Autovalores e autovetores com Python

Prof. Doherty Andrade

www.metodosnumericos.com.br

1 Autovalores, autovetores e diagonalização de matrizes

Autovalores e autovetores

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import scipy.linalg as la
```

Vamos definir agora, antes de apresentar exemplos, autovalor e autovetor de uma matriz. Consideremos uma matriz quadradra A de dimensão n.

Um vetor não-nulo $v \in R^n$ é dito ser um autovetor de A associado ao autovalor λ se satisfaz ao sistema

$$Av = \lambda v.$$

Ou equivalentemente, se o sistema

$$(A - \lambda I)v = 0,$$

onde I é matriz identidade de dimensão n.

Note que se v é autovetor associado ao autovalor λ, então todo múltiplo não nulo de v tamém autovetor assoaicado a λ. De fato, seja ukv com $k \neq 0$ real. Então, temos:

$$A(u) = A(kv) = kA(v) = k(\lambda v) = \lambda(kv) = \lambda u.$$

Note que o vetor nulo é sempre solução do sistema, mas estamos só interessados em soluções não-nulas. As soluções não-nulas ocorrem exatamente quando $\det(A - \lambda I) = 0$.

Se $\det(A - \lambda I) \neq 0$, a solução é unica, sendo portanto, a solução trivial.

Calculando $\det(A - \lambda I)$ obtemos um polinômio de grau n em λ, portanto, os autovalores de A são exatamente as raízes deste polinômio.

Ao polinômio $p(\lambda) = \det(A - \lambda I)$ chamamos de polinômio característico de A.

Vejamos alguns exemplos:

```
In [2]: A = np.array([[4,1,0],[0,2,1],[0,0,3]])
        print('A=',A)
```

```
A= [[4 1 0]
 [0 2 1]
 [0 0 3]]
```

A função **la.eig** retorna o par (autovalor, autovetor), nesta ordem. Note que os autovalores em geral são números complexos que no Python tem a extensão j. Já os autovetores virão em seguida na forma de vetores. Observe que ao primeiro autovalor corresponde á primeira coluna que é autovetor e assim por diante. Assim, neste exemplo, associado ao autovalor $\lambda = 4$ corresponde o autovetor da primeira coluna $v = (1, 0, 0)$.

Os autovetores são apresentados normalizados (unitários de norma 1).

O polinômio característico de A é:

$$p(x) = x^3 - 9\,x^2 + 26\,x - 24.$$

Note que as raízes são 2, 3 e 4.

```
In [3]: from sympy import Matrix
        from sympy.abc import x, y
        A = Matrix([[4,1,0],[0,2,1],[0,0,3]])
        p= A.charpoly(x).as_expr()
        p

Out[3]: x**3 - 9*x**2 + 26*x - 24

In [4]: A = np.array([[4,1,0],[0,2,1],[0,0,3]])
        autos = la.eig(A)
        print(autos[0])
        print(autos[1]) #autovetor deve ser lido em coluna

[4.+0.j 2.+0.j 3.+0.j]
[[ 1.         -0.4472136  -0.57735027]
 [ 0.          0.89442719  0.57735027]
 [ 0.          0.          0.57735027]]


In [5]: print(autos[0])

[4.+0.j 2.+0.j 3.+0.j]


In [6]: print(autos[1]) #atenção: colunas

[[ 1.         -0.4472136  -0.57735027]
 [ 0.          0.89442719  0.57735027]
 [ 0.          0.          0.57735027]]
```

Podemos obter os mesmos resultados de modo mais imediata.

```
In [7]: eigvals, eigvecs = la.eig(A)
        print('Os autovalores são:',eigvals)
        print('Os autovetores lido em coluna são:',eigvecs)
```

```
Os autovalores são: [4.+0.j 2.+0.j 3.+0.j]
Os autovetores lido em coluna são: [[ 1.         -0.4472136  -0.57735027]
 [ 0.          0.89442719  0.57735027]
 [ 0.          0.          0.57735027]]
```

Agora vamos verificar os resultados obtidos. Vamos verificar que se os λs e os vetores satisfazem $Av = \lambda v$.

```
In [8]: lambda0 = eigvals[0]
        print(lambda0)
```

```
(4+0j)
```

```
In [9]: v0 = eigvecs[:,0]
        print(v0)
```

```
[1. 0. 0.]
```

```
In [10]: A @ v0 # 4 vezes o autovetor (1,0,0)
```

```
Out[10]: array([4., 0., 0.])
```

```
In [11]: for i in range(len(A)):
             print(eigvals[i])
```

```
(4+0j)
(2+0j)
(3+0j)
```

```
In [12]: lambda1 = eigvals[1]
         print(lambda1)
         v1 = eigvecs[:,1]
         print(v1)
         A @ v1
```

```
(2+0j)
[-0.4472136   0.89442719  0.        ]
```

```
Out[12]: array([-0.89442719,  1.78885438,  0.        ])
```

Note que $Av_1 = 2v_1$.

```
In [13]: lambda2 = eigvals[2]
         print(lambda1)
         v2 = eigvecs[:,2]
         print(v2)
         A @ v2
```

```
(2+0j)
[-0.57735027  0.57735027  0.57735027]
```

```
Out[13]: array([-1.73205081,  1.73205081,  1.73205081])
```

Note que $Av_2 = 3v_2$.

Podemos extrair os autovetores (colunas) da matriz eigvecs do seguinte modo:

```
In [14]: eigvals, eigvecs = la.eig(A)
         B = eigvecs
         print(B)
         B[:,0], # returns the 1th columm
```

```
[[ 1.         -0.4472136  -0.57735027]
 [ 0.          0.89442719  0.57735027]
 [ 0.          0.          0.57735027]]
```

```
Out[14]: (array([1., 0., 0.]),)
```

```
In [15]: B[:,1], # returns the 2th columm
```

```
Out[15]: (array([-0.4472136 ,  0.89442719,  0.        ]),)
```

```
In [16]: B[:,2] # returns the third columm
```

```
Out[16]: array([-0.57735027,  0.57735027,  0.57735027])
```

Vamos verificar todos os produtos $A(v_k)$.

```
In [17]: for k in range(len(A)):
             vk = B[:,k]
             uk = A @ vk
             print(uk)
```

```
[4. 0. 0.]
[-0.89442719  1.78885438  0.        ]
[-1.73205081  1.73205081  1.73205081]
```

2 Diagonalização

Uma matriz quadrada é diagonalizável se for semelhante a uma matriz diagonal. Em outras palavras, é diagonalizável se existir uma matriz invertível P de tal modo que $D = P^{-1}AP$ é uma matriz diagonal.

Portanto, $A = PDP^{-1}$.

Um teorema importante da álgebra linear é que uma matriz quadrada do dimensão n é diagonalizável se e somente se tem n autovetores linearemente independentes. Além disso, P é a matriz que tem suas colunas como autovetores de A e D.

Como exemplo para este importante teorema vamos construir uma matriz A que tenha autovalores $\lambda_0 = 1, \lambda_1 = 2, \lambda_2 = 3$ e autovetores iguais a $v_0 = (1, 1, 0), v_1 = (0, 1, 2, v_2 = (1, 0, 3)$.

De acordo com o teorema acima basta calcular $A = PDP^{-1}$, onde

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

e

$$P = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 2 & 3 \end{bmatrix}.$$

Assim, temos:

```
In [18]: D = np.array([[1,0,0],[0,2,0],[0,0,3]])
         print(D)
```

```
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

```
In [19]: P = np.transpose(np.array([[1,1,0],[0,1,2],[1,0,3]]))
         print(P)
```

```
[[1 0 1]
 [1 1 0]
 [0 2 3]]
```

```
In [20]: M = P @ D @ la.inv(P)
         M
```

```
Out[20]: array([[ 1.8, -0.8,  0.4],
                [-0.6,  1.6,  0.2],
                [ 1.2, -1.2,  2.6]])
```

Verificando... note que os auvetores estão em colunas e normalizados. True

```
In [21]: evals, evecs = la.eig(M)
         print(evals)
         print(evecs)
```

```
[3.+0.j 1.+0.j 2.+0.j]
[[-3.16227766e-01  7.07106781e-01 -2.84766197e-16]
 [ 5.61824562e-17  7.07106781e-01  4.47213595e-01]
 [-9.48683298e-01  5.95161352e-16  8.94427191e-01]]
```

3 Matrizes simétricas

As matrizes simétricas desempenham um papel importante dentro da álgebra linear.

O teorema mais importante sobre as matrizes simétricas é:

Teorema: Matrizes simétricas possuem todos os seus autovalores reais e autovetores associados a autovalores distintos são ortogonais.

Vejamos alguns exemplos. Antes vamos criar uma matriz simétrica.

```
In [22]: n = 4
         M = np.random.randint(0,10,(n,n))
         print('M=' ,M)
         A = M @ np.transpose(M)
         print('A=', A)
```

```
M= [[4 0 4 6]
 [7 1 4 2]
 [4 9 9 4]
 [6 3 7 2]]
A= [[ 68  56  76  64]
 [ 56  70  81  77]
 [ 76  81 194 122]
 [ 64  77 122  98]]
```

Note que os autovalores são reais.

```
In [23]: evals, evecs = la.eig(A)
         print(evals)
```

```
[367.81040794+0.j  44.42842783+0.j  16.36019213+0.j   1.4009721 +0.j]
```

```
In [24]: P =  evecs
         print(P)
```

```
[[ 0.35373883  0.52544349 -0.77004256  0.07623928]
 [ 0.38399215  0.51298862  0.4660353  -0.61008508]
 [ 0.68770925 -0.66310774 -0.16109057 -0.247778  ]
 [ 0.504455    0.14505151  0.40483994  0.74872547]]
```

Agora vamos checar que os autovetores são ortogonais entre si.

```
In [25]: v0 = evecs[:,0] # primeira coluna é o primeiro autovetor
         v1 = evecs[:,1]
         v2 = evecs[:,2]
         v3 = evecs[:,3]
```

Realize todos os produtos como indicado abaixo

```
In [26]: v0 @ v1
```

```
Out[26]: -5.551115123125783e-16
```

```
In [27]: v2 @ v3
```

```
Out[27]: 1.887379141862766e-15
```

4 Potência de matriz

Se A é uma matriz quadrada definimos A^K como sendo:

$$A^0 = I$$

$$A^k = \underbrace{A \cdot A \cdot \ldots \cdot A}_{=k}.$$

Dependendo do tamanho da matriz, a potência pode ser difícil de ser calculada. Mas de A for simétrica, portanto diagonalizável, a potência pode ser mais facilmente calculada. Vamos exploar isto.

Suponha que A seja diagonalizável. Então, existem matriz diagonal D e matriz P tais que $A = PDP^{-1}$. Logo, temos que:

$$A^k = (PDP^{-1})^k = \underbrace{PDP^{-1} \cdot PDP^{-1} \cdot \ldots \cdot PDP^{-1}}_{=k} = PD^kP^{-1}.$$

Vamos ver um exemplo. Vamos tomar a matriz simétria A acima e calcular A^{10}.

```
In [28]: n= 10
         print(A)
```

```
[[ 68  56  76  64]
 [ 56  70  81  77]
 [ 76  81 194 122]
 [ 64  77 122  98]]
```

```
In [29]: D = np.diag(evals)
         D
```

```
Out[29]: array([[367.81040794+0.j,   0.        +0.j,   0.        +0.j,
                   0.        +0.j],
                [  0.        +0.j,  44.42842783+0.j,   0.        +0.j,
                   0.        +0.j],
                [  0.        +0.j,   0.        +0.j,  16.36019213+0.j,
                   0.        +0.j],
                [  0.        +0.j,   0.        +0.j,   0.        +0.j,
                   1.4009721 +0.j]])
```

```
In [30]: P
```

```
Out[30]: array([[ 0.35373883,  0.52544349, -0.77004256,  0.07623928],
                [ 0.38399215,  0.51298862,  0.4660353 , -0.61008508],
                [ 0.68770925, -0.66310774, -0.16109057, -0.247778  ],
                [ 0.504455  ,  0.14505151,  0.40483994,  0.74872547]])
```

```
In [31]: %%timeit
         P @ D**n @ la.inv(P)
```

```
123 µs ± 2.99 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
In [32]: P @ D**n @ la.inv(P)
```

```
Out[32]: array([[5.67029458e+24+0.j, 6.15524339e+24+0.j, 1.10237092e+25+0.j,
                 8.08621556e+24+0.j],
                [6.15524339e+24+0.j, 6.68166718e+24+0.j, 1.19665059e+25+0.j,
                 8.77778469e+24+0.j],
                [1.10237092e+25+0.j, 1.19665059e+25+0.j, 2.14313673e+25+0.j,
                 1.57205394e+25+0.j],
                [8.08621556e+24+0.j, 8.77778469e+24+0.j, 1.57205394e+25+0.j,
                 1.15314789e+25+0.j]])
```

Resultado muito mais rápido.