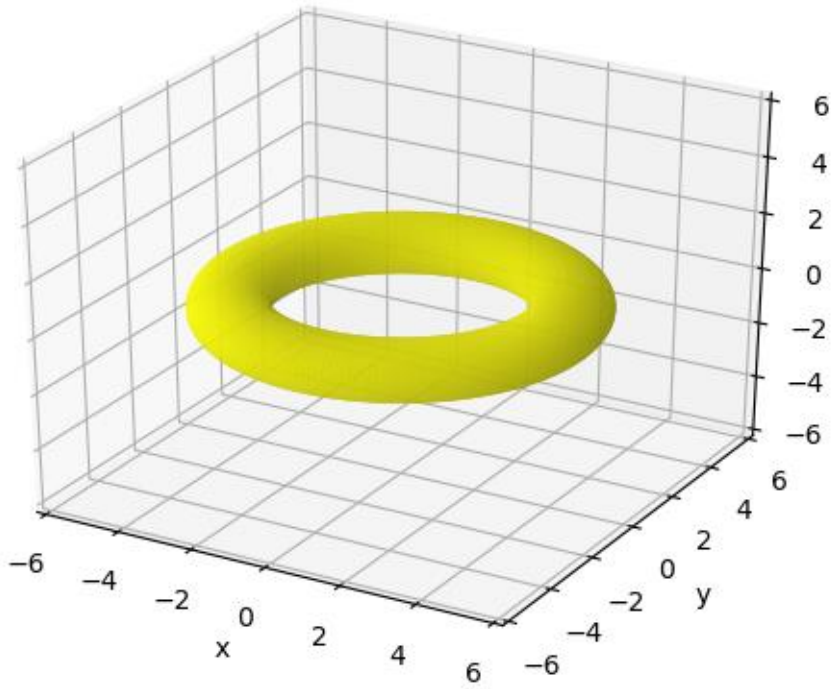
Cálculo de π, pelo método de Monte Carlo

```
errorbar(ns,vol,yerr=errvol
,ecolor='red',color='blue',
label='Estimado')
```

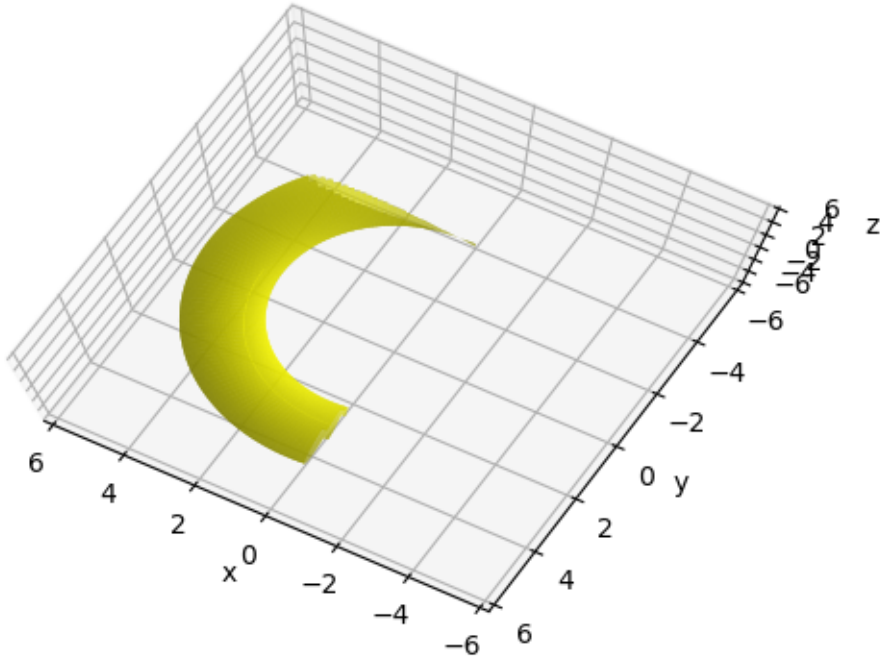
```
ax.set_xscale('log')
ax.set_yscale('log')
```



Exemplo2: calcular o centro de massa de um toro cortado (R=4,r=1,x>-0.5,y>-3)

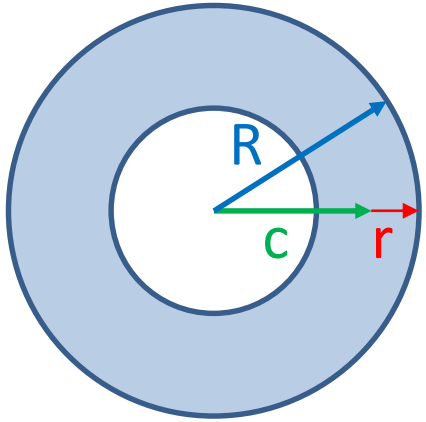


Fácil em coordenadas cilíndricas

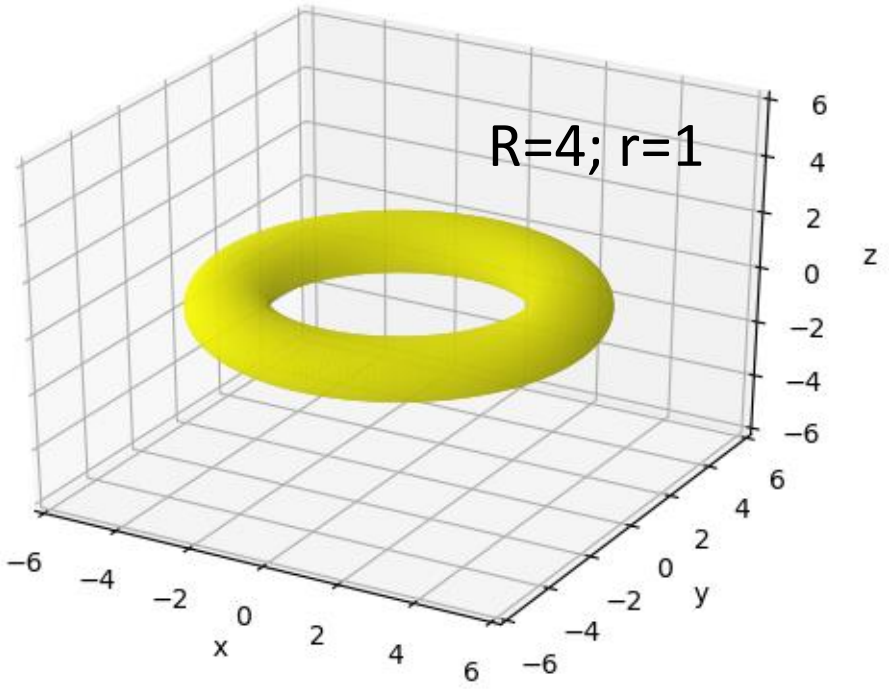


Muito difícil

Toro



$$Z^2 + \left(c - \sqrt{X^2 + Y^2}\right)^2 \leq r^2, c = R - r$$



Toro completo

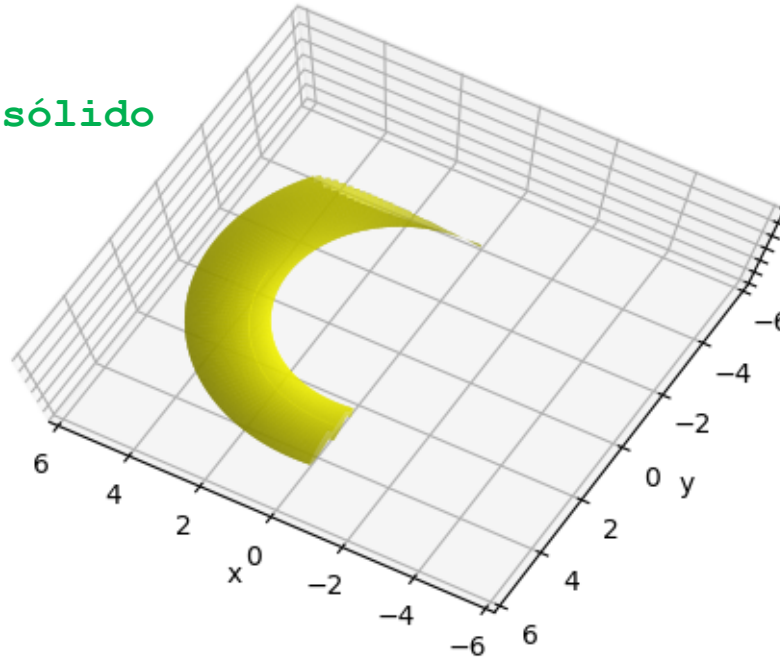
```
import numpy as np; import matplotlib.pyplot as plt
ro=1. # densidade=1 (massa=volume)
r=1; R=4; c=R-r
for n in [100000]:
    xmin=-R; xmax=R; xlen=xmax-xmin
    ymin=-R; ymax=R; ylen=ymax-ymin
    zmin=-r; zmax=r; zlen=zmax-zmin
    V=xlen*ylen*zlen; #volume do paralelepípedo
    Sro=0; Sro2=0 #só massa!
    for k in range(n):
        X=xmin+xlen*np.random.rand();
        Y=ymin+ylen*np.random.rand();
        Z=zmin+zlen*np.random.rand();
        if Z**2+(c-np.sqrt(X**2+Y**2))**2<=r**2:
            Sro=Sro+ro;  Sro2=Sro2+ro*ro;
Sro=Sro/n;Sro2=Sro2/n;massa=V*Sro;
delmas=V*np.sqrt((Sro2-Sro**2)/n);
```

Calculo do volume (ou massa com $\rho = 1 kgm^{-3}$)

```
n= 1 000
massa=  5.7216000e+01 kg +-2.0124556e+00 kg (Err: 3.517%)
n= 10 000
massa=  5.9648000e+01 kg +-6.3851861e-01 kg (Err: 1.070%)
n= 100 000
massa=  5.9086080e+01 kg +-2.0178834e-01 kg (Err: 0.341%)
n= 1 000 000
massa=  5.9292416e+01 kg +-6.3826630e-02 kg (Err: 0.107%)
n= 10 000 000
massa=  5.9214080e+01 kg +-2.0181910e-02 kg (Err: 0.034%)
```

Secção do toro

```
import numpy as np; import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
ro=1.2e3; r=1; R=4; c=R-r
xcut=-0.5;ycut=-3; # planos de corte do sólido
for n in [100000]:
    xmin=-R; xmax=R; xlen=xmax-xmin
    ymin=-R; ymax=R; ylen=ymax-ymin
    zmin=-r; zmax=r; zlen=zmax-zmin
    V=xlen*ylen*zlen; Sro=0; Sro2=0
    for k in range(n):
        X=xmin+xlen*np.random.rand();
        Y=ymin+ylen*np.random.rand();
        Z=zmin+zlen*np.random.rand();
        if Z**2+(c-np.sqrt(X**2+Y**2))**2<=r**2 \
            and X>xcut and Y>ycut:
            Sro=Sro+ro;  Sro2=Sro2+ro*ro;
            ax.scatter(X,Y,Z,'.',color='blue',s=1)
```



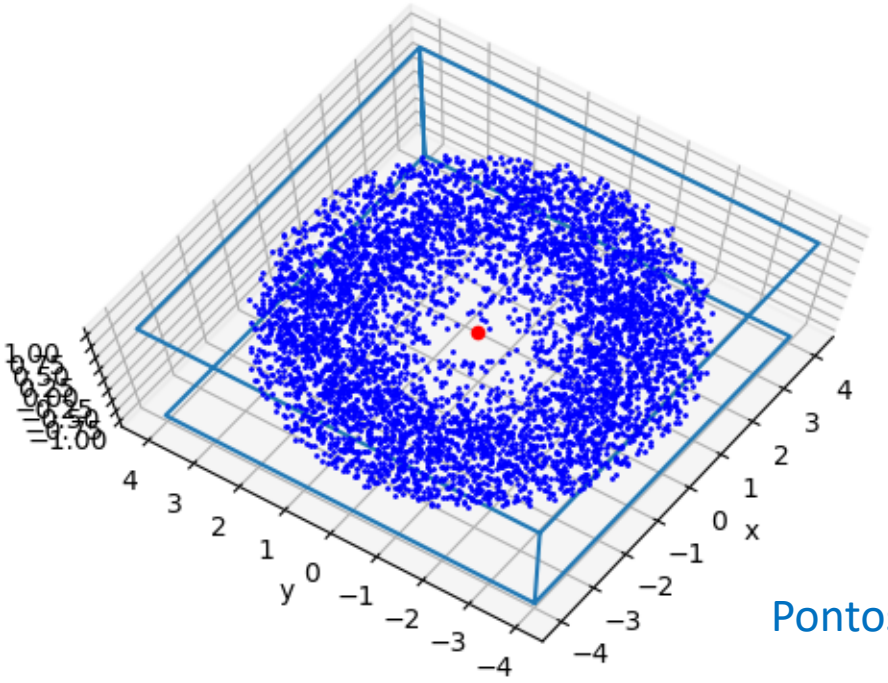
Centro de massa

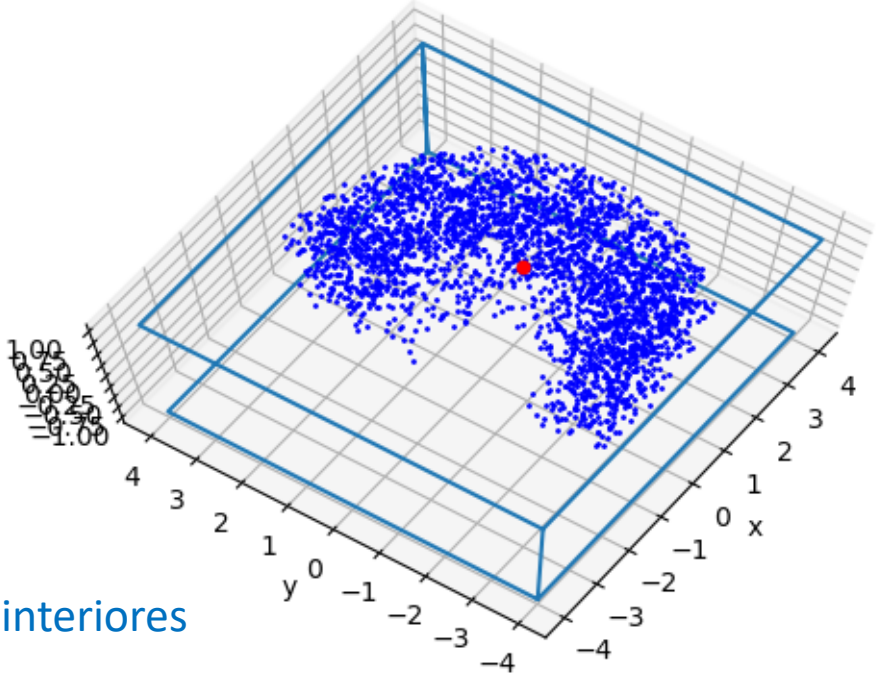
```
Sro=0;Sro2=0;Sx=0;Sx2=0;Sy=0;Sy2=0;Sz=0;Sz2=0;
for k in range(n):
    X=xmin+xlen*np.random.rand();
    Y=ymin+ylen*np.random.rand();
    Z=zmin+zlen*np.random.rand();
    if Z**2+(c-np.sqrt(X**2+Y**2))**2<=r**2 \
            and X>xcut and Y>ycut:
        Sro=Sro+ro;
        Sro2=Sro2+ro*ro;
        Sx=Sx+ro*X;
        Sx2=Sx2+(ro*X)**2;
        Sy=Sy+ro*Y;
        Sy2=Sy2+(ro*Y)**2;
        Sz=Sz+ro*Z;
        Sz2=Sz2+(ro*Z)**2;
```

Toro vs secção de toro (N=10000)

```
if Z**2+
c-np.sqrt(X**2+Y**2))**2<=r**2:
```

```
if Z**2+(c-
np.sqrt(X**2+Y**2))**2<=r**2
and X>xcut and Y>ycut:
```

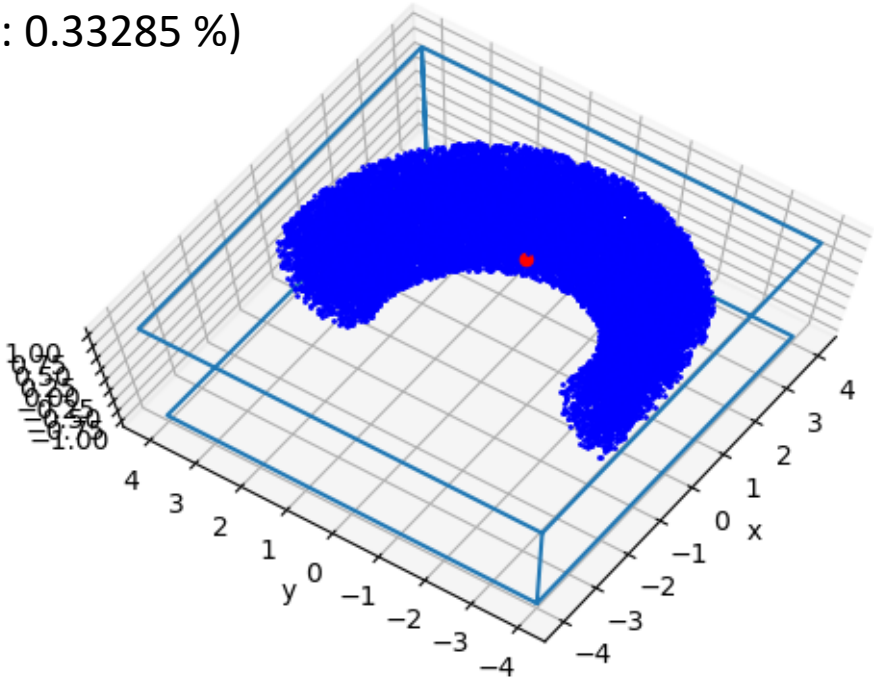




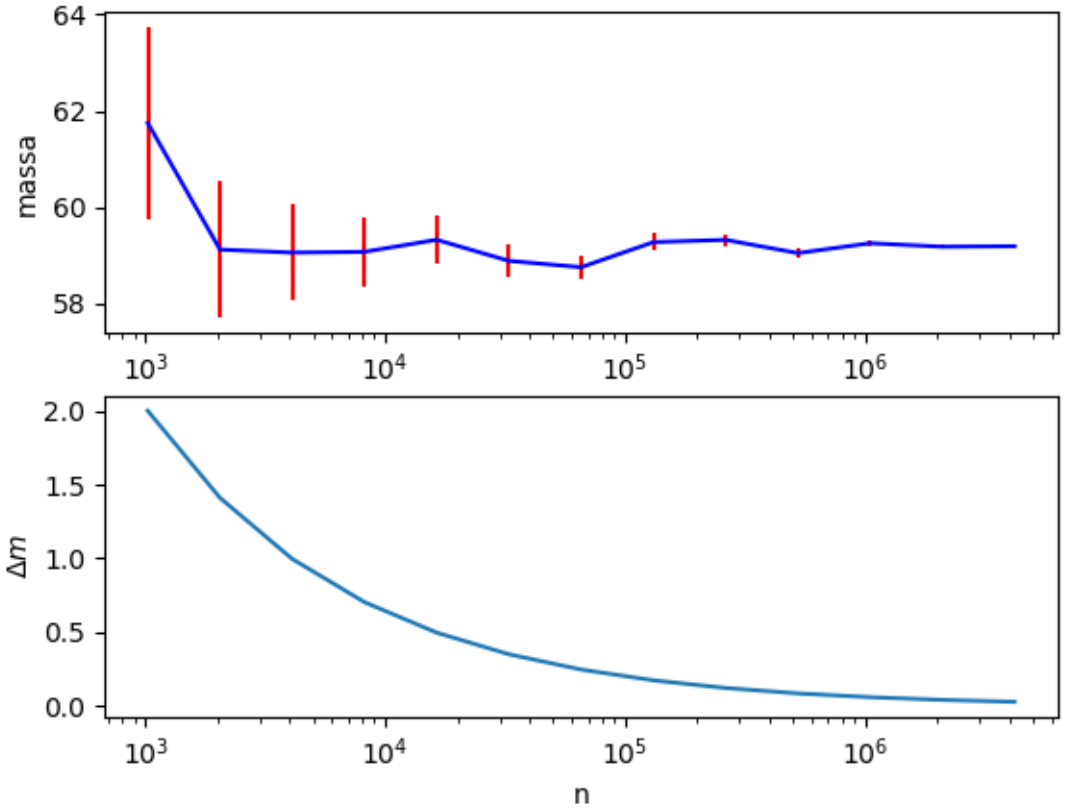
Pontos interiores

n=10000

n= 100000
massa= 3.5145216e+04 kg +- 2.0403723e+02 kg (Err: 0.58055 %)
xCM= 1.8694496e+00 +- 1.3220431e-02 (Err: 0.33051 %)
yCM= 3.8694747e-01 +- 1.4133831e-02 (Err: 0.35335 %)
zCM= 2.9300444e-04 +- 3.3285162e-03 (Err: 0.33285 %)

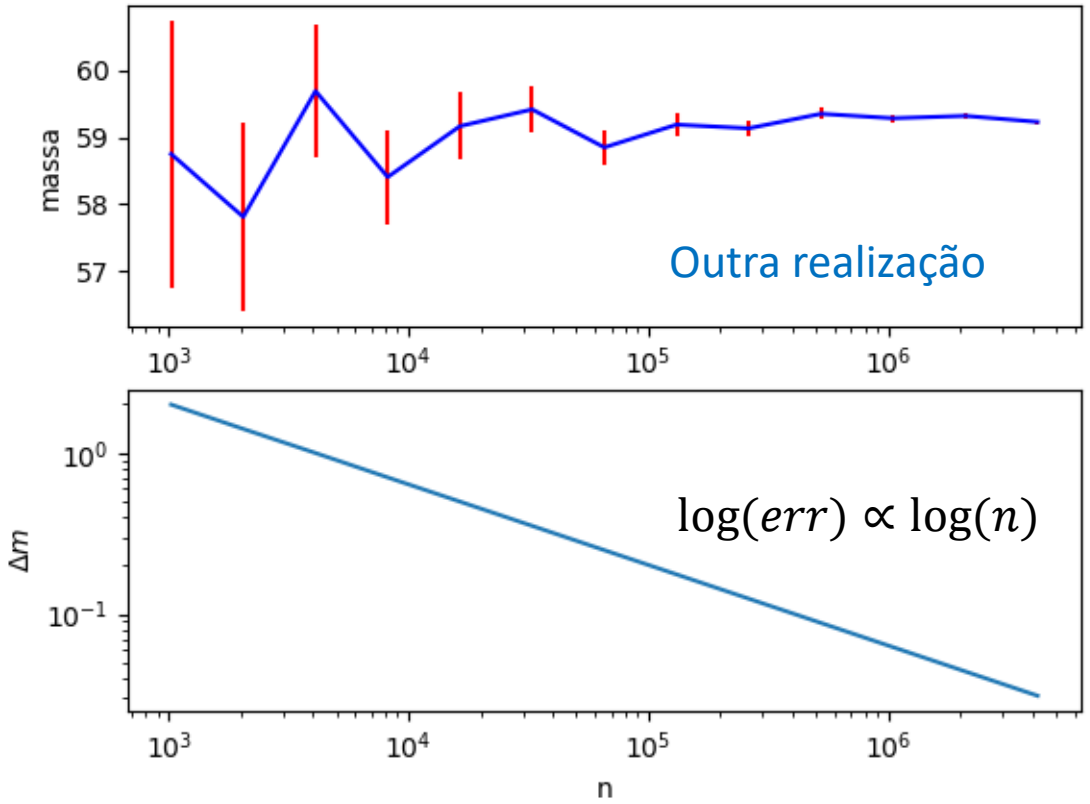


Evolução do erro gráfico semilogx




```
mass=[]
errmass=[]
nns=np.arange(10,23)
ns=2**nns
for n in ns:
    …#código anterior
    mass.append(massa)
    errmass.append(delmas)
plt.figure()
ax=plt.subplot(2,1,1)
ax.set_xscale('log')
plt.errorbar(ns,mass,yerr=errmass,ecolor='red',color='blue')
plt.ylabel('massa')
ax=plt.subplot(2,1,2)
ax.set_xscale('log')
plt.plot(ns,errmass)
plt.ylabel(r'$\Delta m$')
plt.xlabel('n')
```

Evolução do erro gráfico loglog




```
mass=[]
errmass=[]
nns=np.arange(10,23)
ns=2**nns
for n in ns:
    …#código anterior
    mass.append(massa)
    errmass.append(delmas)
plt.figure()
ax=plt.subplot(2,1,1)
ax.set_xscale('log')
plt.errorbar(ns,mass,yerr=errmass,ecolor='red',color='blue')
plt.ylabel('massa')
ax=plt.subplot(2,1,2)
ax.set_xscale('log');plt.set_yscale ('log');
plt.plot(ns,errmass)
plt.ylabel(r'$\Delta m$')
plt.xlabel('n')
```