Calcul Scientifique

Cours 4: Le broadcast

Alexis Lechervy

université de Caen
Basse-Normandie

Sommaire

Le broadcast

Principe

Le broadcast est un mécanisme permettant de traiter deux tableaux numpy de taille différent comme des tableaux de même taille. Les axes de dimension 1 sont complétés virtuellement, par recopie, pour atteindre la même taille que ceux de l'autre tableau. L'intérêt du broadcast est de manipuler ces tableaux comme des tableaux ayant la même taille sans pour autant les calculer explicitement.

Exemples :
(3,1,4,5,1) (1,7,4,1,9) -> (3,7,4,5,9)
(3,1,4,5,1) (1,7,6,1,9) -> impossible, dimension non compatible

Broadcast explicite

Il est possible de faire un broadcast explicite à l'aide des fonctions *np.broadcast* (produit un itérateur) ou *np.broadcast_arrays* (produisant les tableaux).

```
>>> x.shape
(3,1,4,5,1)
>>> y.shape
(1,7,4,1,9)
>>> xx,yy = np.broadcast_array(x, y)
>>> print(xx.shape,yy.shape)
(3,7,4,5,9) (3,7,4,5,9)
```

Broadcast implicite

Broadcast implicite

Les opérations arithmétiques et logique peuvent faire appel implicitement au broadcast. C'est l'usage essentiel du broadcast sous numpy.

Règles générales

Lors d'une opération entre deux tableau, numpy commence par comparer les dimensions des deux tableaux dimensions par dimensions. L'opération sera possible si :

1. toute les dimensions sont identiques,
2. les dimensions sont identiques ou un des tableaux à une dimension à 1.

Dans le deuxième cas, le tableau le plus petit sera recopié suivant la dimension à 1 jusqu'à avoir la même taille que l'autre.

Opérations arithmétiques

+ , - , * , / , // , ** , % ,

Opérations logiques

== , != , < , > , <= , >= , np.logical_and , np.logical_or , np.logical_xor

Le broadcast entre un tableau et une valeur

Principe

Une opération arithmétique ou logique entre un tableau M et un valeur a est équivalente à la même opération entre chaque valeurs $M_{i,j}$ du tableau et la valeur a.

Exemple simple : Opération entre un vecteur et un scalaire

v > a est équivalent à v>a*np.ones(v.shape)

$$\begin{bmatrix} 1 & 9 \end{bmatrix} > 5 \longrightarrow \begin{bmatrix} 1 & 9 \end{bmatrix} > \begin{bmatrix} 5 & 5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 > 5 & 9 > 5 \end{bmatrix} \longrightarrow \begin{bmatrix} \mathit{False} & \mathit{True} \end{bmatrix}$$

Exemple simple : Opération entre une matrice et un scalaire

M+s est équivalent à M+s*np.ones(M.shape)

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + 5 \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix} \longrightarrow \begin{bmatrix} 1+5 & 2+5 \\ 3+5 & 4+5 \end{bmatrix} \longrightarrow \begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix}$$

Opération entre une matrice (n,m) et un vecteur (1,m)

Principe

Soit M un array de taille (n, m) et v un array de taille $(1, m)$. L'opération numpy $M + v$ correspond à construire une matrice de taille (n, m) où le vecteur v est recopié m fois puis à la sommer avec M.

Exemple 1 : M+v

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 11 & 22 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 11 & 22 \\ 11 & 22 \end{bmatrix}$$

Exemple 2 : M+v

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 11 & 22 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 11 & 22 \\ 11 & 22 \\ 11 & 22 \end{bmatrix}$$

Exemple 3 : M+v

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 11 & 22 & 33 \end{bmatrix} \longrightarrow \text{Erreur}$$

Opération entre une matrice (n,m) et un vecteur (m,)

Principe

Cette opération est équivalente à la précédente.

Rappel

Le parcoure des tableaux se fait en mémoire de la dernière dimension vers la premier dimension. Les données des tableaux de taille : (m,) (1,m) (1,1,m) (1,1,1,m) ... sont donc stocké dans le même ordre. Par conséquence, il n'est pas nécessaire d'utiliser *np.newaxis* et un tableau de dimension (m,) sera traité ici comme un tableau de dimension (1,m).

Exemple 1 : M+v

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 11 & 22 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 11 & 22 \\ 11 & 22 \end{bmatrix}$$

Exemple 2 : M+v

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 11 & 22 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 11 & 22 \\ 11 & 22 \\ 11 & 22 \end{bmatrix}$$

Opération entre une matrice (n,m) et un vecteur (n,1)

Principe

Soit M un array de taille (n, m) et v un array de taille $(n, 1)$. L'opération numpy $M + v$ correspond à construire une matrice de taille (n, m) où le vecteur v est recopié m fois à la sommer avec M. Le vecteur est recopié sur les colonne non sur les lignes.

Exemple 1 : M+v[:,np.newaxis]

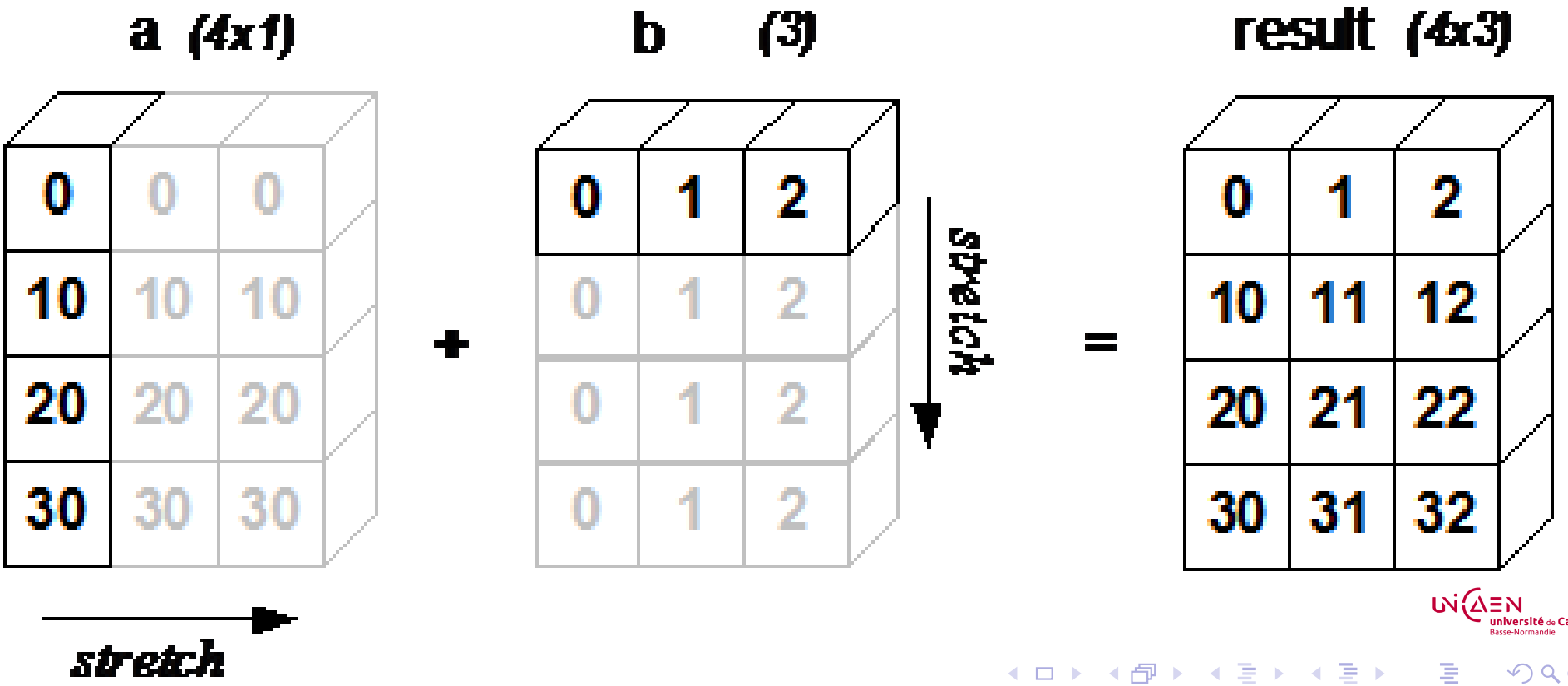
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 5 \\ 6 & 6 \end{bmatrix}$$

Exemple 2 : M+v[:,np.newaxis]

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 7 \\ 8 & 8 \\ 9 & 9 \end{bmatrix}$$

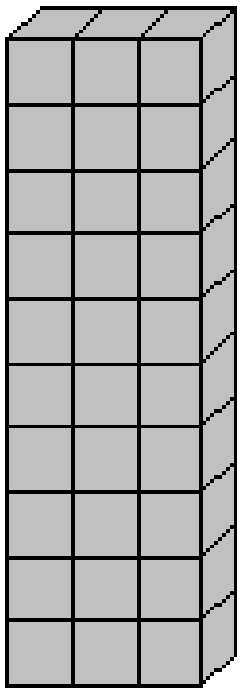
Broadcast entre un tableau ligne et un tableau colonne

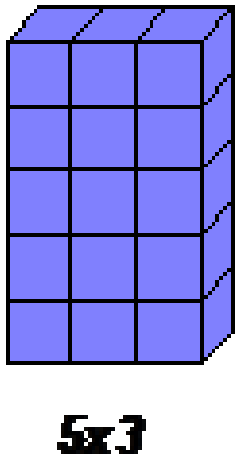
```
>>> a = np.array([0.0,10.0,20.0,30.0])
>>> b = np.array([1.0,2.0,3.0])
>>> a[:,newaxis] + b
```

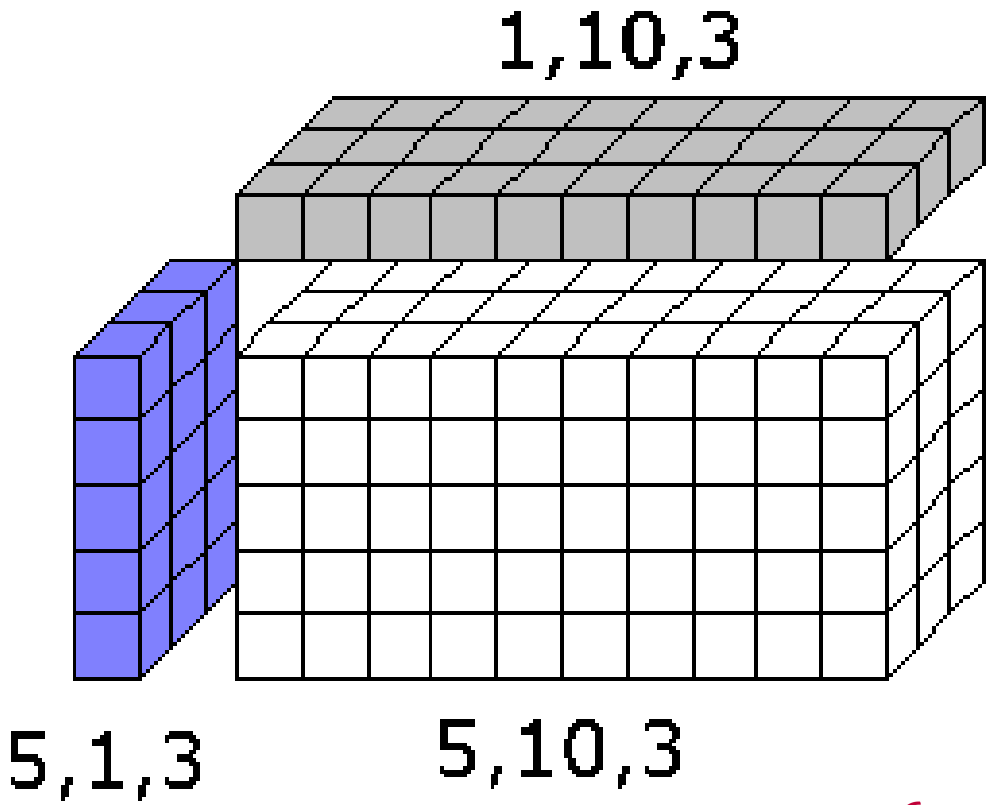


Broadcast avec des matrices de tailles différentes

```
>>> a.shape
(10,3)
>>> b .shape
(5,3)
>>> c = a[newaxis,:,:] * b[:,newaxis,:]
>>> c .shape
(5,10,3)
```







Sommaire

Table de multiplications

Objectif

Créer une table des multiplications enter 0 et 10.

```
>>> x = np.arange(11)
>>> x[:,np.newaxis]*x
array([[  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [  0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20],
       [  0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30],
       [  0,  4,  8, 12, 16, 20, 24, 28, 32, 36, 40],
       [  0,  5, 10, 15, 20, 25, 30, 35, 40, 45, 50],
       [  0,  6, 12, 18, 24, 30, 36, 42, 48, 54, 60],
       [  0,  7, 14, 21, 28, 35, 42, 49, 56, 63, 70],
       [  0,  8, 16, 24, 32, 40, 48, 56, 64, 72, 80],
       [  0,  9, 18, 27, 36, 45, 54, 63, 72, 81, 90],
       [  0, 10, 20, 30, 40, 50, 60, 70, 80, 90,100]])
```

Tester les valeurs d'un tableau

ET

Tester est-ce toute les valeurs du tableau sont supérieurs à 4 ($M_0 > 4$ et $M_1 > 4$ et $M_2 > 4$...) :

np.prod(M>4)==1 ou np.all(M>4)

Ou

Tester est-ce une des valeurs est supérieur à 3 ($M_0 > 3$ ou $M_1 > 3$ ou $M_2 > 3$...) :

np.sum(M>3) !=0 ou np.any(M>3)

Exemple : Pourcentage de glucide/lipide/protéine

Objectif

Calculer le pourcentage de glucide, Lipide et Protéine d'aliment connaissant la quantité pour 100g de glucide, Lipide et Protéine de ces aliments.

	Pomme	Oeuf	Lait	Boeuf
Glucide	14	0.7	5	0
Lipide	0.2	11	1	15
Protéine	0.3	13	3.4	26

Calcul de la quantité de glucide+lipide+protéine pour chaque aliment

s = np.sum(M,axis=0) –> [14.5 24.7 9.4 51.]

Pourcentage de glucide+lipide+protéine pour chaque aliment

print(M.shape) –> (3,4)
print(s.shape) –> (4,)
100*M/s

	Pomme	Oeuf	Lait	Boeuf
Glucide	96.55	2.83	53.19	0.
Lipide	1.38	44.53	10.64	29.41
Protéine	2.07	52.63	36.17	70.59

Exemple : Ajouter un vignettage à une photo

Objectif

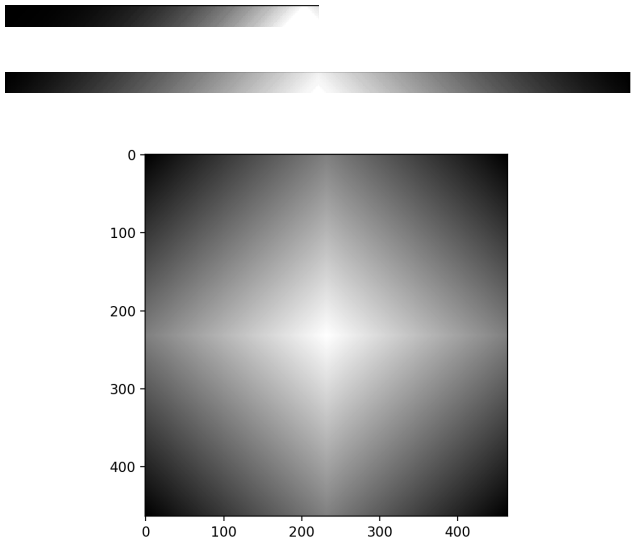
Ajouter du vignettage à une image. Cela consiste à assombrie les bords d'une photo.

Construire un filtre

v = np.arange(im.shape[0]//2)/(im.shape[0]//2)

line = np.concatenate((v,1-v))

masque = (line[:,np.newaxis]+line[np.newaxis, :])/2



Exemple : Ajouter un vignettage à une photo

On applique le masque

```
imFinal = masque[ :, :,np.newaxis]*im
imFinal = imFinal.astype('ubyte')
```

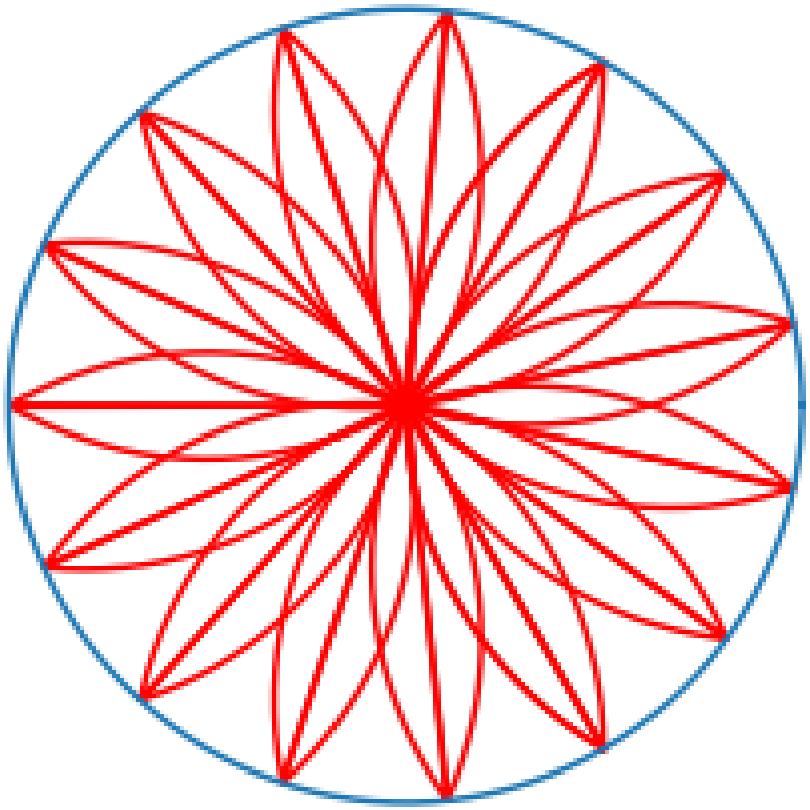
Avant :

Après

Rosace (0)

Objectif

Tracer une rosace comme :



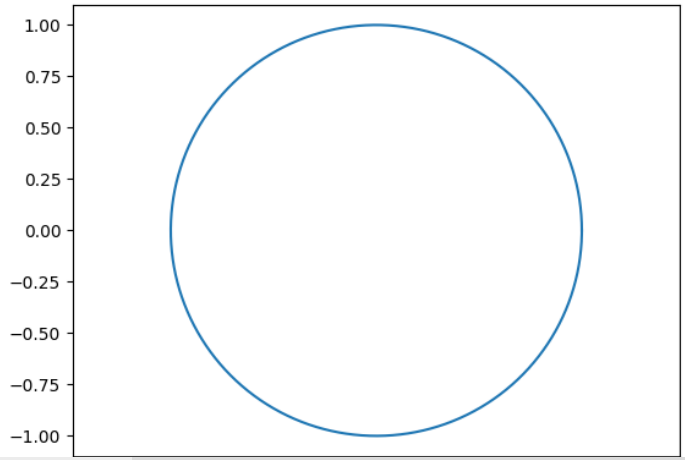
Rosace (1)

Tracer un cercle de centre (0,0) et de rayon 1

```
x = np.linspace(0,2*np.pi,1000)
x1 = np.cos(x)
x2 = np.sin(x)
P = np.stack((x1,x2),axis=1)
plt.plot(P[:,0],P[:,1])
plt.axis('equal')
plt.show()
```

Autre solution avec broacast

```
x = np.linspace(0,2*np.pi,1000)
a = np.array([0,-np.pi/2]
P = np.cos(x[:,np.newaxis] + a)
plt.plot(P[:,0],P[:,1])
plt.axis('equal')
plt.show()
```

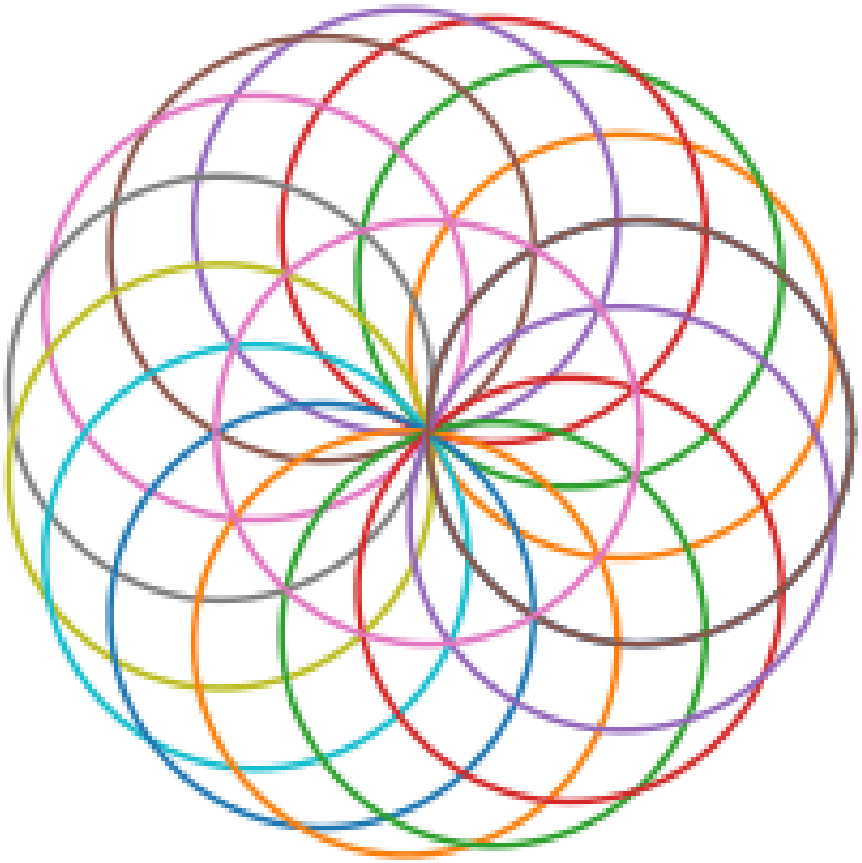


Rosace (2) :Tracer des cercle de centres différents

But

On souhaite tracer des cercles de rayon 1 dont les centres sont des points espacé régulièrement sur le cercle de centre (0,0) et de rayon 1.

Résultat



Rosace (2) :Tracer des cercle de centres différents

But

On souhaite tracer des cercles de rayon 1 dont les centres sont des points espacé régulièrement sur le cercle de centre (0,0) et de rayon 1.

```
#generation des centres de cercle
n = 16
xc = np.linspace(0,2*np.pi,n)
centre = np.cos(xc[:,np.newaxis]+np.array([0,-np.pi/2]))

#generation des cercles
x = np.linspace(0,2*np.pi,1000)
p = np.cos(x[:,np.newaxis]+np.array([0,-np.pi/2]))
c = p[:,np.newaxis,:] + centre[np.newaxis,:,:]

# cercle principal
c1 = np.cos(x[:,np.newaxis]+np.array([0,-np.pi/2]))

plt.plot(c[:,:,0],c[:,:,1])
plt.plot(c1[:,0],c1[:,1])
plt.axis('equal')
plt.axis('off')
plt.show()
```

Rosace (3) : Sélection des parties dans le cercle central

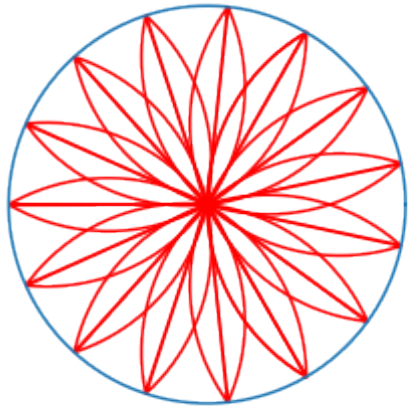
But

On souhaite ne garde dans le dessin précédent uniquement les tracés à l'intérieur du cercle centrale.

Code

```
r = c * (np.sum(c**2,axis=2)<1)[:,:,np.newaxis]
plt.plot(r[:,:,0],r[:,:,1],c='r')
plt.plot(c1[:,0],c1[:,1])
plt.axis('equal')
plt.axis('off')
plt.show()
```

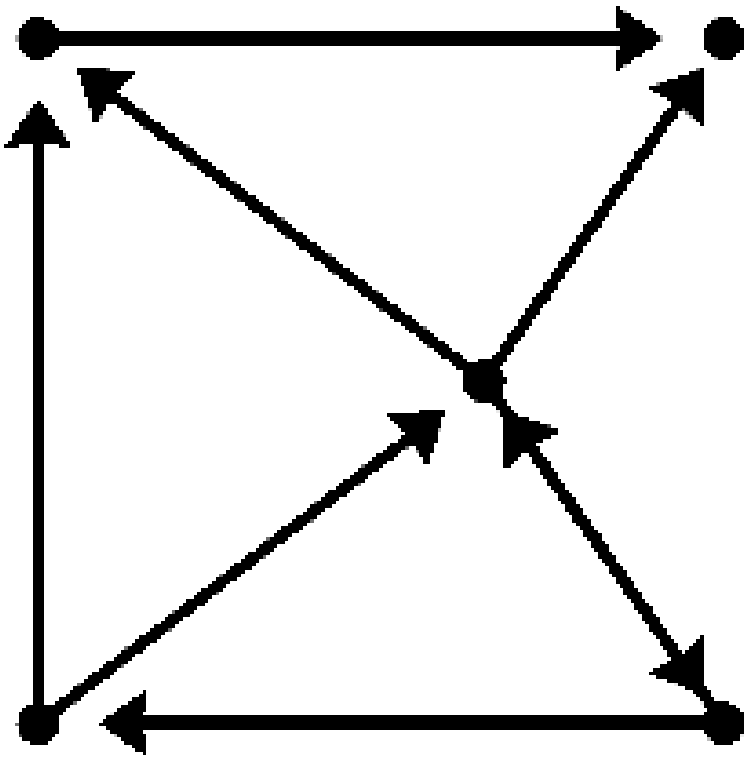
Résultat

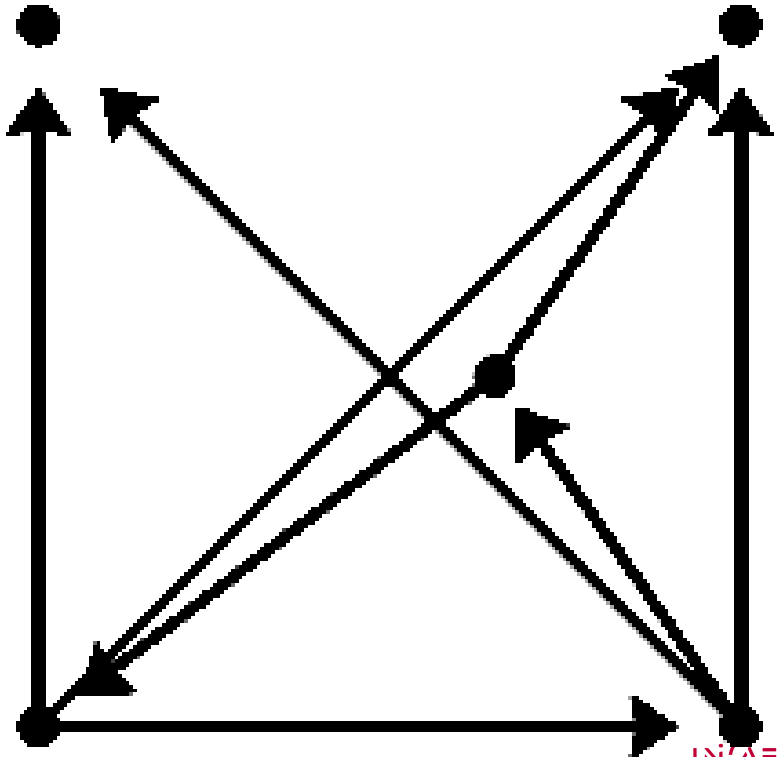


Noeud à distance 2 sur un graphe

But

Soit un graphe g donné, on veut connaître les nœuds que l'on peut atteindre après deux mouvements sur le graphe.





Noeud à distance 2 sur un graphe

Code

```
g = np.array([[0,1,0,0,1],
              [0,0,0,1,0],
              [1,0,0,0,1],
              [0,0,0,0,0],
              [0,1,1,1,0]])
g2 = np.sum(g[:,:,np.newaxis]*g, axis=1)>0
```

Rappel : g est équivalent à g[np.newaxis, :, :]

Explications

- g[:, :,np.newaxis] (en bleu) nous indique les premiers mouvements possibles.
- g (en gris) indique les seconds mouvements possibles.
- Le produit indique les chemins possibles. Un 1 en i,j,k indique que l'on peut passer de i à k en passant par j.
- La somme permet de récupérer le nombre de chemin allant de i à k.