Aula 6

Resolução de sistemas de equações lineares:

Método de Gauss.

Resolver

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 1 \\ 5x_1 + 2x_2 + x_3 = 2 \\ x_1 + x_2 + x_3 = 1 \end{cases}$$

Método:

Modificando progressivamente o sistema com transformações equivalentes:

(a) Substituindo uma equação por uma sua combinação linear com outra

(b) Trocando equações

algoritmo

Usar equação (1) para eliminar x_1:

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 1 \\ (5x_1 + 2x_2 + x_3 = 2) - \frac{1}{2}(10x_1 + 2x_2 + x_3 = 1) \Leftrightarrow 0 + x_2 + 0.5x_3 = 1.5 \\ (x_1 + x_2 + x_3 = 1) - \frac{10x_1 + 2x_2 + x_3 = 1}{10} \Leftrightarrow 0 + 0.8x_2 + 0.9x_3 = 0.9 \end{cases}$$

Do mesmo modo usa-se a nova equação (2) para eliminar x_2 na equação (3).
Resultado: (matriz triangular superior)

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 1 \\ 0 + x_2 + 0.5x_3 = 1.5 \\ 0 + 0 + 0.5x_3 = -0.3 \end{cases}$$

O sistema obtido pode ser resolvido debaixo para cima por substituição.

Algoritmo de eliminação de Gauss ($M = N$)

1º Passo (eliminação)

Transformar o sistema $A\vec{x} = \vec{b}$, no sistema equivalente $U\vec{x} = \vec{c}$, onde U é uma matriz triangular superior, i.e.:

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1N} \\ 0 & u_{22} & \cdots & u_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & u_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \cdots \\ c_N \end{bmatrix}$$

2º passo, resolver de baixo para cima (*backsubstitution*):

$$x_N = \frac{c_N}{u_{NN}}, etc \ldots$$

Eliminação

Deixa-se a 1ª linha sem modificação.

Usa-se a 1ª linha para eliminar todos os coeficientes da 1ª coluna nas linhas abaixo (2,…N). Para a linha 2 será

$$-\frac{a_{21}}{a_{11}}(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = b_1)$$
$$+$$
$$(a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N = b_2)$$
$$\cdots$$
$$0 + \left(a_{22} - \frac{a_{21}}{a_{11}a_{22}}\right)x_2 + \cdots = b_2 - \frac{a_{21}}{a_{11}}b_1$$

Usa-se a nova linha 2 para eliminar $a_{k2}, k \geq 3$, etc … até chegar ao fim.

Só funciona se em cada linha k usada para a eliminação se tiver $a_{kk} \neq 0$.

gaussElim (preliminares)

```
def gaussElim(M,d):
    A=np.copy(M); b=np.copy(d) #Preserva M,d
    x=np.zeros(b.shape)
    Ash=A.shape
    n=Ash[0] #n° linhas de A
    n2=Ash[1] #n° colunas de A
    Bsh=b.shape
    n3=Bsh[0] #n° termos de b
    if n!=n2 or n3!=n or len(Bsh)!=1 or len(Ash)!=2:
        print('Erro de dimensão')
        x=float('nan')
        return x
```

gaussElim

```
def gaussElim(M,d):
    (…)
    for k in range(0,n-1):
        if A[k,k]==0:
            x=float('nan')
            return x
        for j in range(k+1,n): #elimination
            e=A[j,k]/A[k,k]
            for m in range(k,n):
                A[j,m]=A[j,m]-e*A[k,m]
            b[j]=b[j]-e*b[k]
    for k in range(n-1,-1,-1): # backsubstitution
        sum=0.
        for j in range(k+1,n):
            sum=sum+A[k,j]*x[j]
        x[k]=(b[k]-sum)/A[k,k]
    return x
```

Limitações

O algoritmo de Gauss só funciona se o determinante da matriz não for nulo, pois nesse caso as equações não são linearmente independentes (não há solução).

Mesmo nesse caso, falhará se a equação eliminante tiver um zero na diagonal. Essa dificuldade pode ser resolvida trocando essa equação por outra (numa linha inferior) que não tenha o mesmo problema.

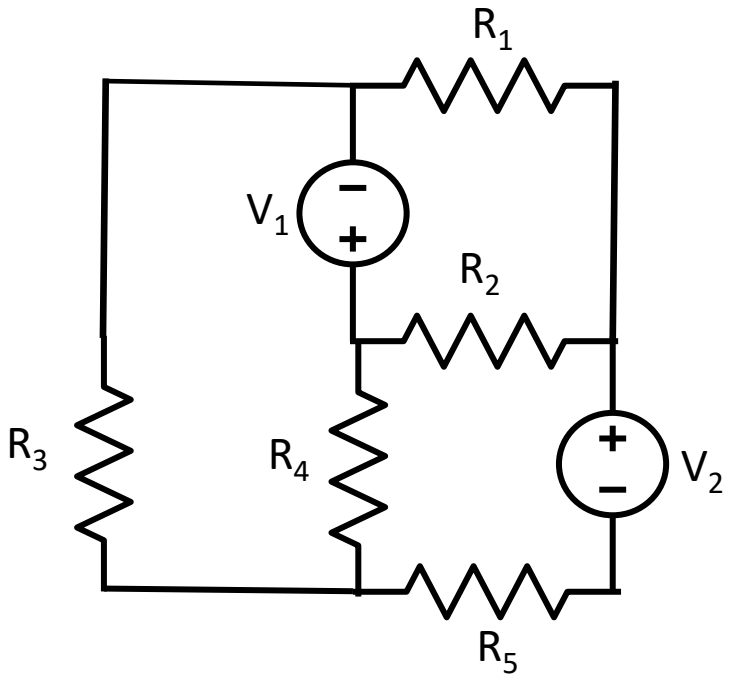
Resolução de um circuito linear (Kirchoff)

Lei das malhas: a queda de tensão ao longo de um circuito fechado é nula (conservação energia)

Leis dos nós: a soma algébrica da corrente num nó é nula (conservação da carga)

Dados $V_{1,2}$ e $R_{1\text{-}5}$ determinar as correntes $I_{1\text{-}5}$

5 incógnitas requerem 5 equações linearmente independentes



Construir a matriz do sistema de equações

Resolver o sistema é fácil (em python)…

A dificuldade pode estar na construção do sistema.

Precisamos de 5 equações linearmente independentes, i.e., para um sistema:

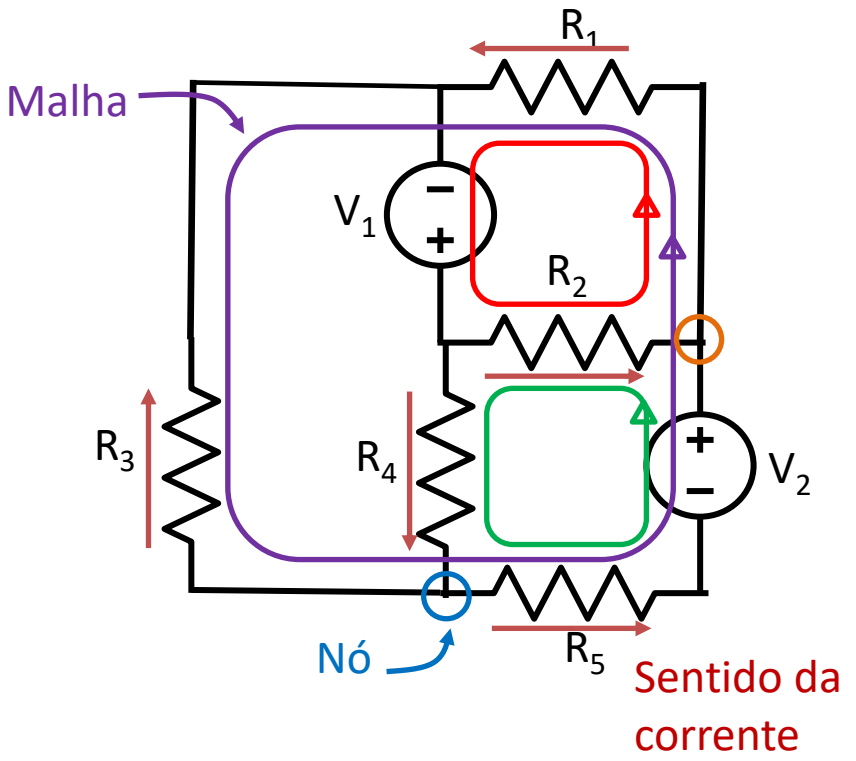
$$M\vec{x} = \vec{b}$$

Terá de ser:

$$\det(M) \neq 0$$

Leis de Kirchoff
3 malhas, 2 nós

$$\begin{cases} R_1 I_1 + R_2 I_2 - V_1 = 0 \\ -R_2 I_2 + R_4 I_4 + R_5 I_5 - V_2 = 0 \\ R_1 I_1 - R_3 I_3 + R_5 I_5 - V_2 = 0 \\ -I_3 + I_4 - I_5 = 0 \\ -I_1 + I_2 + I_5 = 0 \end{cases}$$



O sentido da corrente em cada componente é arbitrado. Se o resultado for negativo, isso quer dizer que, nesse componente, a corrente flui em sentido oposto

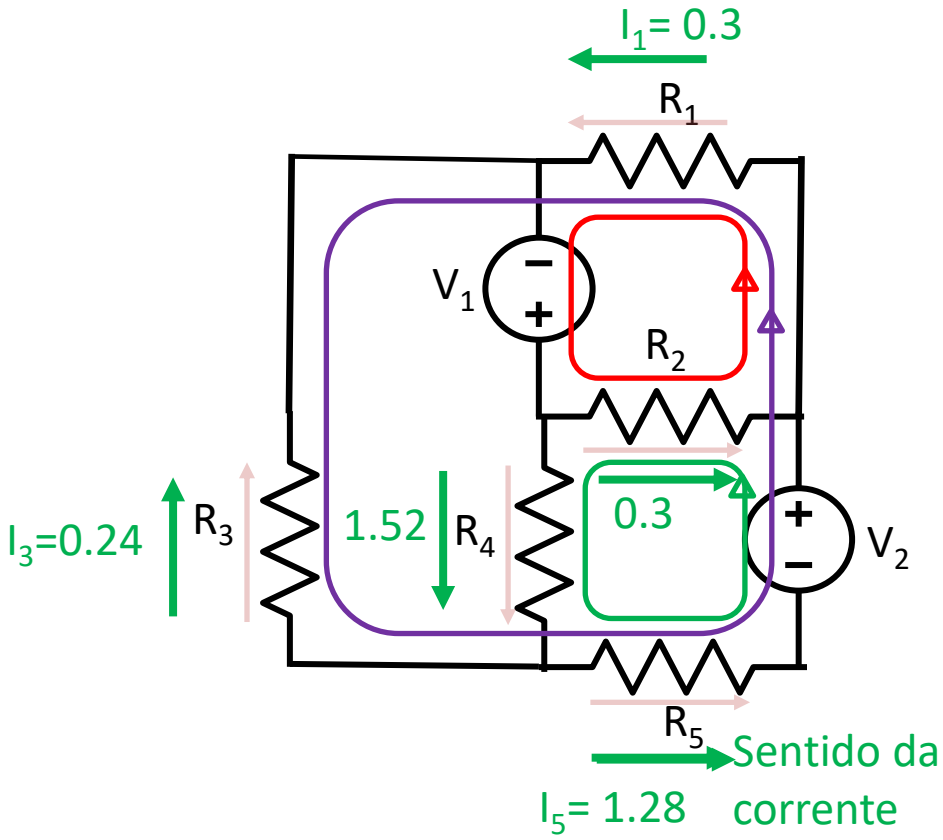
Forma matricial

$$\begin{cases} R_1 I_1 + R_2 I_2 - V_1 = 0 \\ -R_2 I_2 + R_4 I_4 + R_5 I_5 - V_2 = 0 \\ R_1 I_1 - R_3 I_3 + R_5 I_5 - V_2 = 0 \\ -I_3 + I_4 - I_5 = 0 \\ -I_1 + I_2 + I_5 = 0 \end{cases}$$

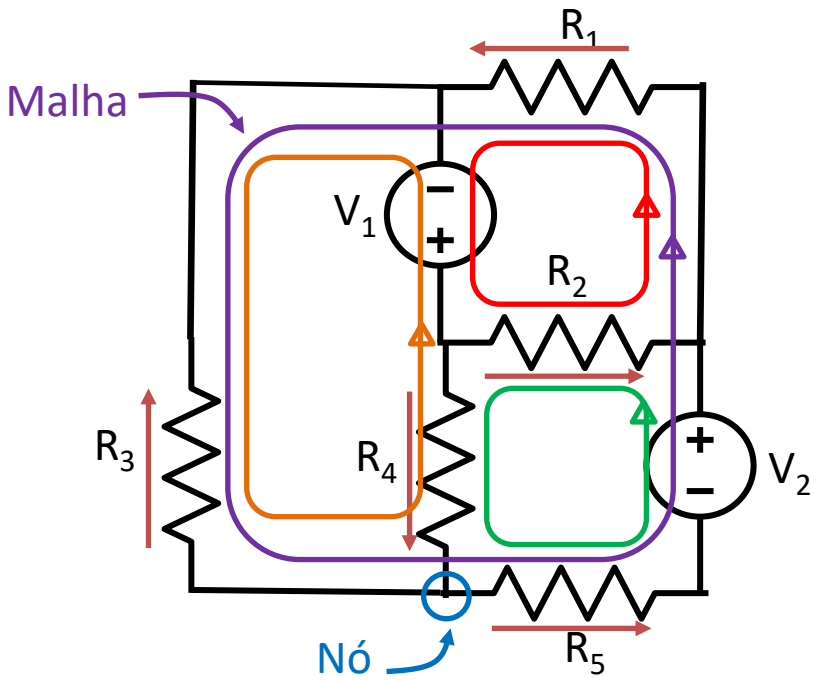
$$\begin{bmatrix} R_1 & R_2 & 0 & 0 & 0 \\ 0 & -R_2 & 0 & R_4 & R_5 \\ R_1 & 0 & -R_3 & 0 & R_5 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_2 \\ 0 \\ 0 \end{bmatrix}$$

```
import numpy as np
R1=15;R2=18;R3=10;R4=5;R5=14
V1=10;V2=20
nI=5
M=np.array([[R1,R2,0,0,0],\
    [0,-R2,0,R4,R5],\
    [R1,0,-R3,0,R5],\
    [0,0,-1,1,-1],\
    [-1,1,0,0,0]],dtype=float)
b=np.array([V1,V2,V2,0,0],dtype=float)
I=np.linalg.solve(M,b) #I=gaussElim(M,b)
print(I)
>>[ 0.3030303   0.3030303   0.24125874  1.51748252
1.27622378]
```

```
I=[ 0.3030303   0.3030303
0.24125874  1.51748252  1.27622378]
```



4 malhas, 1 nó



$$\begin{cases} R_1 I_1 + R_2 I_2 - V_1 = 0 \\ -R_2 I_2 + R_4 I_4 + R_5 I_5 - V_2 = 0 \\ R_1 I_1 - R_3 I_3 + R_5 I_5 - V_2 = 0 \\ -I_3 + I_4 - I_5 = 0 \\ -R_3 I_3 - R_4 I_4 + V_1 = 0 \end{cases} \Leftarrow$$

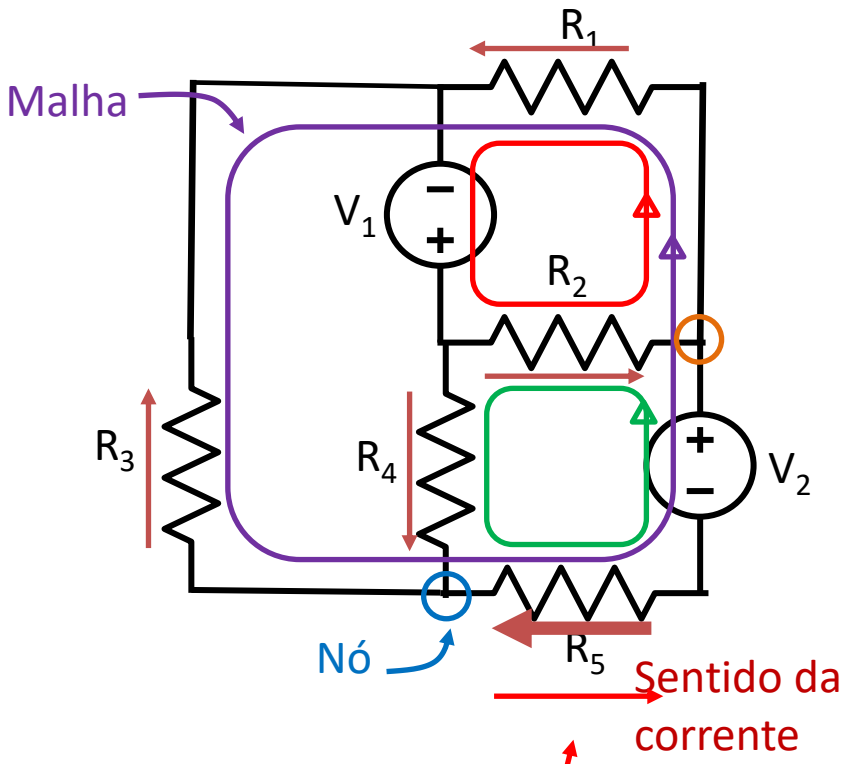
```
>>LinAlgError: Singular matrix
np.linalg.det(M)
>>0.0
```

Não funciona porque as equações não são linearmente independentes

3 malhas, 2 nós
≠prescrição de sentido

$$\begin{cases} R_1 I_1 + R_2 I_2 - V_1 = 0 \\ -R_2 I_2 + R_4 I_4 - R_5 I_5 - V_2 = 0 \\ R_1 I_1 - R_3 I_3 - R_5 I_5 - V_2 = 0 \\ -I_3 + I_4 + I_5 = 0 \\ -I_1 + I_2 - I_5 = 0 \end{cases}$$

```
I=[ 0.3030303   0.3030303
0.24125874  1.51748252
-1.27622378]
```



Outra solução direta numpy (mais lenta)

```
import numpy as np
R1=15;R2=18;R3=10;R4=5;R5=14
V1=10;V2=20
nI=5
M=np.array([[R1,R2,0,0,0],\
    [0,-R2,0,R4,R5],\
    [R1,0,-R3,0,R5],\
    [0,0,-1,1,-1],\
    [-1,1,0,0,0]],dtype=float)
b=np.array([V1,V2,V2,0,0],dtype=float)
I=np.matmul(np.linalg.inv(M),b)
print(I)
>>[0.3030303  0.3030303  0.24125874 1.51748252
1.27622378] ok
```

$$\begin{bmatrix} R_1 & R_2 & 0 & 0 & 0 \\ 0 & -R_2 & 0 & R_4 & R_5 \\ R_1 & 0 & -R_3 & 0 & R_5 \\ 0 & 0 & -1 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \end{bmatrix} = \begin{bmatrix} V_1 \\ V_2 \\ V_2 \\ 0 \\ 0 \end{bmatrix}$$

Comentários

A solução de problemas regidos por sistemas de equações lineares é simples. A dificuldade pode estar no correto estabelecimento do sistema de equações.

Os arrays devem ser sempre explicitamente declarados comO float, para evitar ambiguidade e erros.

É sempre possível testar a solução:

```
x=np.linalg.solve(M,b)
B=np.matmul(M,x)
print((B-b)/b) #erro de arredondmento
```

Gestão de dados

$f(t), f(x,y), f(x,y,z), f(x,y,z,t), f(\lambda,\phi)$

Muitos dados de interesse representam séries temporais $f(t)$, distribuições espaciais $f(x,y,z)$, mapas georeferenciados $f(\lambda,\phi)$

Em python esses dados são descritos por objetos **`np.array`** de diferente dimensionalidade (**`shape`**).

A estrutura desses dados pode ser complicada:

- Os dados georeferenciados (longitude,latitude,altitude) usam coordenadas esféricas (cícilicas)
- Os dados temporais seguem as regras do calendário (meses e anos de duração variável)

Leitura de dados estruturados

Dados de pouca complexidade podem ser lidos/escritos em ascii, com funções numpy:

```
a=np.loadtxt('ficheiro.txt',skiprows=0)
```

Funciona se os dados do ficheiro tiverem a forma de uma tabela $(m \times n)$ i.e. de um array numpy

Mas se se tiver feito

```
np.savetxt('f2.txt',[a,b,c])
```

pode fazer-se

```
a,b,c=np.loadtxt('f2.txt')
```

pois cada um dos objetos (`a,b,c`) terá uma forma de array, e podem ter diferente `shape`

Dados de média complexidade

O formato xls/xlsx permite uma gestão simples de dados, utilizando a estrutura rígida (mas muito abrangente) das folhas de cálculo, desde que esses dados não sejam demasiado extensos e possam ser organizados em tabelas bidimensionais.

As séries temporais e os mapas podem ser facilmente transferidos (ler/escrever) neste formato.

Exemplo

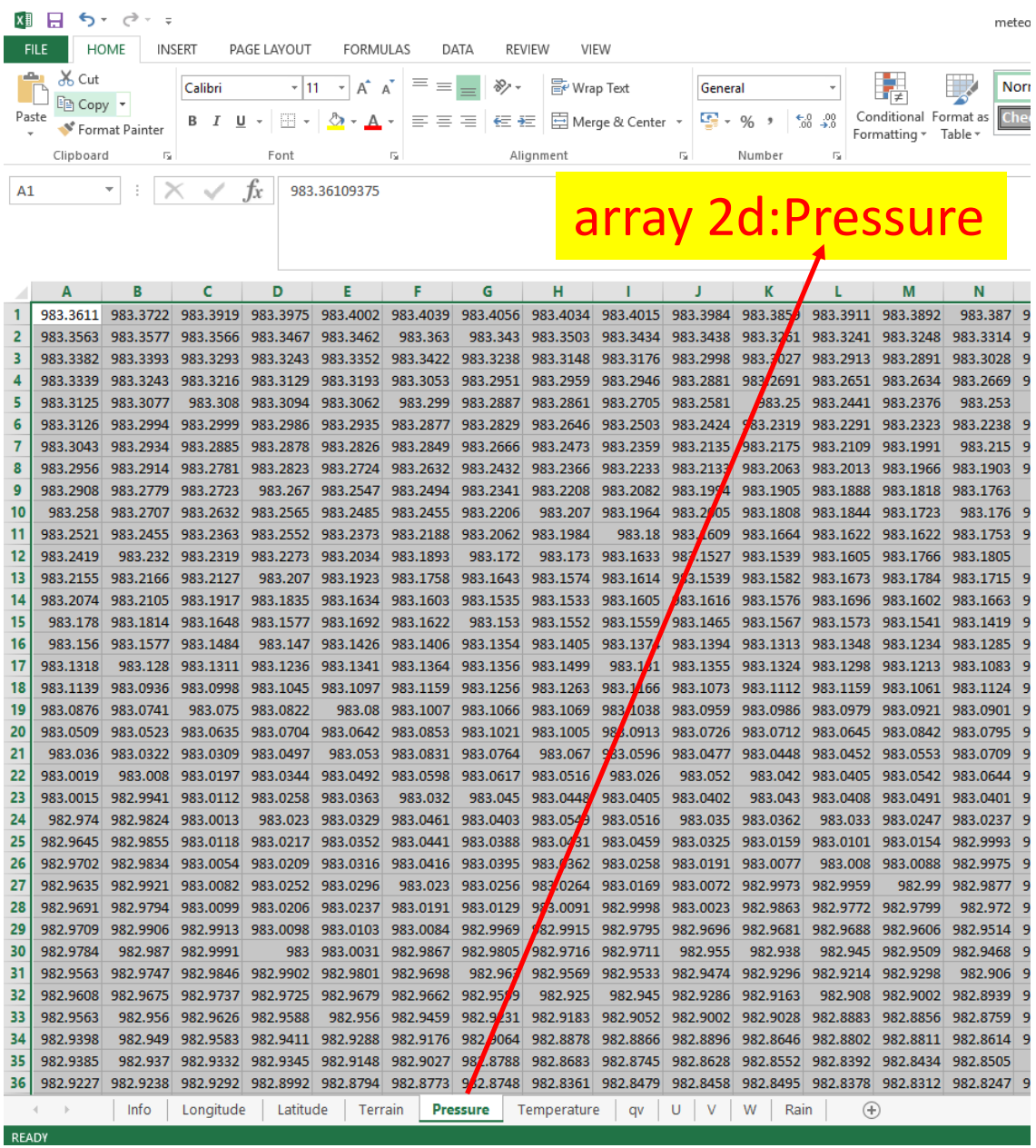
escalares

	A	B	C	D
1	Variable	units	level	height
2	Longitude	deg		
3	Latutide	deg		
4	Pressure	hPa	4	470m
5	Temperature	K	4	470m
6	qv	g/kg	4	470m
7	U	m/s	4	470m
8	V	m/s	4	470m
9	W	m/s	4	470m
10	Rain	mm	0	
11	Terrain	m	0	
12				
13	year	2015		
14	month	6		
15	day	9		
16	hour	22		
17	min	0		
18	sec	0		
19				
20				

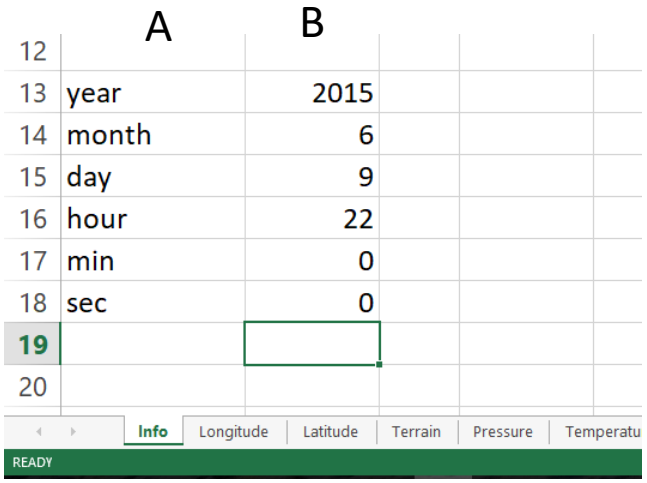
Info

Info | Longitude | Latitude | Terrain | Pressure | Temperatu

READY



openpyxl, datetime



	A	B
12		
13	year	2015
14	month	6
15	day	9
16	hour	22
17	min	0
18	sec	0
19		
20		

Info | Longitude | Latitude | Terrain | Pressure | Temperatu

READY

```
import numpy as np
import openpyxl as pyxl
import datetime
dados='meteo_model.xlsx'
wb=pyxl.load_workbook(dados) #abre o workbook
wsI=wb['Info'] #abre a worksheet Info
ano=wsI['B13'].value #lê célula
mes=wsI['B14'].value
dia=wsI['B15'].value
hora=wsI['B16'].value
min=wsI['B17'].value
seg=wsI['B18'].value
tempo=datetime.datetime(ano,mes,dia,hora,min,seg)
print(tempo)
>>2015-06-09 22:00:00
```

Ler tabela completa

2 formas de ler célula:
`ws.cell(row=2,column=4).value`
≡`ws['B4'].value`

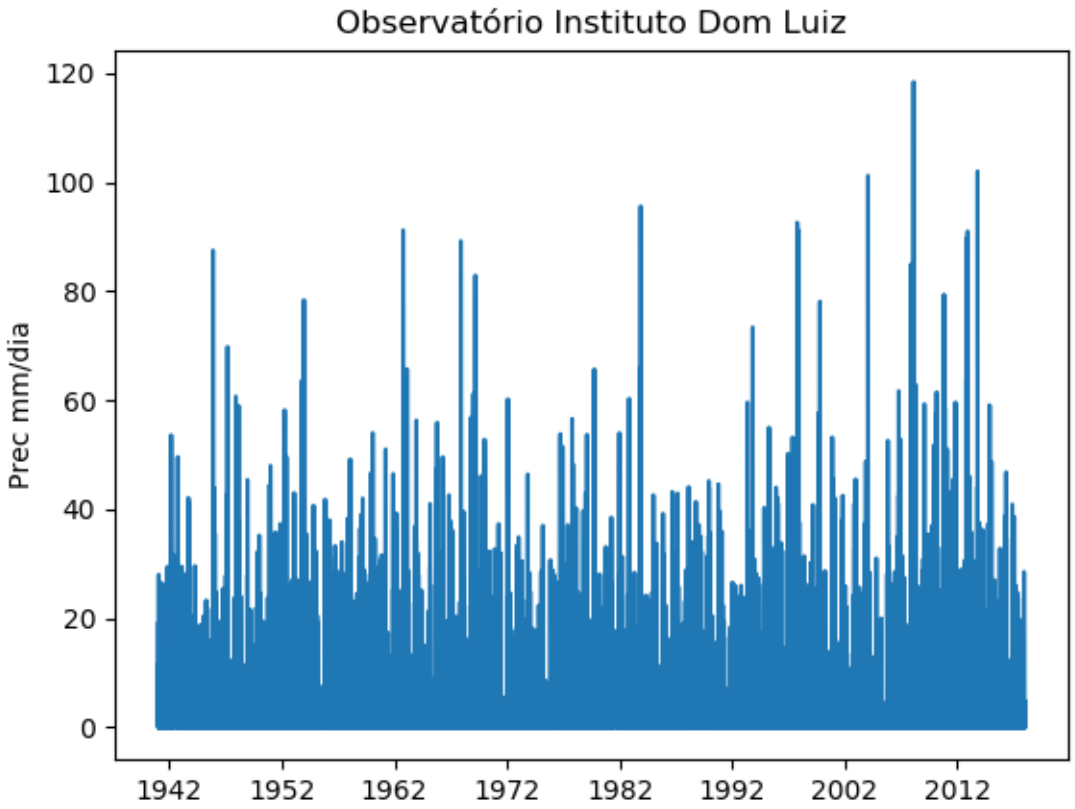
```
import numpy as np
import openpyxl as pyxl
dados='meteo_model.xlsx'
wb=pyxl.load_workbook(dados)
ws=wb['Pressure']
rows=ws.max_row #identifica dimensão da worksheet
cols=ws.max_column
pressure=np.zeros((rows,cols))
for r in range(rows):
    for c in range(cols):
        pressure[r,c]=ws.cell(row=r+1,\
                column=c+1).value
```

Leitura de série temporal

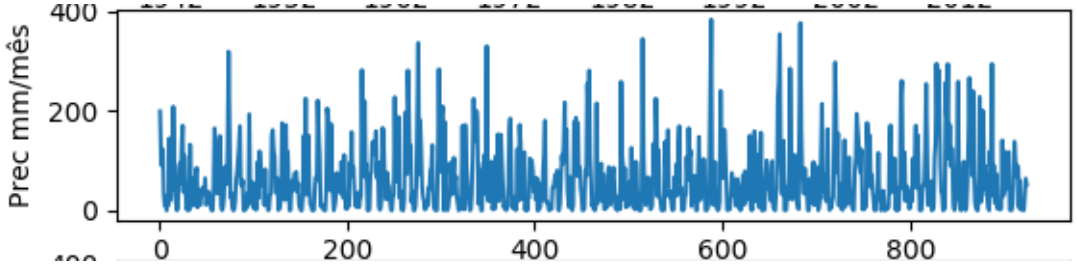
```
import numpy as np
import datetime
import matplotlib.pyplot as plt
dados=np.loadtxt('prec24h_535_2.dat')
ano=np.array(dados[:,0],dtype=int);
mes=np.array(dados[:,1],dtype=int);
dia=np.array(dados[:,2],dtype=int);
prec=dados[:,3]
del dados
tempo=[]
for kd in range(len(prec)):
  tempo.append(datetime.datetime\
  (ano[kd],mes[kd],dia[kd]))

plt.plot(tempo,prec)
plt.ylabel('Prec mm/dia')
plt.title('Instituto Dom Luiz')
plt.savefig('IDL_Prec_1941_2017.png')
```

	Ano	Mês	Dia	Prec
0000	* * * Top of File * * *			
0001	1941	1	1	19.0000000
0002	1941	1	2	9.19999981
0003	1941	1	3	0.00000000
0004	1941	1	4	0.00000000
0005	1941	1	5	0.00000000
0006	1941	1	6	0.00000000
0007	1941	1	7	0.00000000
0008	1941	1	8	11.5000000
0009	1941	1	9	0.00000000
0010	1941	1	10	5.69999981
0011	1941	1	11	2.09999990
0012	1941	1	12	4.90000010
0013	1941	1	13	9.39999962
0014	1941	1	14	0.400000006
0015	1941	1	15	2.59999990
0016	1941	1	16	5.50000000
0017	1941	1	17	1.20000005
0018	1941	1	18	0.100000001
0019	1941	1	19	5.40000010
0020	1941	1	20	11.3999996
0021	1941	1	21	28.0000000
0022	1941	1	22	25.6000004
0023	1941	1	23	9.30000019
0024	1941	1	24	17.5000000
0025	1941	1	25	0.00000000
0026	1941	1	26	6.40000010
0027	1941	1	27	1.79999995
0028	1941	1	28	4.40000010
0029	1941	1	29	5.40000010
0030	1941	1	30	0.800000012
0031	1941	1	31	11.3000002
0032	1941	2	1	3.29999995
0033	1941	2	2	0.00000000
0034	1941	2	3	13.1999998
0035	1941	2	4	0.00000000
0036	1941	2	5	0.00000000

Série temporal diária



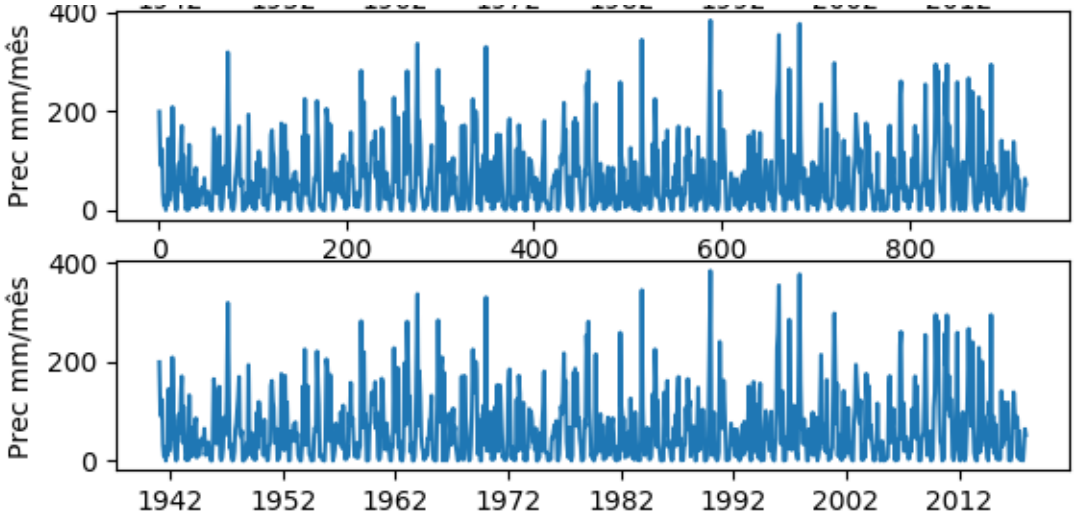
Total mensal




```
ndMES=[31,28,31,30,31,30,31,31,30,31,30,31]
anoFIRST=np.min(ano); anoLAST=np.max(ano)
numMES=12*(anoLAST-anoFIRST+1) #número total de meses
precM=np.zeros((numMES),dtype=float)
mJ=-1; dJ=-1
for yy in range(anoFIRST,anoLAST+1):
    for mm in range(1,12+1):
        mJ=mJ+1
        ndm=ndMES[mm-1]
        if mm==2 and (yy%4==0 and (yy%100!=0 or yy%400==0)):
            ndm=ndm+1
        for dd in range(ndm):
            dJ=dJ+1
            precM[mJ]=precM[mJ]+prec[dJ]
plt.subplot(3,1,2)
plt.plot(np.linspace(1,numMES,numMES),precM)
plt.ylabel('Prec mm/mês')
```

Ano bissexto

Total mensal c/datetime




```
mesLIST=[]
for yy in range(anoFIRST,anoLAST+1):
    for mm in range(1,13):
        mesLIST.append(datetime.datetime(yy,mm,15))
plt.subplot(3,1,3)
plt.plot(mesLIST,precM)
plt.ylabel('Prec mm/mês')
```

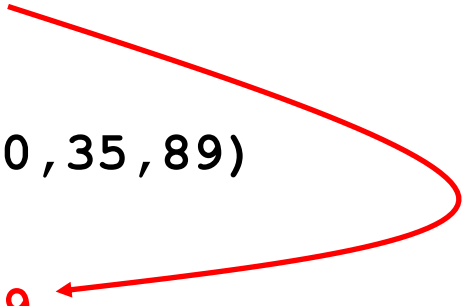
Datas julianas

Contagem de dias sucessivos desde uma data de referência.

Pode ser referido a um ano:

$$dia\_juliano \in [1{,}365\ ou\ 366]$$

Ou a um periodo anos. Em **datetime** os dias julianos são contados a partir de 0001-01-01 (1º dia do calendário moderno)



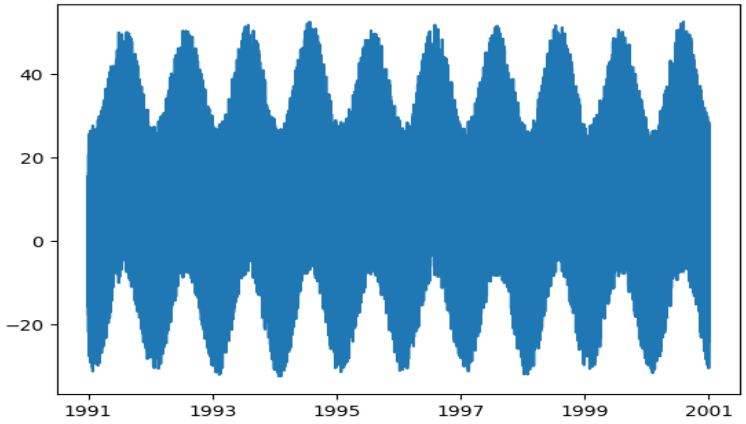
```
X=datetime.datetime(2018,3,16,12,00,35,89)
print(X,X.toordinal())
>>2018-03-16 12:00:35.000089 736769
X=datetime.datetime(1,1,1,12,00,35,89)
print(X,X.toordinal())
>>0001-01-01 12:00:35.000089 1
```

datetime
calendar

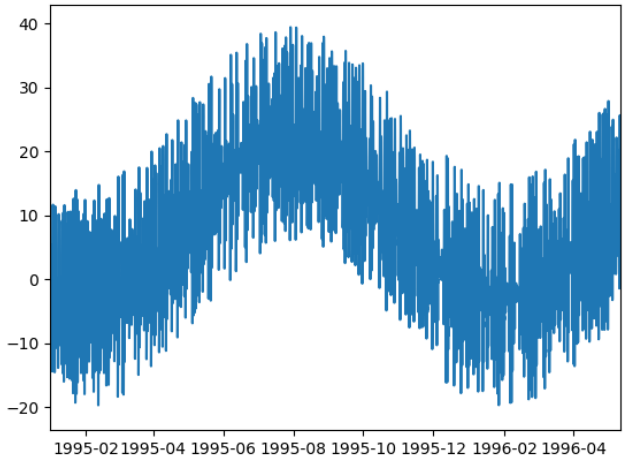
Dados sintéticos

```
import numpy as np
import datetime;import calendar
ySTART=1991;yEND=2000
dataSTA=datetime.datetime(ySTART,1,1,0);
dataEND=datetime.datetime(yEND,12,31,23)
numOBS=(dataEND.toordinal()-dataSTA.toordinal()+1)*24
T=np.zeros((numOBS),dtype=float);kh=-1
for ano in range(ySTART,yEND+1):
    AmpA=12+2*(np.random.rand()-0.5) #AmpA∈[11,13] media 12
    if calendar.isleap(ano):
        ndays=366
    else:
        ndays=365
    for julian in range(1,ndays+1):
        AmpD=10+10*(np.random.rand()) #AmpD∈[10,30] media 20
        for hour in range(0,24):
            kh=kh+1
            T[kh]=10+AmpD*np.sin(2*hour*np.pi/24.+4*np.pi/3)\
            +AmpA*np.sin(2*julian*np.pi/ndays+4*np.pi/3)
```

Datetime
.timedelta

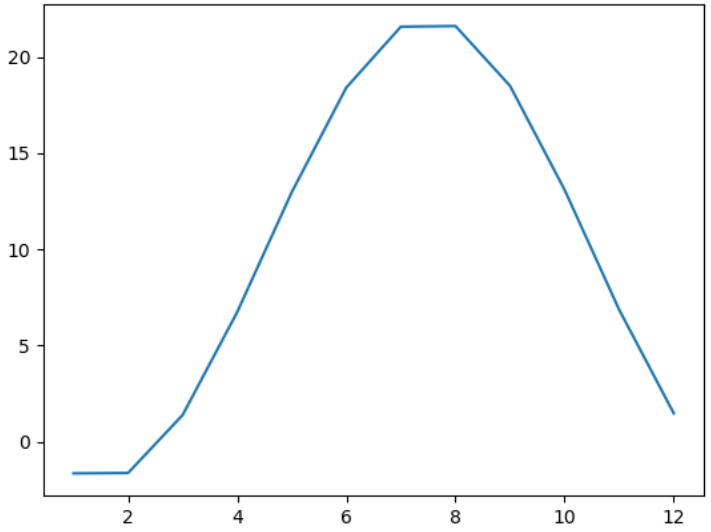



```
import maptplotlib.pyplot as plt
time0=dataSTA
dateList = [] #empty list
for hs in range(0, numOBS):
    dateList.append(time0+datetime.timedelta(hours=hs))
plt.plot(dateList,T)
```



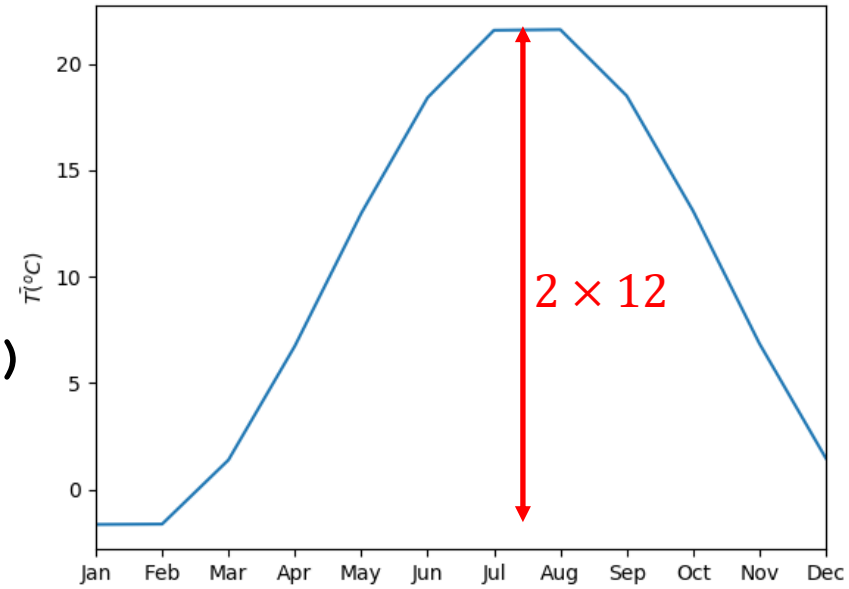

```
plt.figure()
plt.plot(dateList,T)
plt.xlim(datetime.datetime(1995,1,1),\
         datetime.datetime(1996,5,12))
plt.savefig('serie_t_extrato.png')
```

Ano médio




```
Tmes=np.zeros((12),dtype=float)
Nmes=np.zeros((12),dtype=int)
for kh in range(len(T)):
    mes=dateList[kh].month-1
    Tmes[mes]=Tmes[mes]+T[kh]
    Nmes[mes]=Nmes[mes]+1
Tmes=Tmes/Nmes
plt.plot(np.linspace(1,12,12),Tmes)
plt.xlim(1,12)
plt.xlabel('mês')
plt.ylabel(r'$T\bar (^oC)$')
```

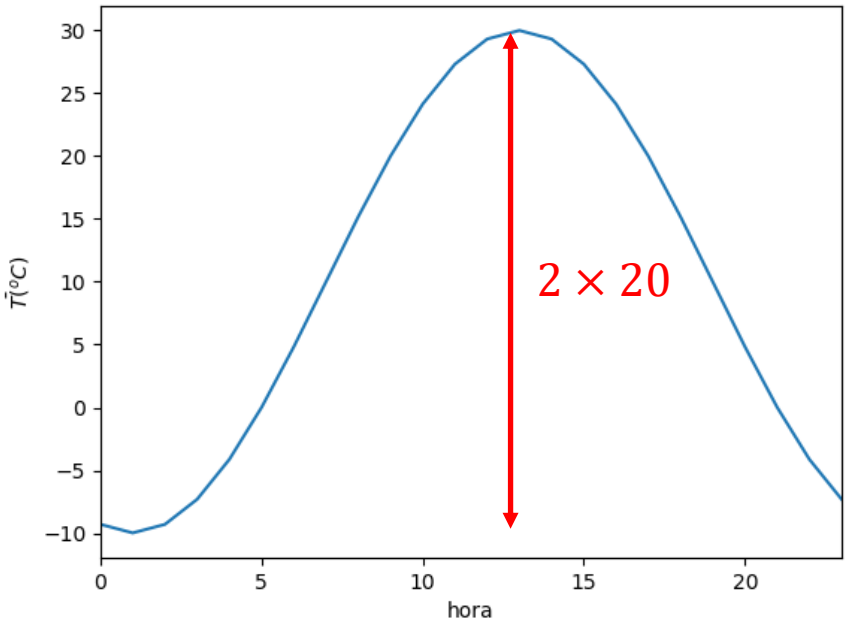
Ano médio
versão 2




```
Tmes=np.zeros((12),dtype=float)
Nmes=np.zeros((12),dtype=int)
for kh in range(len(T)):
    mes=dateList[kh].month-1
    Tmes[mes]=Tmes[mes]+T[kh]
    Nmes[mes]=Nmes[mes]+1
Tmes=Tmes/Nmes
my_xticks = ['Jan','Feb','Mar','Apr','May',\
    'Jun','Jul','Aug','Sep','Oct','Nov','Dec']
plt.xticks(np.linspace(1,12,12), my_xticks)
plt.plot(np.linspace(1,12,12),Tmes)
plt.xlim(1,12)
plt.ylabel(r'$T\bar (^oC)$')
```

Dia médio

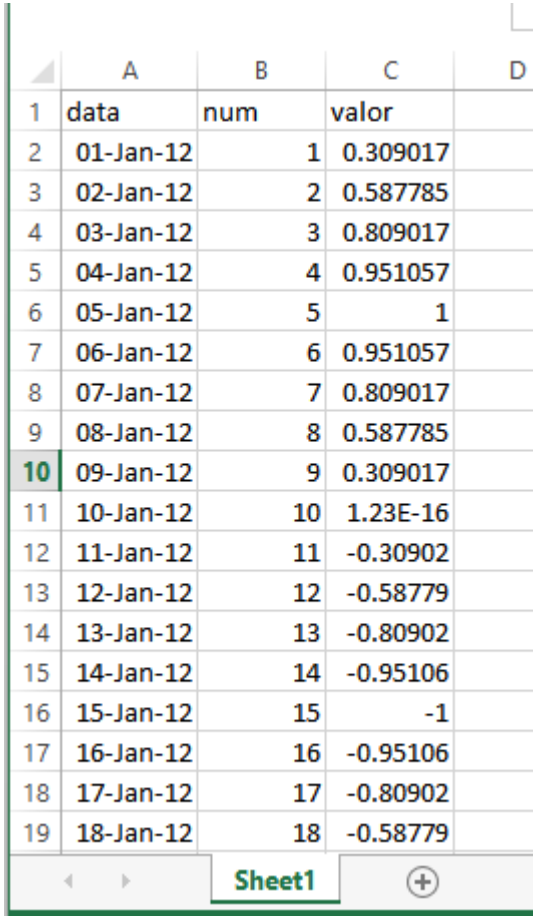
```
Th=np.zeros((24),dtype=float
Nh=np.zeros((24),dtype=int)
for kh in range(len(T)):
    hora=dateList[kh].hour-1
    Th[hora]=Th[hora]+T[kh]
    Nh[hora]=Nh[hora]+1
Th=Th/Nh
plt.figure()
plt.plot(np.linspace(0,23,24),Th)
plt.xlim(0,23)
plt.xlabel('hora')
plt.ylabel(r'$T\bar (^oC)$')
```



Nota: o ruído desapareceu na média

datas excel

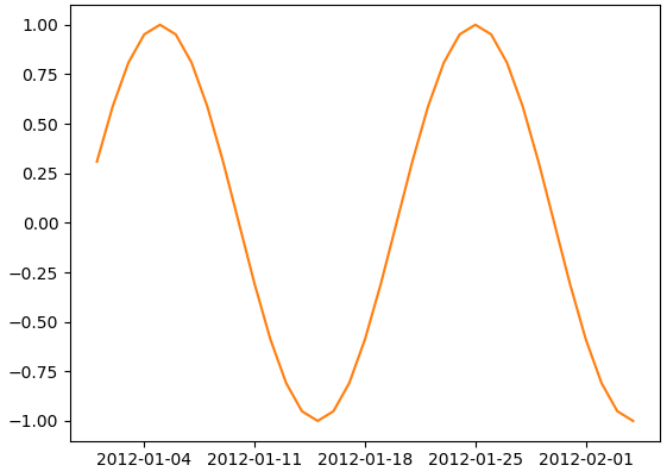
```
import numpy as np
import datetime
import openpyxl as pyxl
import matplotlib.pyplot as plt
wb=pyxl.load_workbook('data_xls.xlsx',\
    data_only=True)
ws=wb['Sheet1']
rows=ws.max_row
datas=[] #lista
valor=np.zeros((rows-1))
for r in range(2,rows+1):
    datas.append(ws.cell(row=r,column=1).value)
    valor[r-2]=ws.cell(row=r,column=3).value
for r in range(len(datas)):
    print(r,datas[r])
plt.plot(datas,valor)
```



	A	B	C	D
1	data	num	valor	
2	01-Jan-12	1	0.309017	
3	02-Jan-12	2	0.587785	
4	03-Jan-12	3	0.809017	
5	04-Jan-12	4	0.951057	
6	05-Jan-12	5	1	
7	06-Jan-12	6	0.951057	
8	07-Jan-12	7	0.809017	
9	08-Jan-12	8	0.587785	
10	09-Jan-12	9	0.309017	
11	10-Jan-12	10	1.23E-16	
12	11-Jan-12	11	-0.30902	
13	12-Jan-12	12	-0.58779	
14	13-Jan-12	13	-0.80902	
15	14-Jan-12	14	-0.95106	
16	15-Jan-12	15	-1	
17	16-Jan-12	16	-0.95106	
18	17-Jan-12	17	-0.80902	
19	18-Jan-12	18	-0.58779	

Sheet1

```
plt.plot(datas,valor)
```



	A	B	C	D
1	data	num	valor	
2	01-Jan-12	1	0.309017	
3	02-Jan-12	2	0.587785	
4	03-Jan-12	3	0.809017	
5	04-Jan-12	4	0.951057	
6	05-Jan-12	5	1	
7	06-Jan-12	6	0.951057	
8	07-Jan-12	7	0.809017	
9	08-Jan-12	8	0.587785	
10	09-Jan-12	9	0.309017	
11	10-Jan-12	10	1.23E-16	
12	11-Jan-12	11	-0.30902	
13	12-Jan-12	12	-0.58779	
14	13-Jan-12	13	-0.80902	
15	14-Jan-12	14	-0.95106	
16	15-Jan-12	15	-1	
17	16-Jan-12	16	-0.95106	
18	17-Jan-12	17	-0.80902	
19	18-Jan-12	18	-0.58779	

Sheet1

```
for r in range(len(datas)):
    print(r,datas[r])
0 2012-01-01 00:00:00
1 2012-01-02 00:00:00
2 2012-01-03 00:00:00
3 2012-01-04 00:00:00
...
34 2012-02-04 00:00:00
35 2012-02-05 00:00:00
```

Objetos datetime (hh:mm:ss a zero)

Dados estruturados mais complicados

Dados em mais de 2 dimensões, ou de grande dimensão precisam de ser transferidos por sistemas de dados mais avançados, que permitam o acesso direto rápido a subconjuntos.

Um único ficheiro pode conter várias variáveis de uma simulação em 4 dimensões (x,y,z,t). Se pretender conhecer a evolução da temperatura num ponto T(x0,y0,z0,t) só quero ler (rapidamente) a série temporal respectiva.

O python suporta um sistema deste tipo, designado por netCDF4, que é utilizado na transferência de dados meteorológicos, climáticos, imagens de satélite.