# Aula 8

Integração numérica

$$\int_a^b f(x)dx$$

$$\iint_S f(x,y)\ dx\ dy$$

$$\iiint_V f(x,y,z)\ dx\ dy\ dz$$

# Calcular: $\int_a^b f(x)dx$

Massa de placa semi-circular homogénea (densidade $\rho \equiv kgm^{-2}$).

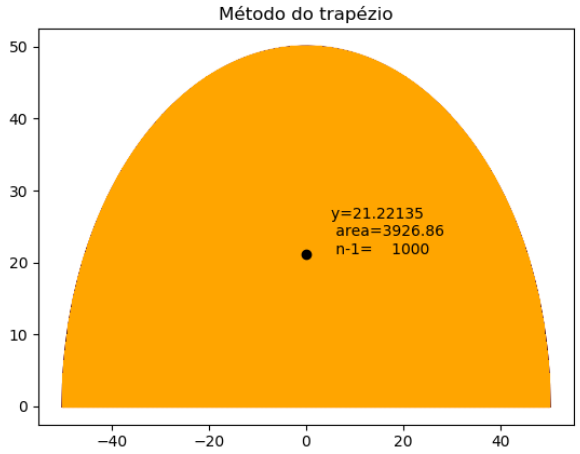
$y = f(x) = \sqrt{R^2 - x^2}$

$$m = \iint_S \rho \, dS = \int_{-R}^{R} \rho y dx = \frac{\rho \pi R^2}{2}$$

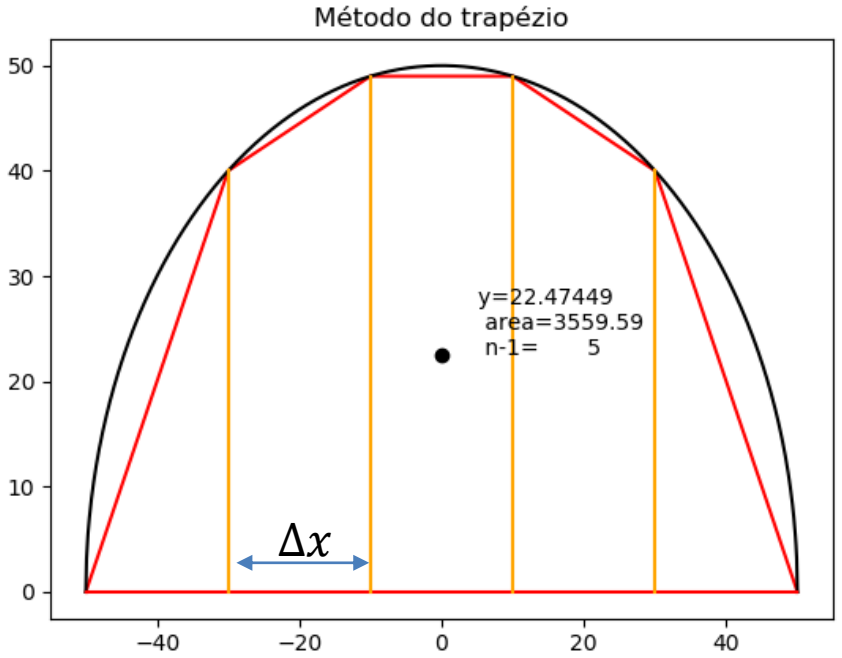
($m = área$, quando $\rho = 1$)

Centro de massa:

$$\vec{r}_{CM} = \frac{1}{m} \iint_S \vec{r} \, \rho dS$$



# Método do trapézio




(1) Divide-se o intervalo de integração em N sub-intervalos

(2) Aproxima-se a função integranda pela linha quebrada

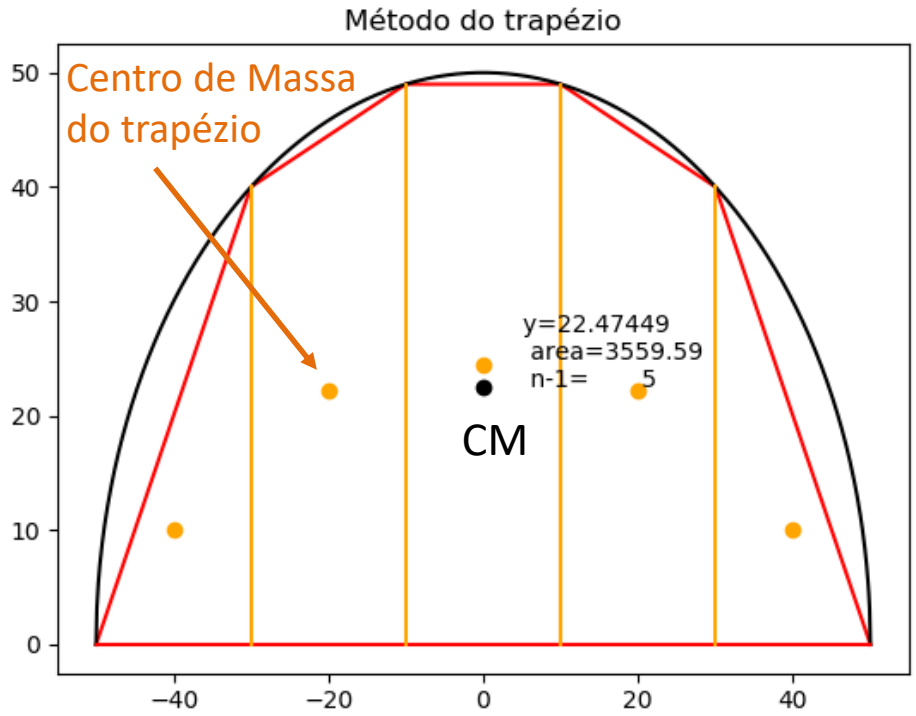
(3) Cada elemento de integração é um trapézio (neste caso, com 2 triângulos em cada limite)

(4) No $\lim_{N\to\infty}$ a linha quebrada converge para a função.

$$\int_{x_0}^{x_N} f(x)dx \approx S_N = \Delta x \sum_{k=1}^{N} \frac{f_{k-1}+f_k}{2} = \Delta x \left[\frac{f_0}{2} + f_1 + \cdots + f_{N-1} + \frac{f_N}{2}\right]$$

# Cálculo do centro de massa pelo método do trapézio

```
import numpy as np
import matplotlib.pyplot as plt
def placa(x,R):
    f=np.sqrt(R**2-x**2)
    return f
ro=1.;R=50.;kp=0; n=6
x=np.linspace(-R,R,n+1);dx=x[1]-x[0]
f=placa(x,R)
massa=ro*(f[0]+f[n])/2
xcm=ro*(f[0]*x[0]+f[n]*x[n])/2
ycm=ro*(f[0]*f[0]/2+f[n]*f[n]/2)/2
for k in range(1,n):
    massa=massa+ro*f[k]
    xcm=xcm+ro*f[k]*x[k]
    ycm=ycm+ro*f[k]*f[k]/2
massa=massa*dx
xcm=xcm*dx/massa
ycm=ycm*dx/massa
plt.scatter(xcm,ycm,color='black',zorder=10)
```




$$\Delta x \left[ \frac{f_0}{2} + f_1 + \cdots + f_{N-1} + \frac{f_N}{2} \right]$$

n é o número de intervalos
(n+1) pontos

# Convergência trapézio

```
from sympy import integrate,Symbol,sqrt
x=Symbol('x');f=Symbol('f');f=sqrt(50**2-x**2)
a=integrate(f,(x,-50,50))
yCM=integrate(f**2/2,(x,-50,50))/a
print('Analítico área=',a,'=%23.16f yCM=%23.16f' %(a,yCM))

n=     6 massa=  3559.592 kg xcm=     0.000 m ycm= 22.47448714 m
n=    11 massa=  3796.311 kg xcm=     0.000 m ycm= 21.73162294 m
n=   101 massa=  3922.836 kg xcm=     0.000 m ycm= 21.24101229 m
n=  1001 massa=  3926.859 kg xcm=    -0.000 m ycm= 21.22134832 m
n= 10001 massa=  3926.987 kg xcm=    -0.000 m ycm= 21.22068133 m
n=100001 massa=  3926.991 kg xcm=     0.000 m ycm= 21.22065979 m

Analítico área= 1250*pi =  3926.990816987241 4 yCM=
21.220659078919379 0
```

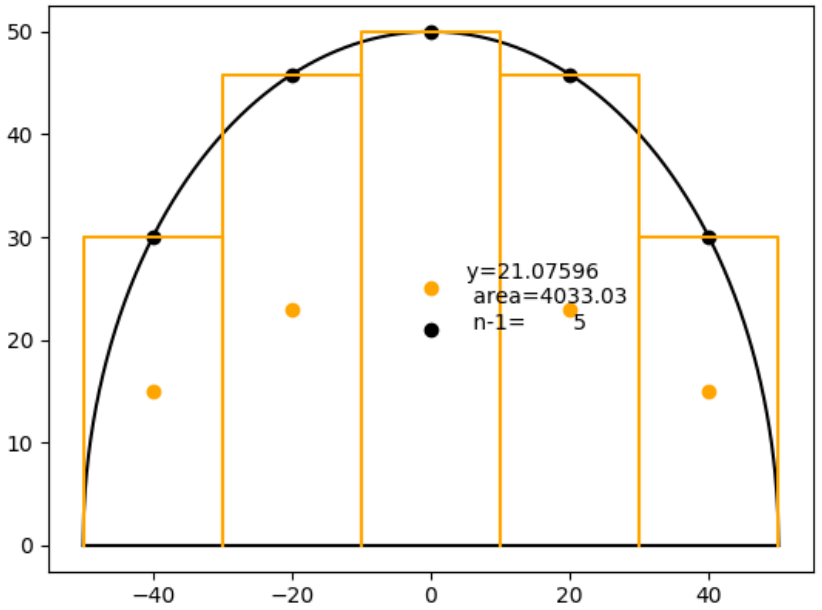
# TRAPÉZIO vers.2

```
n=100
mPM=intTRAP(placa,-R,R,n)
xPM=intTRAP(placax,-R,R,n)/mTRAP
yPM=intTRAP(placay,-R,R,n)/mTRAP
```

```
import numpy as np
def placa(x,R=50):
    f=np.sqrt(R**2-x**2)
    return f
def placax(x,R=50):
    f=placa(x,R)*x
    return f
def placay(x,R=50):
    f=placa(x,R)**2/2
    return f
```

```
def intTRAP(fun,a,b,n):
    dx=(b-a)/n
    x=np.linspace(a,b,n+1)
    f=fun(x)
    integral=0.5*(f[0]+f[n])
    for k in range(1,n):
        integral=integral+f[k]
    integral=integral*dx
    return integral
```

# Método do ponto médio




(1) Divide-se o intervalo de integração em N sub-intervalos

(2) Toma-se o valor da função no ponto médio

(3) Cada elemento de integração é um retângulo

(4) No $\lim_{N\to\infty}$ a escada converge para a função.

$$\int_{x_0}^{x_N} f(x)dx \approx S_N = h\sum_{k=1}^{N} f_{k-\frac{1}{2}} = h\left[f_{1/2} + f_{3/2} + \cdots + f_{N-1/2}\right]$$

# PONTO MÉDIO

```
n=100
mPM=intPM(placa,-R,R,n)
xPM=intPM(placax,-R,R,n)/mPM
yPM=intPM(placay,-R,R,n)/mPM
```

```
import numpy as np
def placa(x,R=50):
    f=np.sqrt(R**2-x**2)
    return f
def placax(x,R=50):
    f=placa(x,R)*x
    return f
def placay(x,R=50):
    f=placa(x,R)**2/2
    return f
```

```
def intPM(fun,a,b,n):
    dx=(b-a)/n
    x=np.linspace(a+dx/2,b-dx/2,n)
    f=fun(x)
    integral=0.
    for k in range(n):
        integral=integral+f[k]
    integral=integral*dx
    return integral
```


```
def intPM(fun,a,b,n):
    dx=(b-a)/n
    x=np.linspace(a+dx/2,b-dx/2,n)
    integral=dx*np.sum(fun(x))
    return integral
```

# Vantagens trapézio vs ponto médio

Os dois métodos têm desempenho comparável. Por vezes a função não pode ser calculada nos pontos limite, apesar de ser integrável: nesse caso usa-se o ponto médio.

O método do ponto médio é extensível a problemas multidimensionais.

# Método de Simpson

No método de Simpson, a função é reconstruída entre cada três pontos por uma parábola:

$$\int_{x_0}^{x_N} f(x)dx \approx S_N$$

$$= h\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \cdots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right]$$

# SIMPSON

```
n=100
mSI=intSIMP(placa,-R,R,n)
xSI=intSIMP(placax,-R,R,n)/mSI
ySI=intSImp(placay,-R,R,n)/mSI
```

```
import numpy as np
def placa(x,R=50):
    f=np.sqrt(R**2-x**2)
    return f
def placax(x,R=50):
    f=placa(x,R)*x
    return f
def placay(x,R=50):
    f=placa(x,R)**2/2
    return f
```

```
def intSIMP(fun,a,b,n):
    dx=(b-a)/n
    x=np.linspace(a,b,n+1)
    coef=np.zeros(x.shape,dtype=float)
    coef[0]=1./3
    coef[1:n:2]=4./3
    coef[2:n:2]=2./3
    coef[n]=1./3
    integral=dx*fun(x).dot(coef)
    return integral
```

Para outra função mudar

$$\Delta x \left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \cdots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right]$$

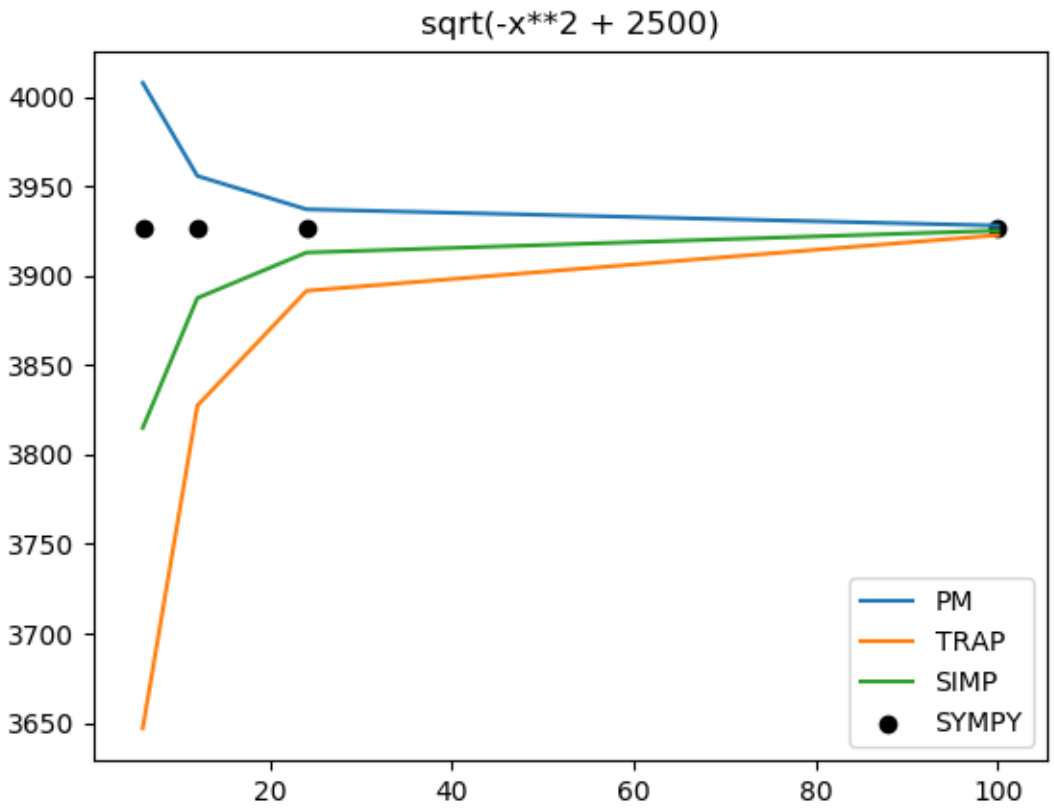
$$= \Delta x\, \vec{c}.\vec{f}$$

$$\vec{c} = \left[\frac{1}{3}, \frac{4}{3}, \frac{2}{3}, \frac{4}{4}, \frac{2}{3}, \cdots, \frac{2}{3}, \frac{4}{3}, \frac{1}{3}\right]$$

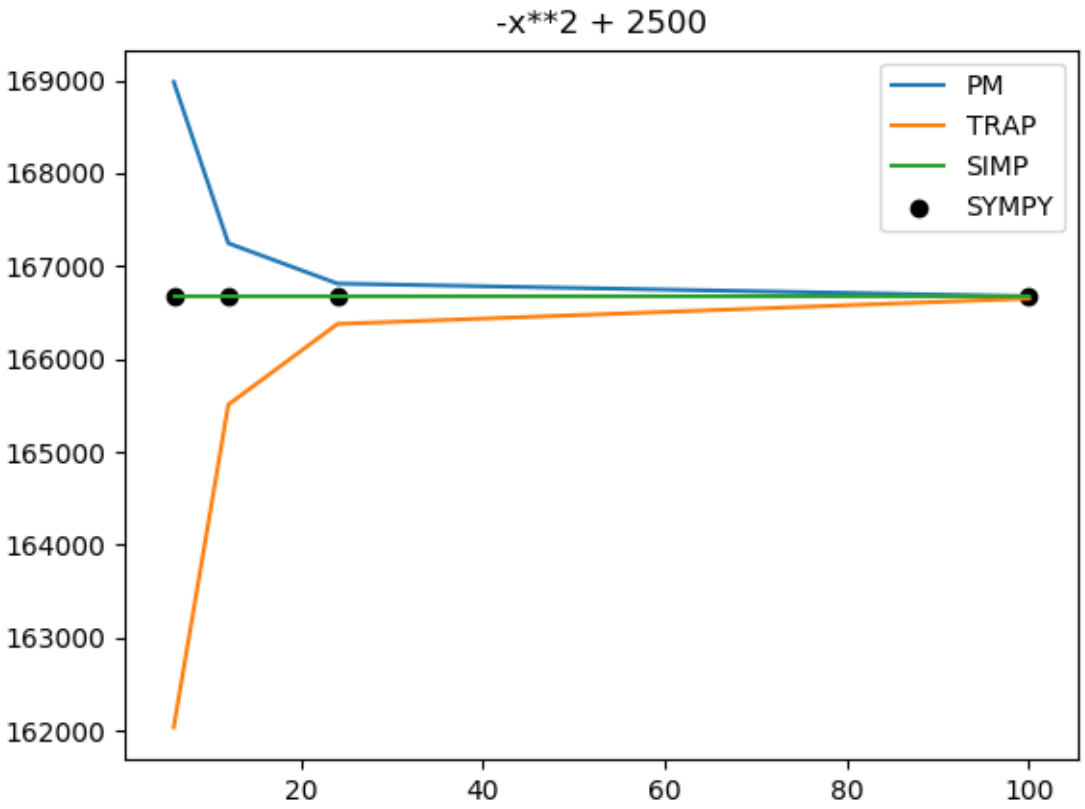
# Testes de desempenho

```
kp=-1;R=50
points=np.array([6,12,24,100]);area=np.zeros((len(points),3),dtype=float)
for n in points:
    kp=kp+1
    mPM=intPM(placa,-R,R,n)
    xPM=intPM(placax,-R,R,n)/mPM; yPM=intPM(placay,-R,R,n)/mPM
    mTRAP=intTRAP(placa,-R,R,n)
    xTRAP=intTRAP(placax,-R,R,n)/mTRAP; yTRAP=intTRAP(placay,-R,R,n)/mTRAP
    mSI=intSIMP(placa,-R,R,n)
    xSI=intSIMP(placax,-R,R,n)/mSI; ySI=intSIMP(placay,-R,R,n)/mSI
    area[kp,0]=mPM;area[kp,1]=mTRAP;area[kp,2]=mSI
from sympy import integrate,Symbol,sqrt
x=Symbol('x');f=Symbol('f');f=sqrt(50**2-x**2)
aSYM=integrate(f,(x,-R,R)) #solução exata
ySYM=integrate(f**2/2,(x,-R,R))/aSYM
plt.plot(points,area[:,0],label='PM')
plt.plot(points,area[:,1],label='TRAP')
plt.plot(points,area[:,2],label='SIMP')
plt.scatter(points,aSYM*np.ones(points.shape),label='SYMPY',color='black')
plt.title(f);plt.legend()
```

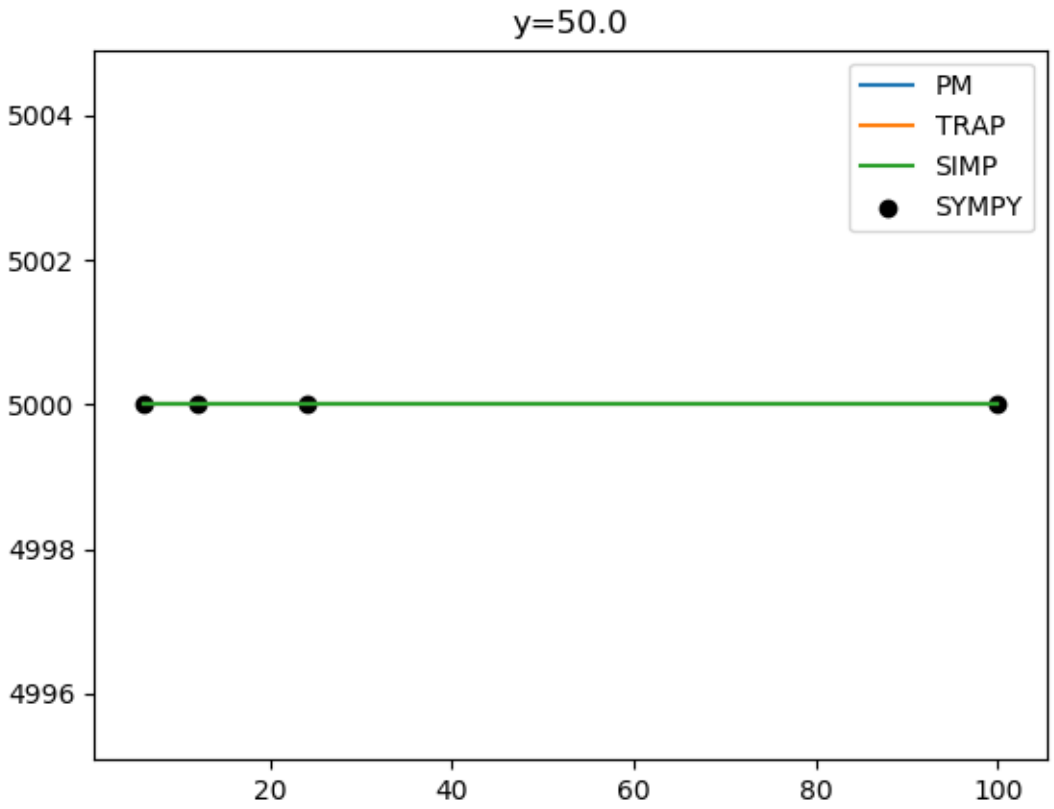
# semicírculo



# Parábola (SIMPSON dá resultado exato!)



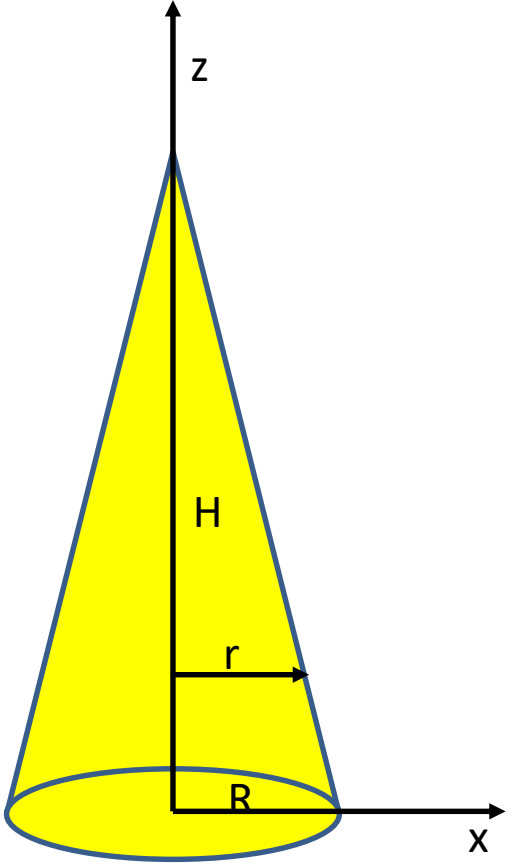
# Retângulo (todos exatos)



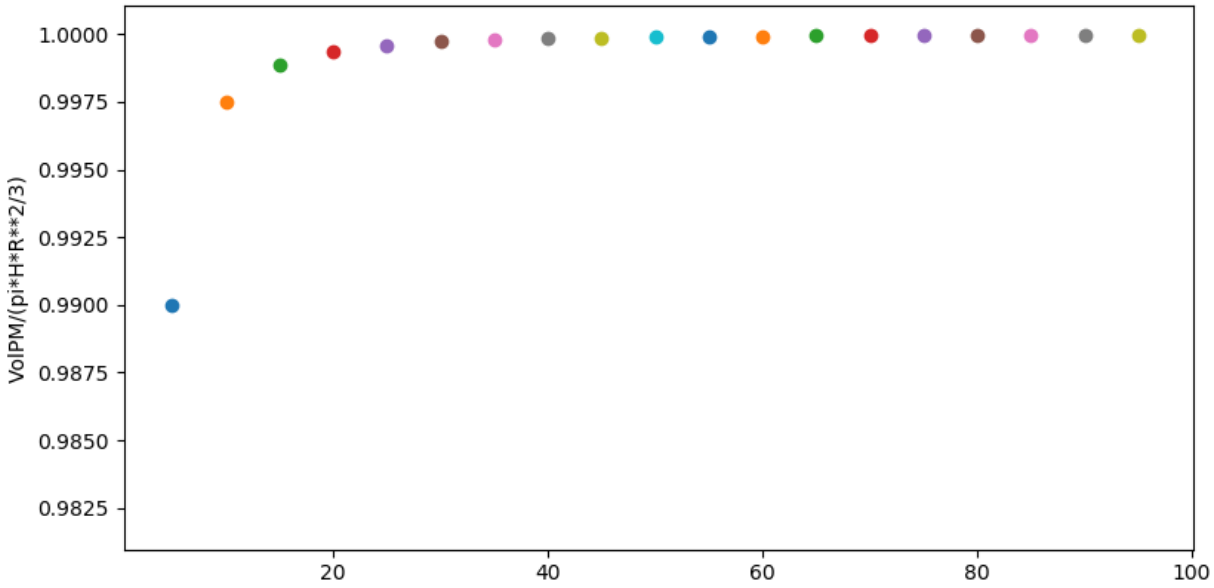
# Volume de um cone (integral 1D)

```
import numpy as np;from sympy import Symbol,integrate,pi
def circulo(z,R=2,H=10):
    circ=np.pi*(R-z/H*R)**2
    return circ
def intPMed(fun,a,b,n):
    dx=(b-a)/n
    x=np.linspace(a+dx/2,b-dx/2,n)
    integral=dx*np.sum(fun(x))
    return integral
H=10;R=2
z=Symbol('z');r=Symbol('r');r=(R-z/H*R)
Vol=integrate(pi*r**2,(z,0,H))
ns=np.arange(5,100,5)
for n in ns:
    VolPM=intPMed(circulo,0,H,n)
    print('VolPM=%25.16e VolANA=%25.16e' % (VolPM,Vol))
    plt.scatter(n,VolPM/Vol)
plt.ylabel('VolPM/('+str(Vol)+')')
```

$$r = R - \frac{z}{H}R$$



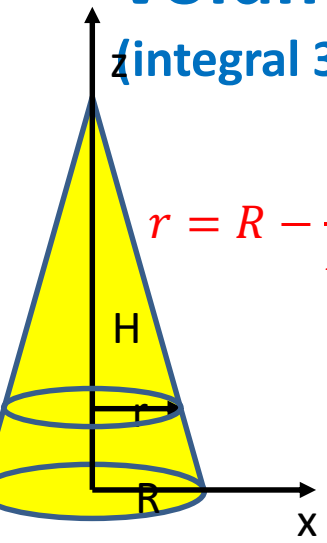
# Volume de um cone (integral 1D)



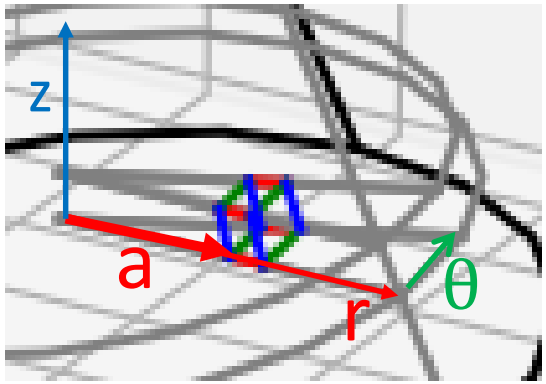
# Volume de um cone

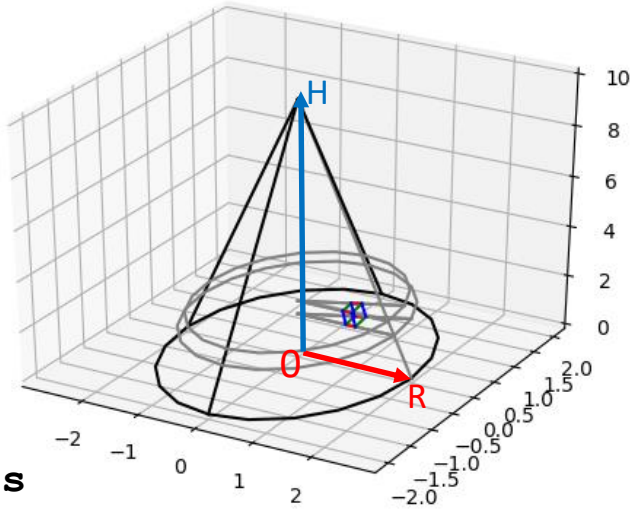
## (integral 3D, coord polares)

$$V = \int_{z=0}^{H} \left[ \int_{a=0}^{r=R-\frac{zR}{H}} \left( \int_{\theta=0}^{2\pi} a \, d\theta \right) da \right] dz$$




$$r = R - \frac{z}{H} R$$




```
from sympy import Symbol,integrate,pi,sin,cos
H=Symbol('H');z=Symbol('z');
r=Symbol('r');R=Symbol('R');
theta=Symbol('theta'); a=Symbol('a')
r=(R-z/H*R)
Vol3=\
integrate(integrate(integrate(a,(theta,0,2*pi)),(a,0,r)),(z,0,H))
Print(Vol3)
>>pi*H*R**2/3
```

# Comentário

$$V = \int_{z=0}^{H} \left[ \int_{a=0}^{r=R-\frac{zR}{H}} \left( \int_{\theta=0}^{2\pi} a \, d\theta \right) da \right] dz$$

Este cálculo podia ser feito numericamente com o método do ponto médio, mas era trabalhoso: seria necessário distribuir os “pontos médios” de forma homogénea em 3 dimensões (polares).

# Graficos 3D

Trajetória balística com $g = const$, sem atrito, sem Coriolis

$$\vec{a} = -g\vec{k}$$

$$\vec{v} = u\vec{\imath} + v\vec{\jmath} + w(t)\vec{k}$$

Admitindo que o projétil parte da posição $\vec{r} = (0,0,0)$, o tempo decorrido aré voltar ao solo $(z = 0)$ será $t_{Max} = 2 * w_0/g$

Equações da trajetória:

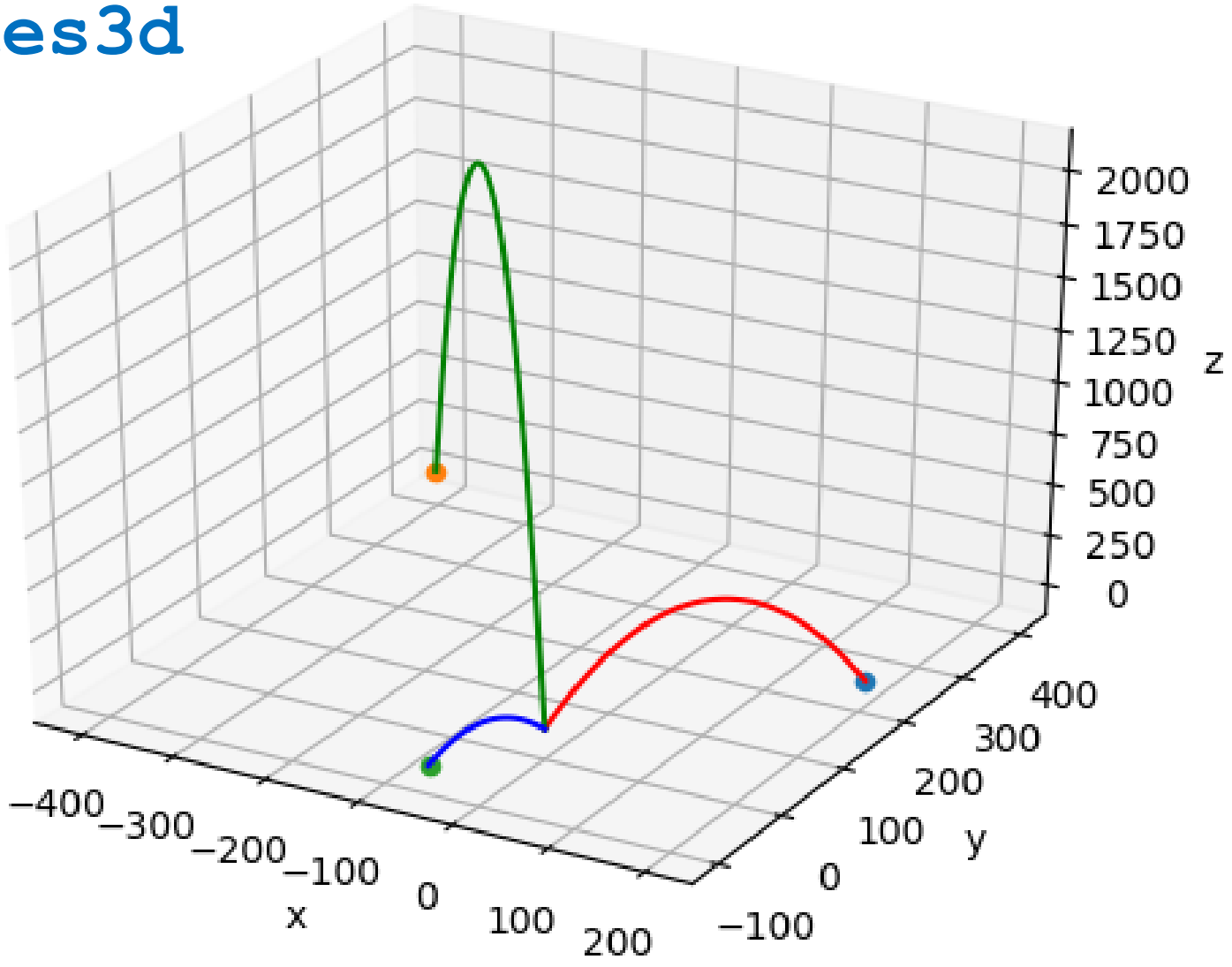
$$\begin{cases} x = x_0 + ut \\ y = y_0 + vt \\ z = z_0 + w_0 t - \frac{1}{2} g t^2 \end{cases}$$

# Trajetória balística analítica

$$\begin{cases} x = x_0 + ut \\ y = y_0 + vt \\ z = z_0 + w_0 t - \frac{1}{2} g t^2 \end{cases}$$

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
u=[10,-10,-5];v=[10,10,-10];w0=[100,200,50]
x0=0;y0=0;z0=0;t0=0;g=9.8;cores=['green','red','blue']
plt.close('all');fig=plt.figure(1) #cria figura 1
ax=fig.add_subplot(111,projection='3d') #cria figura 3D
for k in range(len(cores)):
    tmax=2*w0[k]/g
    t=np.linspace(t0,tmax,101)
    x=x0+u[k]*t; y=y0+v[k]*t
    z=z0+w0[k]*t-1/2*g*t**2
    ax.plot(xs=x,ys=y,zs=z,color=cores[k])
    ax.set_xlabel('x')
    ax.set_ylabel('y');ax.set_zlabel('z')
    ax.scatter(x[-1],y[-1],z[-1])
```

# Axes3d


2000
1750
1500
1250
1000
750
500
250
0
z
400
300
200
100
0
−100
y
−400
−300
−200
−100
0
100
200
x