Nome	Nº	P	G

Exame 1

Identifique-se nas páginas 1 e 3. (NOTA A solução proposta não é a única possível)

1. As funções seguintes contêm (~3) erros fatais. Corrija-as. Admita que, em todos os casos se fez `import numpy as np`

(a) Cálculo de (x, y, z)

$$\begin{cases} x - 2z = 0 \\ x + y - 5z = 0 \\ 7x - 8y + 9 = 0 \end{cases}$$

```
M=[[1,0,-2],[1,2,-5],[7,-8,9]]
b=np.array([0.,0.,0.])
X=np.linalg.solve(M,b)
x=X[1];y=X[2];z=X[3]
```

(a) corrigido

```
M=np.array([[1,0,-2],[1,1,-5],[7,-
8,9]])
b=np.array([0.,0.,-9.])
X=np.linalg.solve(M,b)
x=X[0];y=X[1];z=X[2]
```

(b) Cálculo da média de uma matrix

```
def medMAT(M):
    m,n=M.shape
    for i in range(n):
        for j in range(m):
            med=med+M[i,j]
    return med
```

(b) corrigido

```
def medMAT(M):
    m,n=M.shape
    med=0.
    for i in range(m):
        for j in range(n):
            med=med+M[i,j]
    med=med/(m*n)
    return med
```

(c) Cálculo de

$$RMS = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(x_k - o_k)^2}$$

```
def RMS(x,o):
    sigma=0.
    for k in range(N)
        sigma=sigma+(x[k]-o[k])**2
    sigma=np.sqrt(sigma/N)
    return RMS
```

(c) corrigido

```
def RMS(x,o):
    sigma=0.
    N=len(x)
    for k in range(N):
        sigma=sigma+(x[k]-o[k])**2
    sigma=np.sqrt(sigma/N)
    return sigma
```

(d) Localização do elemento de maior valor absoluto numa matriz

```
def fAmax(M):
    mx=M[0,0]
    for i in range(lin):
        for j in range(col):
            if M[i,j]>mx:
                mx=M[i,j]
                iM=i
                jM=j
    return mx,i,j
```

(d) corrigido

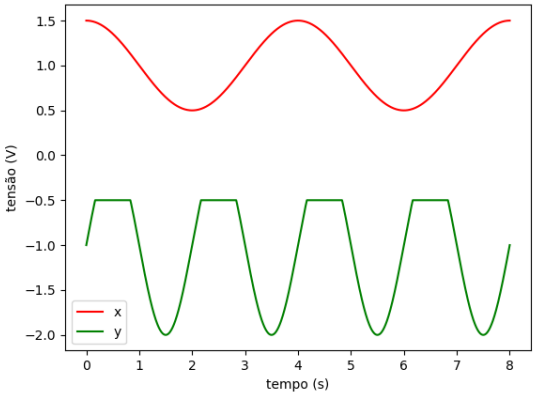
```
def fAmax(M):
    mx=np.abs(M[0,0])
    lin,col=M.shape
    for i in range(lin):
        for j in range(col):
            if np.abs(M[i,j])>mx:
                mx=np.abs(M[i,j])
                iM=i
                jM=j
    return mx,iM,jM
```

2. Escreva uma função que possa ser usada na forma **IS=isSIM(M)** que dada a matriz **M:** verifique se se trata de uma matriz quadrada; caso se trate de uma matriz quadrada verifique se é uma matriz simétrica, devolvendo: 1 se for simétrica, 0 se não for, -1 se não for quadrada.

```
import numpy as np
def isSIM(M):
    lin,col=M.shape
    if lin!=col:
        return -1
    for i in range(lin):
        for j in range(col):
            if M[i,j]!=M[j,i]:
                return 0
    return 1
#teste (não era pedido)
print('simetrica=',isSIM(np.array([[1,2,3],[2,5,7],[3,7,9]])))
print('não simetrica=',isSIM(np.array([[1,2,3],[5,5,7],[3,7,9]])))
print('não quadrada=',isSIM(np.array([[1,2,3],[2,5,7]])))
```

3. Escreva um script que produza a figura, incluindo anotações, com uma resolução no eixo das abcissas de 0.01 s (Nota: as partes curvas das linhas são sinusóides).

```
import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,8,801)
x=1+0.5*np.cos(t*2*np.pi/4.)
plt.plot(t,x,label='x',color='red')
y=-1+np.sin(t*2*np.pi/2.)
for k in range(len(y)):
    y[k]=min(y[k],-0.5)
plt.plot(t,y,label='y',color='green')
plt.legend()
plt.xlabel('tempo (s)')
plt.ylabel('tensão (V)')
```



Nome	Nº	P	G

4. Escreva uma função python para usar forma **dd=dist(ano1,mes1,dia1,ano2,mes2,dia2)** determinando a distância em dias entre as duas datas. Tome em atenção os anos bissextos (admita que os anos bissextos ocorrem de 4 em 4 anos sem exceções). Admita que a segunda data é posterior.

```
def julian(ano,mes,dia):
    ndm=[31,28,31,30,31,30,31,31,30,31,30,31]
    if ano%4==0:
        ndm[1]=29
    jul=0
    for m in range(mes-1):
        jul=jul+ndm[m]
    jul=jul+dia
    return jul
def dist(ano1,mes1,dia1,ano2,mes2,dia2):
    jul1=julian(ano1,mes1,dia1);jul2=julian(ano2,mes2,dia2)
    if ano1==ano2:
        dd=jul2-jul1
        return dd
    dd=julian(ano1,12,31)-jul1
    for ano in range(ano1+1,ano2):
        if ano%4==0:
            nda=366
        else:
            nda=365
        dd=dd+nda
    dd=dd+jul2
    return dd
```

5. O método de Simpson calcula o integral na forma: $\int_{x_0}^{x_N} f(x)dx \approx S_N = \Delta x\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \cdots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right]$. Escreva uma função python que realize esse cálculo.

```
import numpy as np
def simp(f,x):
    n=len(x)
    dx=x[1]-x[0]
    fun=f(x)
    s=(fun[0]+4*fun[1]+fun[n-1])/3.
    for k in range(2,n-1,2):
        s=s+(2*fun[k]+4*fun[k+1])/3.
    s=s*dx
    return s
```

6. Um oscilador harmónico satisfaz a lei de Hooke:

$$F = m\frac{d^2x}{dt^2} = -kx$$

em que $\boldsymbol{k}$ e **m** são constantes. Escreva a equação na forma de um sistema de 2 equações de primeira ordem. Escreva uma função python a usar na forma **v,x=hooke(v0,x0,t,k,m),** que dados os parâmetros (**k,m),** a posição e a velocidade iniciais **(x0,v0)** e um vetor de tempos regularmente espaçados (**t**), calcule as séries temporais da posição (**x**) e da velocidade (**v**) ao longo desse tempo. Utilize o método de Euler, com uma iteração para se aproximar do método do ponto médio.

```
import numpy as np
def hooke(v0,x0,t,k,m):
    N=len(t)
    dt=t[1]-t[0]
    x=np.zeros(t.shape)
    v=np.copy(x)
    x[0]=x0
    v[0]=v0
    for j in range(1,N):
        a=-k/m*x[j-1]
        v[j]=v[j-1]+a*dt
        x[j]=x[j-1]+0.5*(v[j]+v[j-1])*dt
        a=-k/m*0.5*(x[j-1]+x[j])
        v[j]=v[j-1]+a*dt
        x[j]=x[j-1]+0.5*(v[j]+v[j-1])*dt
    return x,v
```