Aula 6

Resolução de sistemas de equações lineares:

Método de Gauss.

Input/output

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 ($\boldsymbol{M = N}$)

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

$$-\frac{a_{21}}{a_{11}}(a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = b_1)$$

$$+$$

$$(a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N = b_2)$$

…

$$0 + \left(a_{22} - \frac{a_{21}}{a_{11}a_{22}}\right)x_2 + \cdots = b_2 - \frac{a_{21}}{a_{11}}b_1$$

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
    #melhor: np.copy(M).astype(int)
    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.

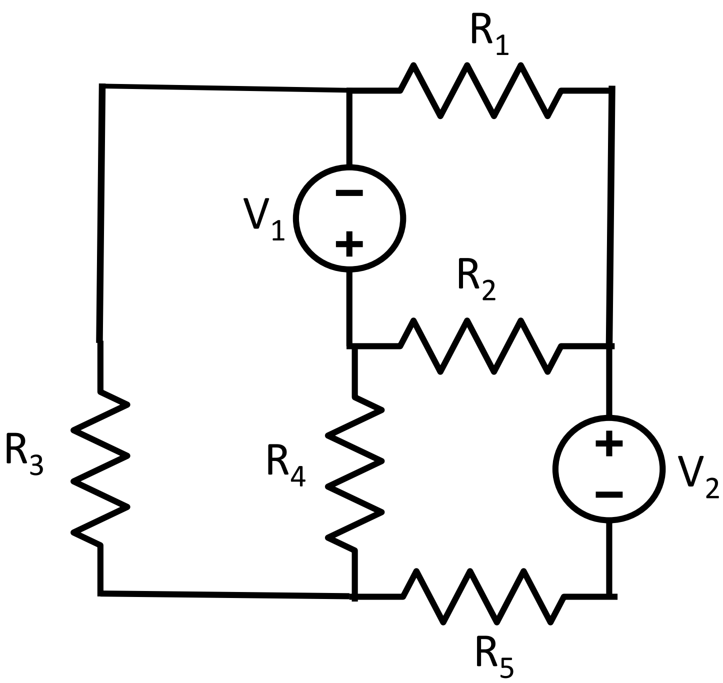
Resolução de um circuito linear (Kirchoff)

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:

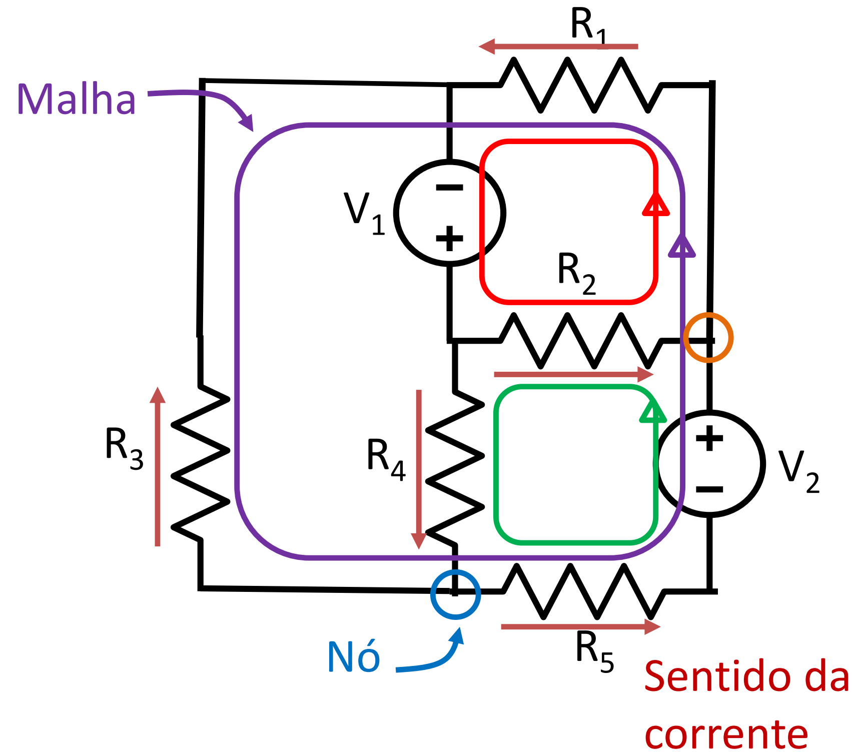
$$M\vec{x} = \vec{b}$$

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

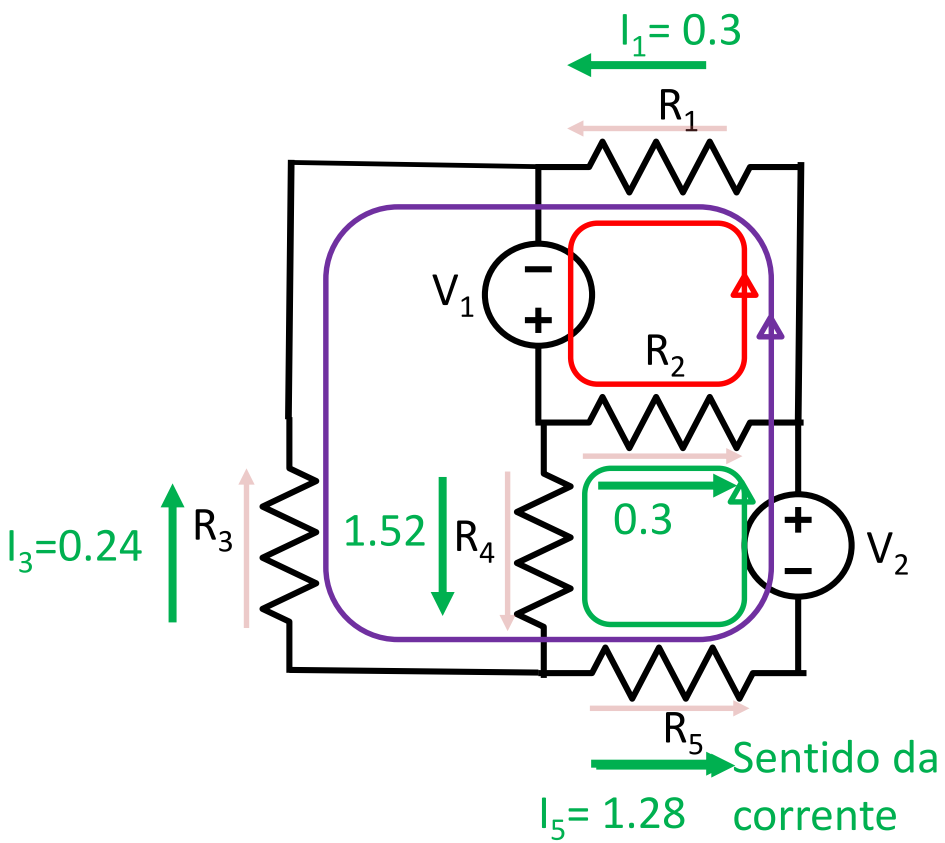
Forma matricial

$$\begin{cases} R_1I_1 + R_2I_2 - V_1 = 0 \\ -R_2I_2 + R_4I_4 + R_5I_5 - V_2 = 0 \\ R_1I_1 - R_3I_3 + R_5I_5 - V_2 = 0 \\ -I_3 + I_4 - I_5 = 0 \\ -I_1 + I_2 + I_5 = 0 \end{cases}$$

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

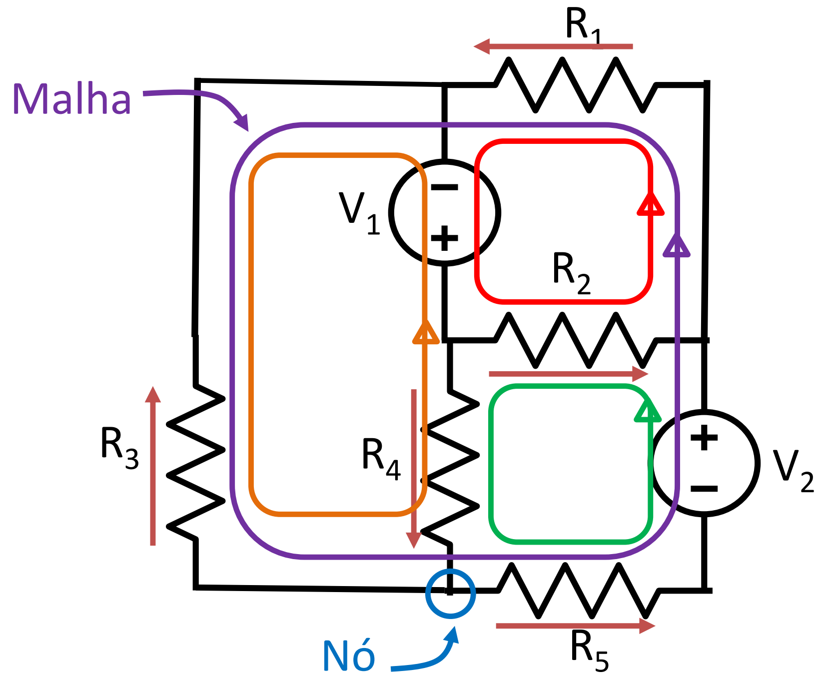
```
import numpy as np
R1=15;R2=18;R3=10;R4=5;R5=14
V1=10;V2=20
nI=5
M=np.array([[R1,R2,0,0,0],\
    [0,-R2,0,R4,R5],\
    [R1,0,-R3,0,R5],\
    [0,0,-1,1,-1],\
    [-1,1,0,0,0]],dtype=float)
b=np.array([V1,V2,V2,0,0],dtype=float)
I=np.linalg.solve(M,b) #I=gaussElim(M,b)
print(I)
>>[ 0.3030303   0.3030303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

I=[0.3030303 0.3030303 0.24125874 1.51748252 1.27622378]



4 malhas, 1 nó

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



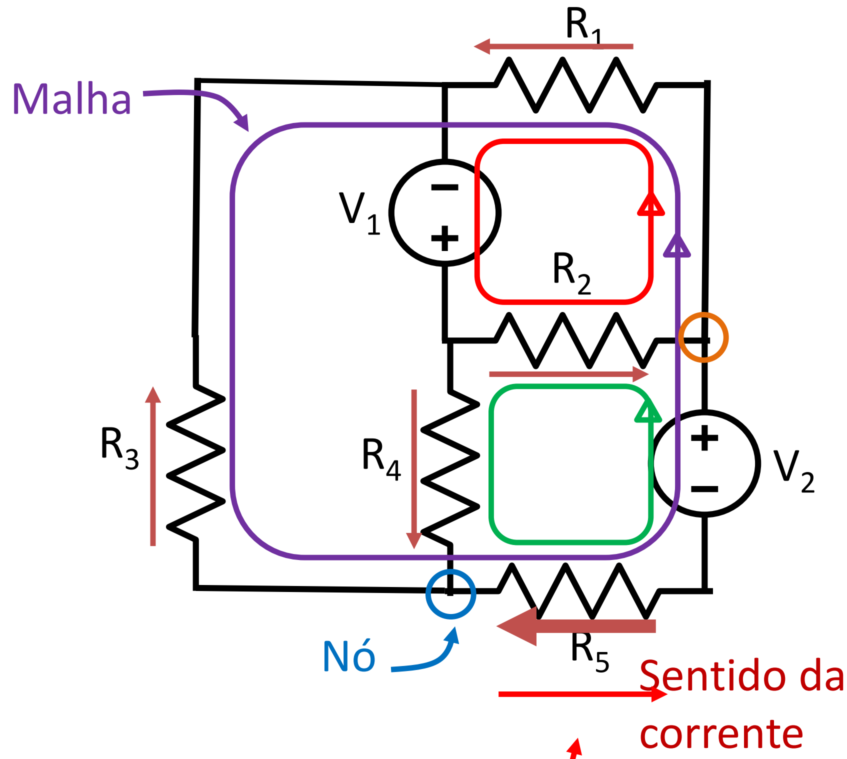

```
>>LinAlgError: Singular matrix
np.linalg.det(M)
>>0.0
```

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 com 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

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

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

Em python esses dados podem ser 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')`

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`

ESCRITA de dados ascii

```
f=open('dados.dat')
for k in range(len(sec)):
     f.write('%4i %4.1f’ % (sec[k],T[k])
```

LEITURA de dados ascii

```
d=open('D1.dat','r')
for k in d:
    ano,mes,dia,hora,T,RH,P=\
          d.readline().split()
    print(ano,mes,dia,hora,T,RH,P)
```

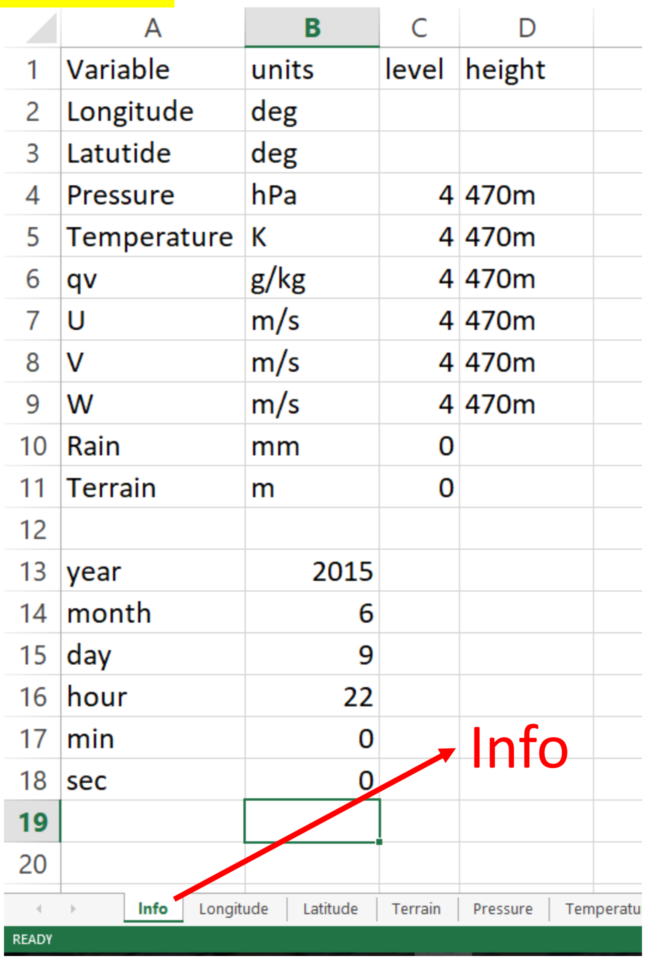
Dados de média complexidade

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.

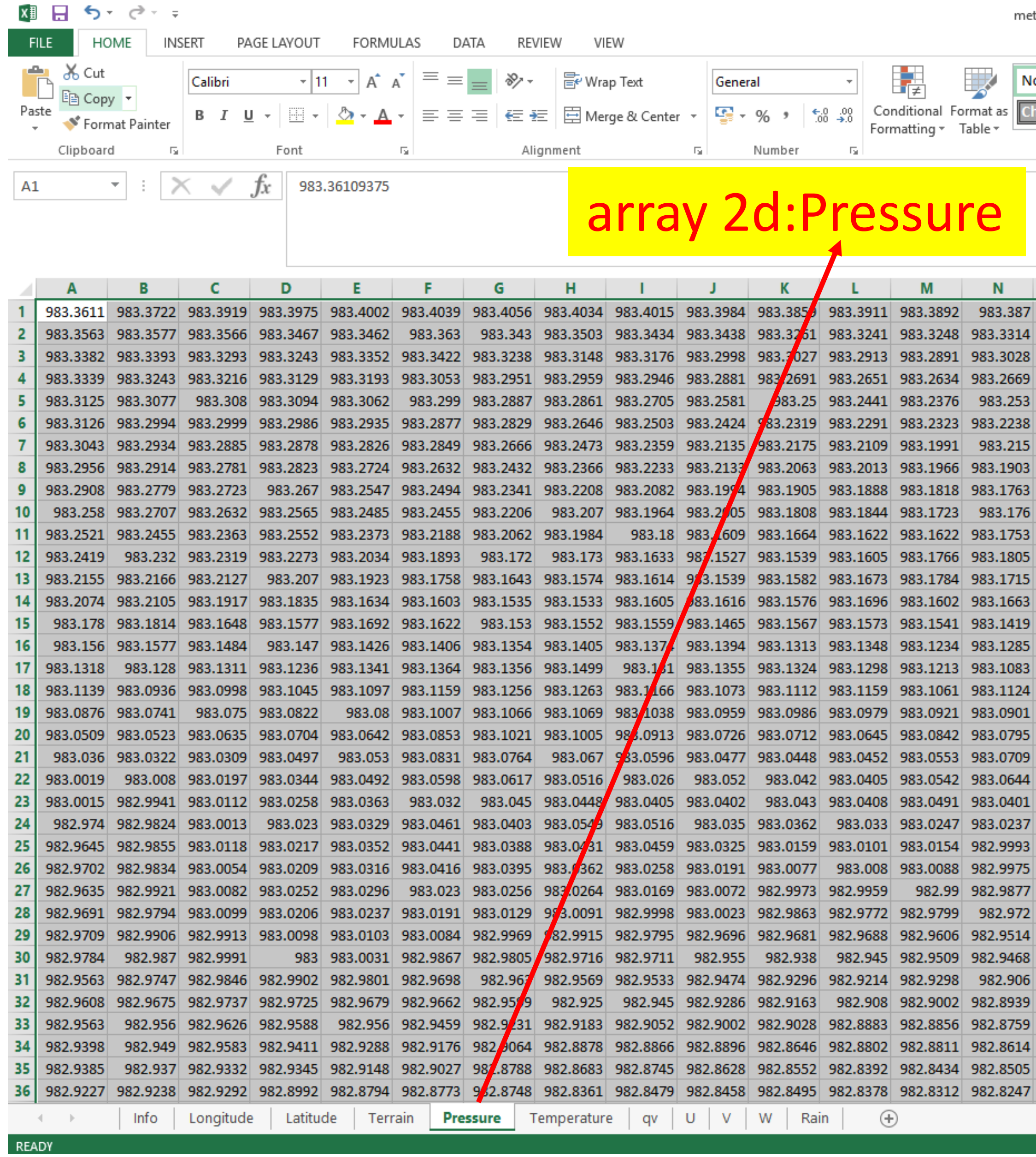
Exemplo

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				

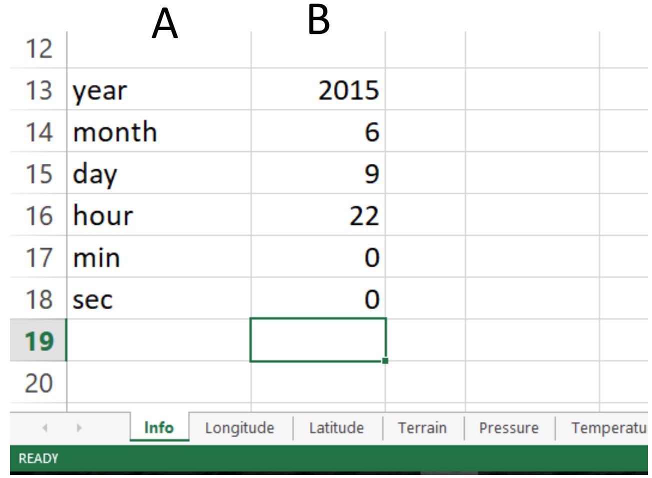
Info



array 2d:Pressure

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	983.3611	983.3722	983.3919	983.3975	983.4002	983.4039	983.4056	983.4034	983.4015	983.3984	983.3859	983.3911	983.3892	983.387
2	983.3563	983.3577	983.3566	983.3467	983.3462	983.363	983.343	983.3503	983.3434	983.3438	983.32[illegible]1	983.3241	983.3248	983.3314
3	983.3382	983.3393	983.3293	983.3243	983.3352	983.3422	983.3238	983.3148	983.3176	983.2998	983.[illegible]027	983.2913	983.2891	983.3028
4	983.3339	983.3243	983.3216	983.3129	983.3193	983.3053	983.2951	983.2959	983.2946	983.2881	983.2691	983.2651	983.2634	983.2669
5	983.3125	983.3077	983.308	983.3094	983.3062	983.299	983.2887	983.2861	983.2705	983.2581	983.25	983.2441	983.2376	983.253
6	983.3126	983.2994	983.2999	983.2986	983.2935	983.2877	983.2829	983.2646	983.2503	983.2424	983.2319	983.2291	983.2323	983.2238
7	983.3043	983.2934	983.2885	983.2878	983.2826	983.2849	983.2666	983.2473	983.2359	983.2135	983.2175	983.2109	983.1991	983.215
8	983.2956	983.2914	983.2781	983.2823	983.2724	983.2632	983.2432	983.2366	983.2233	983.213[illegible]	983.2063	983.2013	983.1966	983.1903
9	983.2908	983.2779	983.2723	983.267	983.2547	983.2494	983.2341	983.2208	983.2082	983.19[illegible]4	983.1905	983.1888	983.1818	983.1763
10	983.258	983.2707	983.2632	983.2565	983.2485	983.2455	983.2206	983.207	983.1964	983.2[illegible]05	983.1808	983.1844	983.1723	983.176
11	983.2521	983.2455	983.2363	983.2552	983.2373	983.2188	983.2062	983.1984	983.18	983.[illegible]609	983.1664	983.1622	983.1622	983.1753
12	983.2419	983.232	983.2319	983.2273	983.2034	983.1893	983.172	983.173	983.1633	983.1527	983.1539	983.1605	983.1766	983.1805
13	983.2155	983.2166	983.2127	983.207	983.1923	983.1758	983.1643	983.1574	983.1614	983.1539	983.1582	983.1673	983.1784	983.1715
14	983.2074	983.2105	983.1917	983.1835	983.1634	983.1603	983.1535	983.1533	983.1605	983.1616	983.1576	983.1696	983.1602	983.1663
15	983.178	983.1814	983.1648	983.1577	983.1692	983.1622	983.153	983.1552	983.1559	983.1465	983.1567	983.1573	983.1541	983.1419
16	983.156	983.1577	983.1484	983.147	983.1426	983.1406	983.1354	983.1405	983.137[illegible]	983.1394	983.1313	983.1348	983.1234	983.1285
17	983.1318	983.128	983.1311	983.1236	983.1341	983.1364	983.1356	983.1499	983.1[illegible]1	983.1355	983.1324	983.1298	983.1213	983.1083
18	983.1139	983.0936	983.0998	983.1045	983.1097	983.1159	983.1256	983.1263	983.1[illegible]66	983.1073	983.1112	983.1159	983.1061	983.1124
19	983.0876	983.0741	983.075	983.0822	983.08	983.1007	983.1066	983.1069	983.1038	983.0959	983.0986	983.0979	983.0921	983.0901
20	983.0509	983.0523	983.0635	983.0704	983.0642	983.0853	983.1021	983.1005	983.0913	983.0726	983.0712	983.0645	983.0842	983.0795
21	983.036	983.0322	983.0309	983.0497	983.053	983.0831	983.0764	983.067	983.0596	983.0477	983.0448	983.0452	983.0553	983.0709
22	983.0019	983.008	983.0197	983.0344	983.0492	983.0598	983.0617	983.0516	983.026	983.052	983.042	983.0405	983.0542	983.0644
23	983.0015	982.9941	983.0112	983.0258	983.0363	983.032	983.045	983.0448	983.0405	983.0402	983.043	983.0408	983.0491	983.0401
24	982.974	982.9824	983.0013	983.023	983.0329	983.0461	983.0403	983.0549	983.0516	983.035	983.0362	983.033	983.0247	983.0237
25	982.9645	982.9855	983.0118	983.0217	983.0352	983.0441	983.0388	983.0[illegible]31	983.0459	983.0325	983.0159	983.0101	983.0154	982.9993
26	982.9702	982.9834	983.0054	983.0209	983.0316	983.0416	983.0395	983.[illegible]362	983.0258	983.0191	983.0077	983.008	983.0088	982.9975
27	982.9635	982.9921	983.0082	983.0252	983.0296	983.023	983.0256	983.0264	983.0169	983.0072	982.9973	982.9959	982.99	982.9877
28	982.9691	982.9794	983.0099	983.0206	983.0237	983.0191	983.0129	983.0091	982.9998	983.0023	982.9863	982.9772	982.9799	982.972
29	982.9709	982.9906	982.9913	983.0098	983.0103	983.0084	982.9969	982.9915	982.9795	982.9696	982.9681	982.9688	982.9606	982.9514
30	982.9784	982.987	982.9991	983	983.0031	982.9867	982.9805	982.9716	982.9711	982.955	982.938	982.945	982.9509	982.9468
31	982.9563	982.9747	982.9846	982.9902	982.9801	982.9698	982.96[illegible]	982.9569	982.9533	982.9474	982.9296	982.9214	982.9298	982.906
32	982.9608	982.9675	982.9737	982.9725	982.9679	982.9662	982.95[illegible]9	982.925	982.945	982.9286	982.9163	982.908	982.9002	982.8939
33	982.9563	982.956	982.9626	982.9588	982.956	982.9459	982.9[illegible]31	982.9183	982.9052	982.9002	982.9028	982.8883	982.8856	982.8759
34	982.9398	982.949	982.9583	982.9411	982.9288	982.9176	982.9064	982.8878	982.8866	982.8896	982.8646	982.8802	982.8811	982.8614
35	982.9385	982.937	982.9332	982.9345	982.9148	982.9027	982.8788	982.8683	982.8745	982.8628	982.8552	982.8392	982.8434	982.8505
36	982.9227	982.9238	982.9292	982.8992	982.8794	982.8773	982.8748	982.8361	982.8479	982.8458	982.8495	982.8378	982.8312	982.8247

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		


```
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')
```

* * * Top of File * * *

Ano	Mês	Dia	Prec
1941	1	1	19.0000000
1941	1	2	9.19999981
1941	1	3	0.00000000
1941	1	4	0.00000000
1941	1	5	0.00000000
1941	1	6	0.00000000
1941	1	7	0.00000000
1941	1	8	11.5000000
1941	1	9	0.00000000
1941	1	10	5.69999981
1941	1	11	2.09999990
1941	1	12	4.90000010
1941	1	13	9.39999962
1941	1	14	0.400000006
1941	1	15	2.59999990
1941	1	16	5.50000000
1941	1	17	1.20000005
1941	1	18	0.100000001
1941	1	19	5.40000010
1941	1	20	11.3999996
1941	1	21	28.0000000
1941	1	22	25.6000004
1941	1	23	9.30000019
1941	1	24	17.5000000
1941	1	25	0.00000000
1941	1	26	6.40000010
1941	1	27	1.79999995
1941	1	28	4.40000010
1941	1	29	5.40000010
1941	1	30	0.800000012
1941	1	31	11.3000002
1941	2	1	3.29999995
1941	2	2	0.00000000
1941	2	3	13.1999998
1941	2	4	0.00000000
1941	2	5	0.00000000

Série temporal diária