Aula 23

Campo elétrico de uma distribuição estacionária de cargas: equação de Poisson

Equação de Poisson

o potencial gerado por uma distribuição contínua de carga elétrica numa placa satisfaz à equação:

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = -\frac{\rho}{\varepsilon_0}$$

em que V é o potencial elétrico, $\rho(x, y)$ a densidade volúmica de carga e ε_0 a permitividade elétrica do meio.

Em três dimensões, seria:

$$\nabla^2 V = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = -\frac{\rho}{\varepsilon_0}$$

Diferenças centradas

A diferença finita centrada, constitui uma aproximação de segunda ordem, i.e.:

$$\frac{\partial^2 \phi}{\partial x^2} \approx \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{\Delta x^2} + E\left(\Delta x^2\right)$$

Resultando da soma das duas séries de Taylor:

$$\phi(x + \Delta x) = \phi(x) + \frac{d\phi}{dx}\Delta x + \frac{1}{2}\frac{d^2\phi}{dx^2}\Delta x^2 + \frac{1}{3!}\frac{d^3\phi}{dx^3}\Delta x^3 + \cdots$$

$$\phi(x - \Delta x) = \phi(x) - \frac{d\phi}{dx}\Delta x + \frac{1}{2}\frac{d^2\phi}{dx^2}\Delta x^2 - \frac{1}{3!}\frac{d^3\phi}{dx^3}\Delta x^3 + \cdots$$

$$\phi(x + \Delta x) + \phi(x - \Delta x) = 2\phi(x) + \frac{d^2\phi}{dx^2}\Delta x^2 + \frac{2}{4!}\frac{d^4\phi}{dx^4}\Delta x^4 + \cdots$$

etc.

Equação de Poisson discreta, em diferenças centradas

Se for $\Delta x = \Delta y = \Delta$, fica

$$\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i.j+1} - 4\phi_{i,j} = \Delta^2 f_{i,j}$$

$$\{i = 1, \dots, M-2; j = 1, \dots, N-2\}$$

onde se notou que as diferenças centradas só se podem calcular nos pontos interiors do domínio. Na fronteira os valores $(i = 0, M-1; j = 0, N-1)$, têm que ser impostos.

Método da relaxação

A solução satisfaz:

$$\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i.j+1} - 4\phi_{i,j} = \Delta^2 f_{i,j}$$

Começamos por aribtrar uma distribuição para ϕ, por exemplo $\phi = 0$, e vamos melhorar essa estimativa, de forma iterativa:

Dada uma estimativa do campo ϕ, na iteração n existe um erro (resíduo R):

$$\phi^n_{i-1,j} + \phi^n_{i+1,j} + \phi^n_{i,j-1} + \phi^n_{i,j+1} - 4\phi^n_{i,j} - \Delta^2 f_{i,j} = R_{i,j}$$

Se se corrigir:

$$\phi^{n+1}_{i,j} = \phi^n_{i,j} + \frac{R_{i,j}}{4}$$

O erro será anulado (mas só nesse ponto!)

Sobre-relaxação simultânea

Só se mantém um array de ϕ. Faz-se:

$$\phi_{i,j} = \phi_{i,j} + \beta \frac{R_{i,j}}{4}$$

i.e., à medida que se altera um ponto de grelha o novo valor já é utilizado no cálculo do resíduo dos pontos adjacentes.

$$1 \leq \beta < 2$$

É o parâmetro de sobre-relaxação. Pode mostrar-se que o método converge mais rapidamente para:

$$\beta_{opt} = 2 - \pi\sqrt{2}\left(\frac{1}{M^2} + \frac{1}{N^2}\right)^{1/2}$$

Python: resolve $\nabla^2 V = f$

```
def poisson(f,V0,X,Y,maxiter,maxres,beta=0):
    [M,N]=f.shape; #determina a dimensão das
matrizes
    dx=X[2,1]-X[1,1];dy=Y[1,2]-Y[1,1]
    if dx!=dy: #Admite-se espaçamento regular
        print('Error in dx,dy')
        return  V0,0,1e30,beta0
    delta=dx
    if (beta<1) or (beta>2): #parametro de
sobrerrelaxação
        beta=2-np.pi*np.sqrt(2.)*\
             np.sqrt(1./M**2+1./N**2)
    V=V0 #inicializa a matriz soluçáo
    iter=0
    resid=2*maxres #garante a primeira iteração
```

python

```
        while resid>maxres and iter<maxiter: #iterações
        iter=iter+1
        resid=0; vmax=0
        for i in range(1,M-1): #vai de 1 a M-2
            for j in range(1,N-1):
                R=V[i,j-1]+V[i,j+1]+V[i-1,j]+\
                  V[i+1,j]-4*V[i,j]-delta**2*f[i,j]
                V[i,j]=V[i,j]+beta*R/4;
                resid=max(resid,abs(R))
#condições fronteira: entram aqui!
#se não se fizer nada V na fronteira fica sempre o mesmo
#o que é uma condição fronteira possível
        vmax=np.max(np.abs(V))
        resid=resid/vmax #residuo relativo
    return V,iter,resid,beta
```

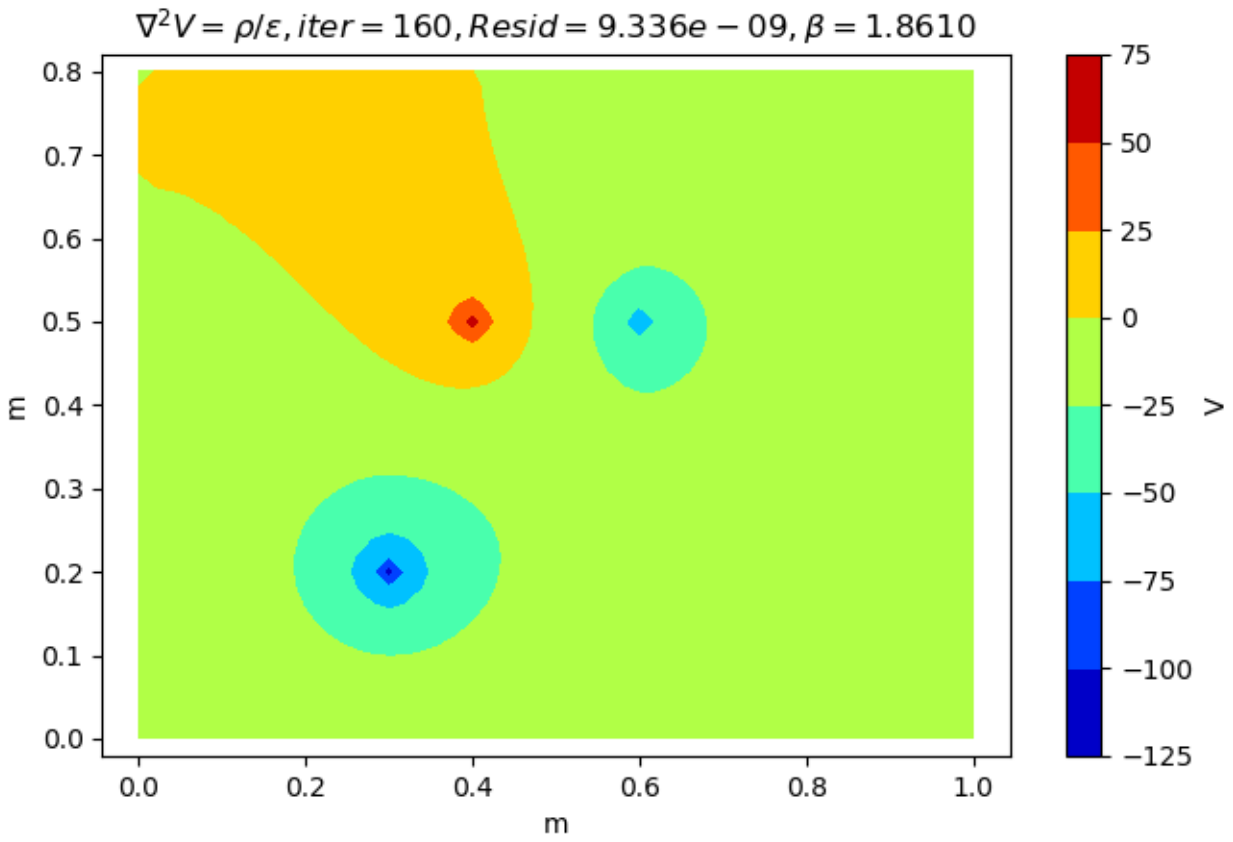
main

```
import numpy as np
import matplotlib.pyplot as plt
plt.close('all') #fecha figuras anteriores
Lx=1.;Ly=1.;M=51;N=41;delta=Lx/(M-1) #Malha de cálculo
X=np.zeros((M,N));Y=np.zeros((M,N));ro=np.zeros((M,N))
for i in range(M):
    for j in range(N):
        X[i,j]=i*delta;Y[i,j]=j*delta
ncargas=4 # define as cargas pontuais e sua localização
xis=np.array([0.4,0.6,0.3,0.5]);yps=np.array([0.5,0.5,0.2,0.8])
q=np.array([1e-9,-1e-9,-1.5e-9,-2e-9]);
for carga in range(ncargas):
    i=int(xis[carga]/delta) #i,j devem ser inteiros
    j=int(yps[carga]/delta)
    ro[i,j]=q[carga]/delta**2
#define a função forçadora na equação de Poisson (segundo membro)
eps0=8.8544e-12; f=-ro/eps0
maxiter=5000 # número máximo de iterações
maxres=1.e-8 # erro relative máximo
V0=0*np.ones((M,N)) #potencial inicial
beta0=0. #beta é calculado em poisson
[V,niter,res,beta]=poisson(f,V0,X,Y,maxiter,maxres)
```

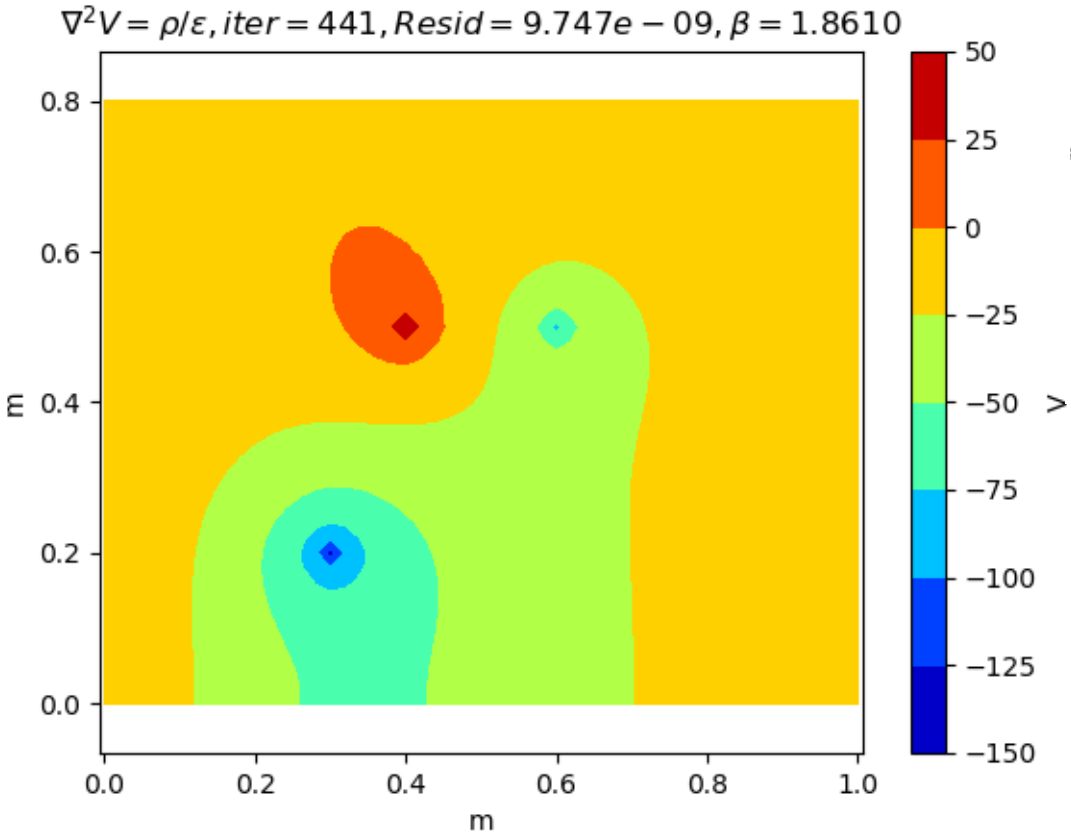
gráficos

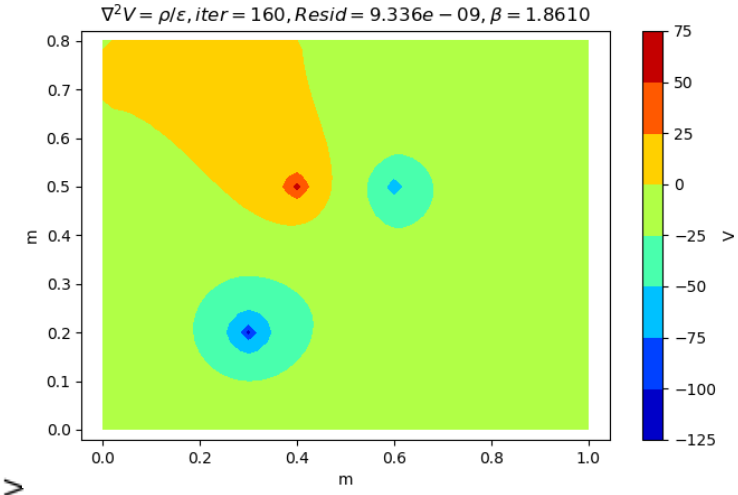
```
plt.figure(2)
map=plt.contourf(X,Y,V,cmap='jet') # gráfico de isolinhas
axes = plt.gca()
axes.set_xlim([0.,Lx])
axes.set_ylim([0.,Ly])
plt.colorbar(map,label='V')
plt.xlabel('m');plt.ylabel('m')
plt.axis('equal')
plt.show()
plt.title(r'$\nabla^{2} V=\rho/\epsilon, iter=%6i,
Resid=%10.3e, \beta=%8.4f  $' % (niter,res,beta))
plt.savefig('Poisson_final.png')
```

Solução com condição fronteira $V_{(0,L_x),(0,L_y)} = 0$

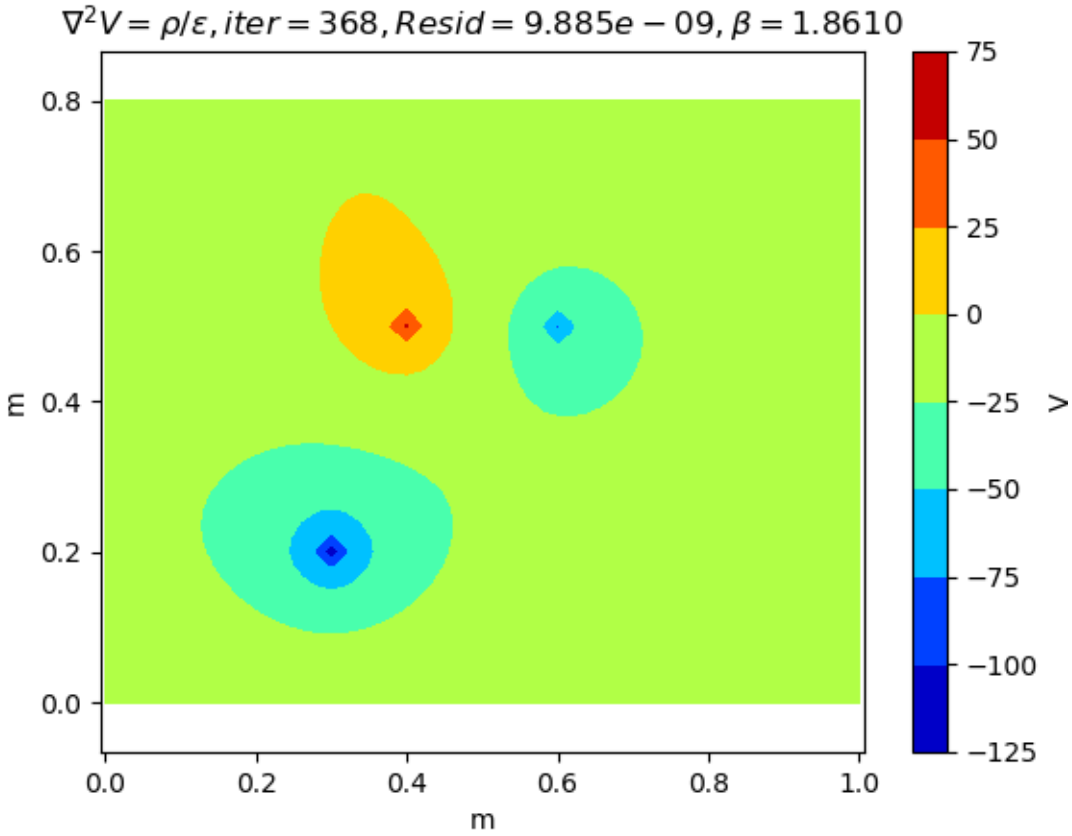


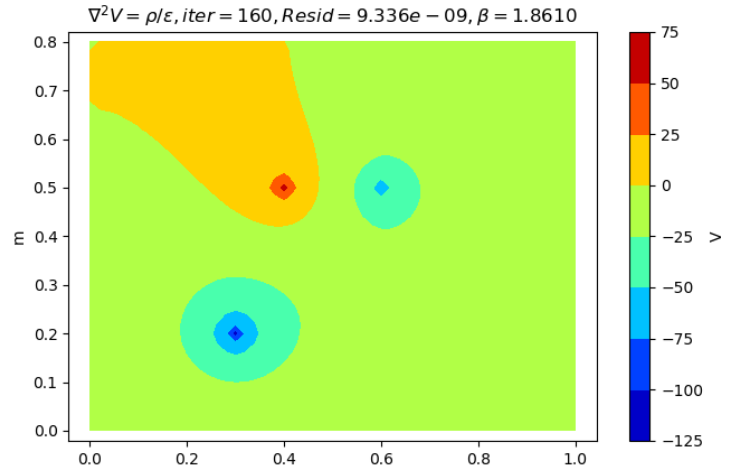
Com tudo =, mas $\left(\frac{\partial V}{\partial x}\right)_{y=o,L_y} = 0$

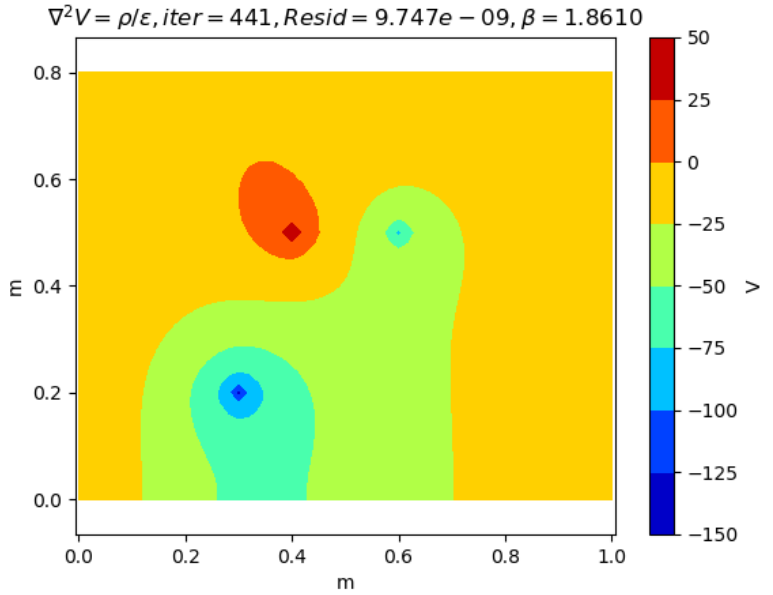




Com tudo =, mas $\left(\frac{\partial V}{\partial y}\right)_{x=o,L_x} = 0$

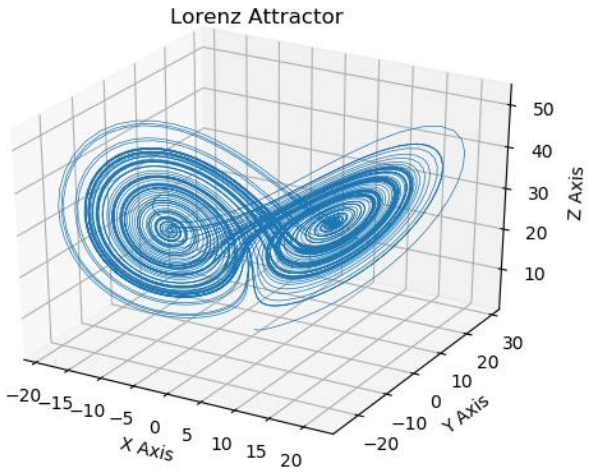






O atractor de Lorenz

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x) \\ \dfrac{dy}{dt} = x(\rho - z) - y, \begin{cases} \sigma = 10 \\ \rho = 28 \\ \beta = 2.667 \end{cases} \\ \dfrac{dz}{dt} = xy - \beta z \end{cases}$$



Variáveis:

x – intensidade da convecção

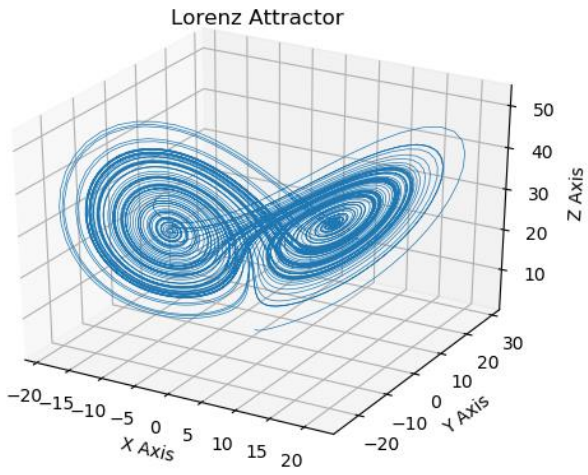
y – forçamento térmico (ΔT entre ascendente e descendente)

z – perturbação do gradiente vertical de temperatura

Parâmetros: σ, ρ, β (dependem da viscosidade, da difusividade térmica e da geometria)

$$\begin{cases} \dfrac{dx}{dt} = \sigma(y - x) \\ \dfrac{dy}{dt} = x(\rho - z) - y, \begin{cases} \sigma = 10 \\ \rho = 28 \\ \beta = 2.667 \end{cases} \\ \dfrac{dz}{dt} = xy - \beta z \end{cases}$$

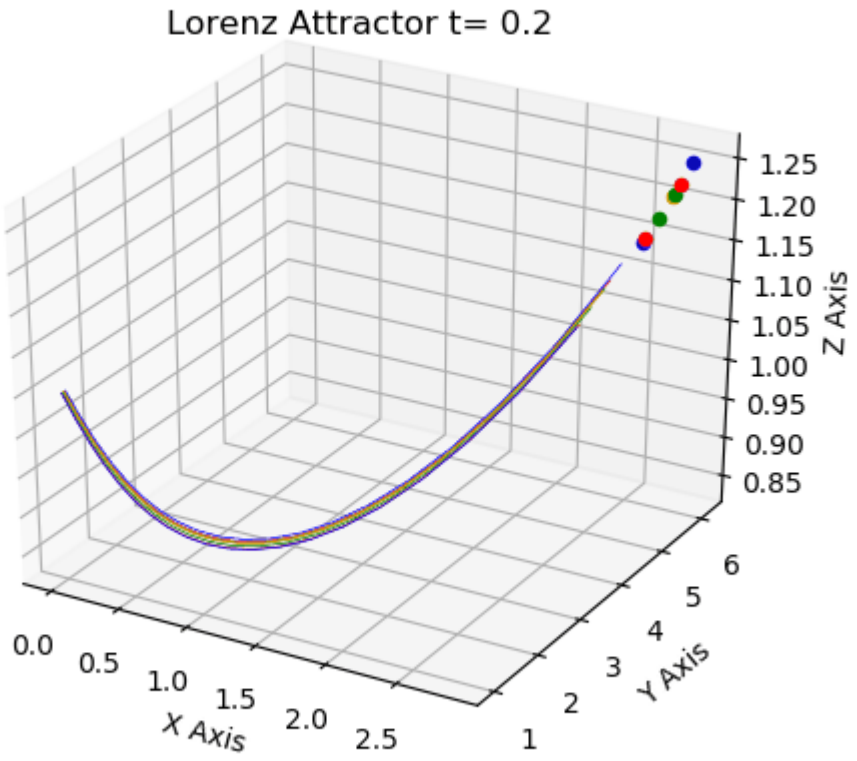
```
import numpy as np;import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
def lorenz(x, y, z, s=10, r=28, b=2.667):
    x_dot = s*(y - x)
    y_dot = r*x - y - x*z
    z_dot = x*y - b*z
    return x_dot, y_dot, z_dot
dt = 0.01; stepCnt = 10000
xs = np.empty((stepCnt + 1,))
ys = np.empty((stepCnt + 1,))
zs = np.empty((stepCnt + 1,))
xs[0], ys[0], zs[0] = (0., 1., 1.05)
for i in range(stepCnt):
    x_dot,y_dot,z_dot=lorenz(xs[i],ys[i],zs[i])
    xs[i+1] = xs[i] +x_dot * dt
    ys[i+1] = ys[i] +y_dot * dt
    zs[i+1] = zs[i] +z_dot * dt
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")
plt.show()
```



Seguindo 4 partículas que começam quase no mesmo ponto

Ao fim de pouco tempo, cada partícula segue trajetorias completamente diferentes:

CAOS



Posição inicial das partículas

```
dt=0.01;stepCnt=2000
num=4
xs = np.empty((stepCnt + 1,num))
ys = np.empty((stepCnt + 1,num))
zs = np.empty((stepCnt + 1,num))
cor=['red','green','blue','orange']
for k in range(num):
    xs[0,k],ys[0,k],zs[0,k]=\
            (0.,1.+0.1*np.random.rand(), 1.05)
```

Integração temporal

```
for k in range(num):

    for i in range(stepCnt):
        x_dot,y_dot,z_dot=\
           lorenz(xs[i,k], ys[i,k], zs[i,k])
        xs[i+1,k]=xs[i,k]+(x_dot * dt)
        ys[i+1,k]=ys[i,k]+(y_dot * dt)
        zs[i+1,k]=zs[i,k]+(z_dot * dt)
```

```
for i in range(20,stepCnt,20):
    fig = plt.figure(1)
    ax = fig.gca(projection='3d')
    plt.scatter(1+np.max(xs[:,0]),\
            1+np.max(ys[:,0]),np.max(zs[:,0])+5,alpha=0)

    for k in range(num):
        ax.plot(xs[0:i,k], ys[0:i,k], zs[0:i,k],\
                lw=0.5,color=cor[k])
        ax.scatter(xs[i,k],ys[i,k],zs[i,k],color=cor[k])
        ax.set_title("Lorenz Attractor t=%4.1f "%(i*dt))
    plt.pause(0.1)
    plt.show()
    if movie!='':
        frame=movie+str(i)+'.png'
        pngs.append(frame)
        plt.savefig(frame)
    plt.clf()
```

```
if len(pngs)>0:
    import imageio
    import os
    images=[]
    for frame in pngs:
        images.append(imageio.imread(frame))
        os.remove(frame)
    imageio.mimsave('lorenz.gif',\
        images,duration=0.1)
```