L2-S3 : UE Simulations numériques

SEANCE 5

Traitement de signaux, Traitement d'images

7 octobre 2024

Introduction

Le traitement d'images est largement utilisé dans plusieurs domaines :

- En physique :
  - analyse des données collectées par instruments
  - observation terrestre, astrophysique, écoulements de fluides, ...
- Médecine (IRM, ...)
- Photographie, Graphisme (Photoshop pour n'en citer qu'un)
- Surveillance :
  - images satellites
  - détection automatique de plaques d'immatriculation
- Dans la vie courante : votre téléphone

Il s'agit de minimiser certains parasites et artefacts instrumentaux ou de faire ressortir certaines caractéristiques de ce qu'on observe, par exemple :

- filtrage pour éliminer les parasites aléatoires et décorrélés ("bruit") $\rightarrow$ lissage, filtre passe-bas
- traitement d'image (filtres de type emboss, détection de contraste, etc...)

Convolution

On va transformer un *signal initial* en un *signal filtré* par une opération de convolution qui fait intervenir une fonction *Filtre* :

$$S_{filtr}(x) = (F \otimes S_{initial})(x) = \int_{-\infty}^{+\infty} S_{initial}(u) F(x-u) \mathrm{d}u$$

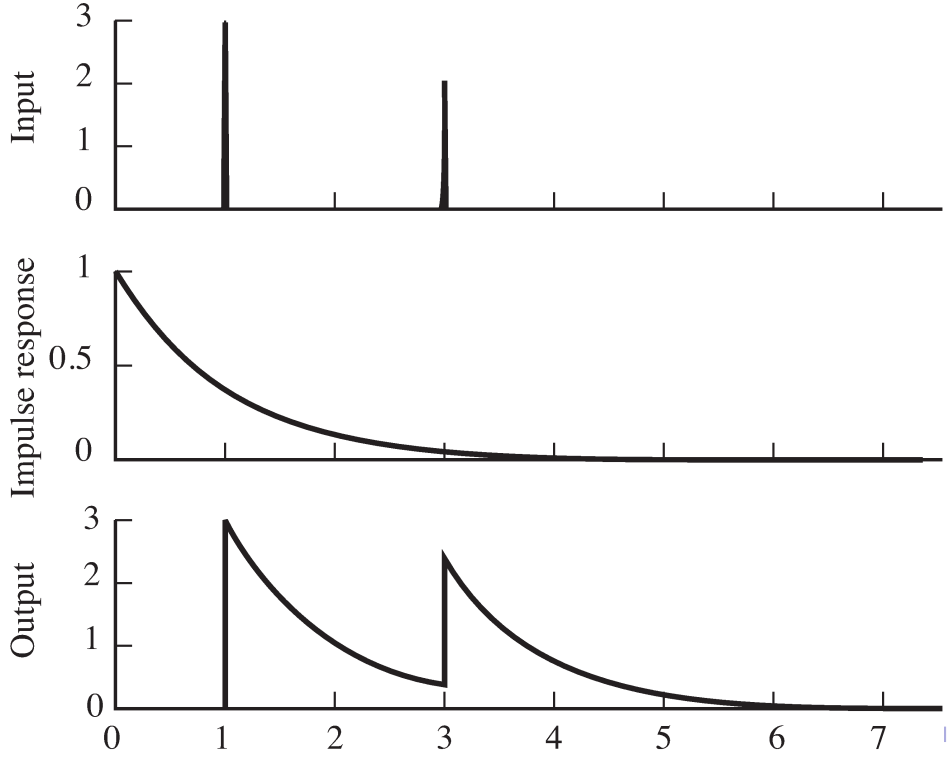
Fonctionne aussi avec un signal dependant de deux variables (traitement d'images)

Convolution

Le signal filtré obtenu lorsque le signal initial est une impulsion de Dirac représente une caractéristique importante d'une fonction Filtre :
Pour $S_{initial}(x) = \delta(x - x_0)$, on obtient :

$$S_{filtr}(x) = \int_{-\infty}^{+\infty} \delta(u - x_0) F(x - u) \mathrm{d}u = F(x - x_0)$$

Réponse impulsionnelle d'un filtre $F(t) = e^{-t/\tau}$



Propriétés de la convolution

Commutativité :

$$(g \otimes f)(x) = \int_{-\infty}^{+\infty} g(u) f(x-u) \mathrm{d}u = \int_{-\infty}^{+\infty} g(x-u') f(u') \mathrm{d}u' = (f \otimes g)(x)$$

Distributivité :

$$f \otimes (g + h) = f \otimes g + f \otimes h$$

Associativité :

$$f \otimes (g \otimes h) = (f \otimes g) \otimes h$$

Utilisation dans Python

Nous travaillerons d'abord sur des convolutions à une dimension.
Les modules `numpy` et `scipy` offrent les convolutions linéaires discrètes :

- `np.convolve(a, v, mode='full')`
  - `a` : vecteur avec N éléments
  - `v` : vecteur avec M éléments
  - `mode` : `'full'` retourne un vecteur N+M-1 éléments ;
    `'same'` retourne un vecteur max(M,N) éléments ;
    `'valid'` retourne un vecteur (max(M, N) - min(M, N) + 1) éléments.
- méthode équivalente dans `scipy` :
  `scipy.signal.convolve(a,v,mode='full',method='auto')`

$$(a \otimes v)[n] = \sum_{m=-\infty}^{m=+\infty} a[m] \times v[n-m]$$

ATTENTION : on suppose que les deux fonctions f et g représentées par les vecteurs `a` et `v` ont le même pas d'échantillonnage sur x.

Utilisation dans Python

Vérifions la formule ci-dessus, et constatons que lorsque l'indice du tableau a ou v n'existe pas, alors on utilise la valeur 0 :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  a=[1,2,3,4,5,6]
5  v=[10,20,30]
6  print("same :",np.convolve(a,v,"same"))
7  print("full :",np.convolve(a,v,"full"))
8  print("valid :",np.convolve(a,v,"valid"))
9  print("n=0 : a[0]*v[0]=",a[0]*v[0])
10 print("n=1 : a[0]*v[1]+a[1]*v[0]=",a[0]*v[1]+a[1]*v[0])
11 print("n=2 : a[0]*v[2]+a[1]*v[1]+a[2]*v[0]=",a[0]*v[2]+a[1]*v[1]+a
       [2]*v[0])
12 print("n=3 : a[1]*v[2]+a[2]*v[1]+a[3]*v[0]=",a[1]*v[2]+a[2]*v[1]+a
       [3]*v[0])
13 print("n=4 : a[2]*v[2]+a[3]*v[1]+a[4]*v[0]=",a[2]*v[2]+a[3]*v[1]+a
       [4]*v[0])
14 print("n=5 : a[3]*v[2]+a[4]*v[1]+a[5]*v[0]=",a[3]*v[2]+a[4]*v[1]+a
       [5]*v[0])
15 print("n=6 : a[4]*v[2]+a[5]*v[1]=",a[4]*v[2]+a[5]*v[1])
16 print("n=7 : a[5]*v[2]=",a[5]*v[2])
```

Utilisation dans Python

Out[0]:

```
same  : [ 40  100  160  220  280  270]
full  : [ 10   40  100  160  220  280  270  180]
valid  : [100  160  220  280]
n=0  :  a[0]*v[0]= 10
n=1  :  a[0]*v[1]+a[1]*v[0]= 40
n=2  :  a[0]*v[2]+a[1]*v[1]+a[2]*v[0]= 100
n=3  :  a[1]*v[2]+a[2]*v[1]+a[3]*v[0]= 160
n=4  :  a[2]*v[2]+a[3]*v[1]+a[4]*v[0]= 220
n=5  :  a[3]*v[2]+a[4]*v[1]+a[5]*v[0]= 280
n=6  :  a[4]*v[2]+a[5]*v[1]= 270
n=7  :  a[5]*v[2]= 180
```

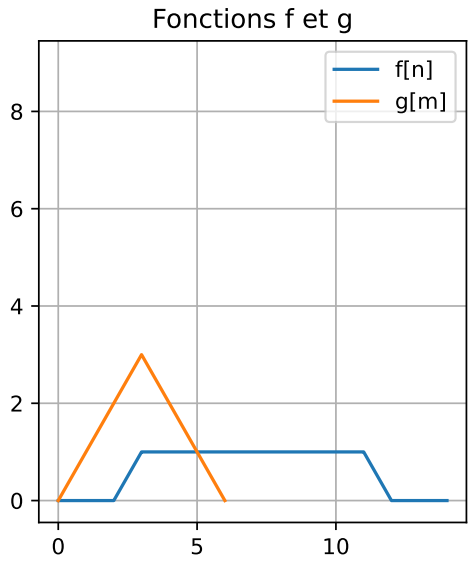
On constate que le calcul manuel redonne bien le résultat de `np.convolve` en mode `full` à condition de considérer les valeurs des tableaux nulles en dehors de leurs définitions.

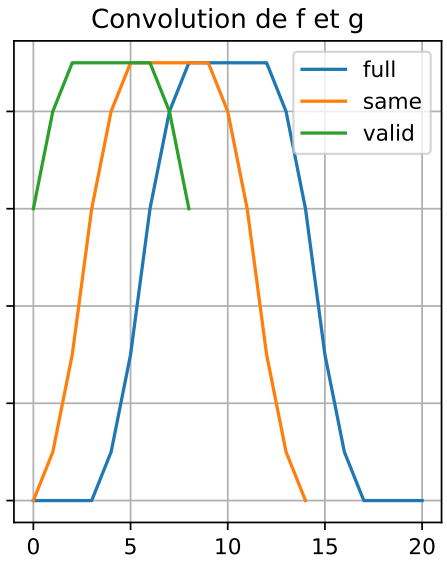
Utilisation dans Python

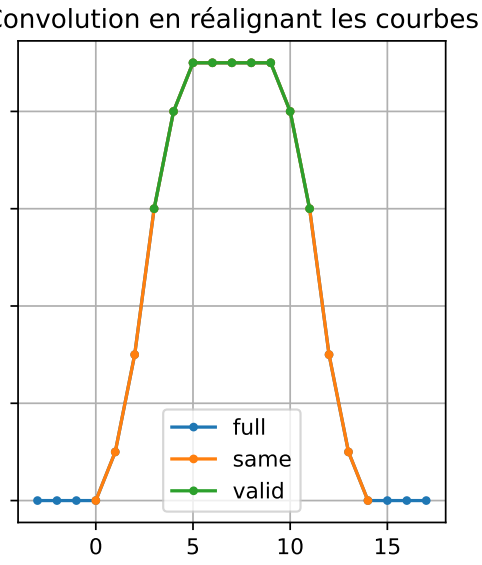
Convolution d'une fonction porte par une fonction triangle plus fine.

```
f=np.array([0,0,0,1,1,1,1,1,1,1,1,1,0,0,0])
g=np.array([0,1,2,3,2,1,0])
# convolution, avec les trois options possibles
full = np.convolve(f,g)
same = np.convolve(f,g,'same')
valid = np.convolve(f,g,'valid')
# superposition correcte des trois options
Max= max(f.size,g.size)
Min= min(f.size,g.size)
size_full=f.size+g.size-1
x_full=np.arange(-int((Min-1)/2),-int((Min-1)/2)+size_full)
x_same=np.arange(f.size)
size_valid=Max-Min+1
x_valid=np.arange(int(Min/2),int(Min/2)+size_valid)

fig, axs = plt.subplots(1, 3, figsize=(12,4),  sharey=True)
axs[0].plot(f, label='f[n]')
axs[0].plot(g, label='g[m]')
axs[0].legend()
axs[0].grid()
axs[0].set_title("Fonctions f et g")
axs[1].plot(full, label="full")
axs[1].plot(same, label="same")
axs[1].plot(valid, label="valid")
axs[1].legend()
axs[1].grid()
axs[1].set_title("Convolution de f et g")
```

Utilisation dans Python







Utilisation dans Python

Bilan :

- `'valid'` : seuls les pixels où les 2 fonctions se recouvrent complètement sont gardées : pas d'effet de bord
- `'same'` : le tableau de sortie a le format du signal le plus long mais aux bords la convolution est un peu fausse
- `'full'` : le tableau de sortie a le format maximal où les deux signaux se recouvrent d'au moins 1 pixel mais des effets de bord sont à craindre

Qu'est ce qu'une image ?

Nous percevons les couleurs avec nos yeux !

- Spectre électromagnétique limité :
  - visible seulement ;
  - pour les images dans un autre domaine EM, on utilise une représentation en fausses couleurs pour nous les rendre visibles.
- Codage
  - notre rétine détecte seulement 3 couleurs : Rouge, Vert et Bleue ;
  - notre cerveau combine les trois couleurs primaires d'intensités différentes pour former toutes les autres couleurs ;
  - de ce fait, tous les appareils (TV, moniteurs, ...) suivent le même principe ;
  - les images sont aussi codées sur 3 couleurs (RGB).

Format d'images

Plusieurs types de format avec compression plus ou moins importante.

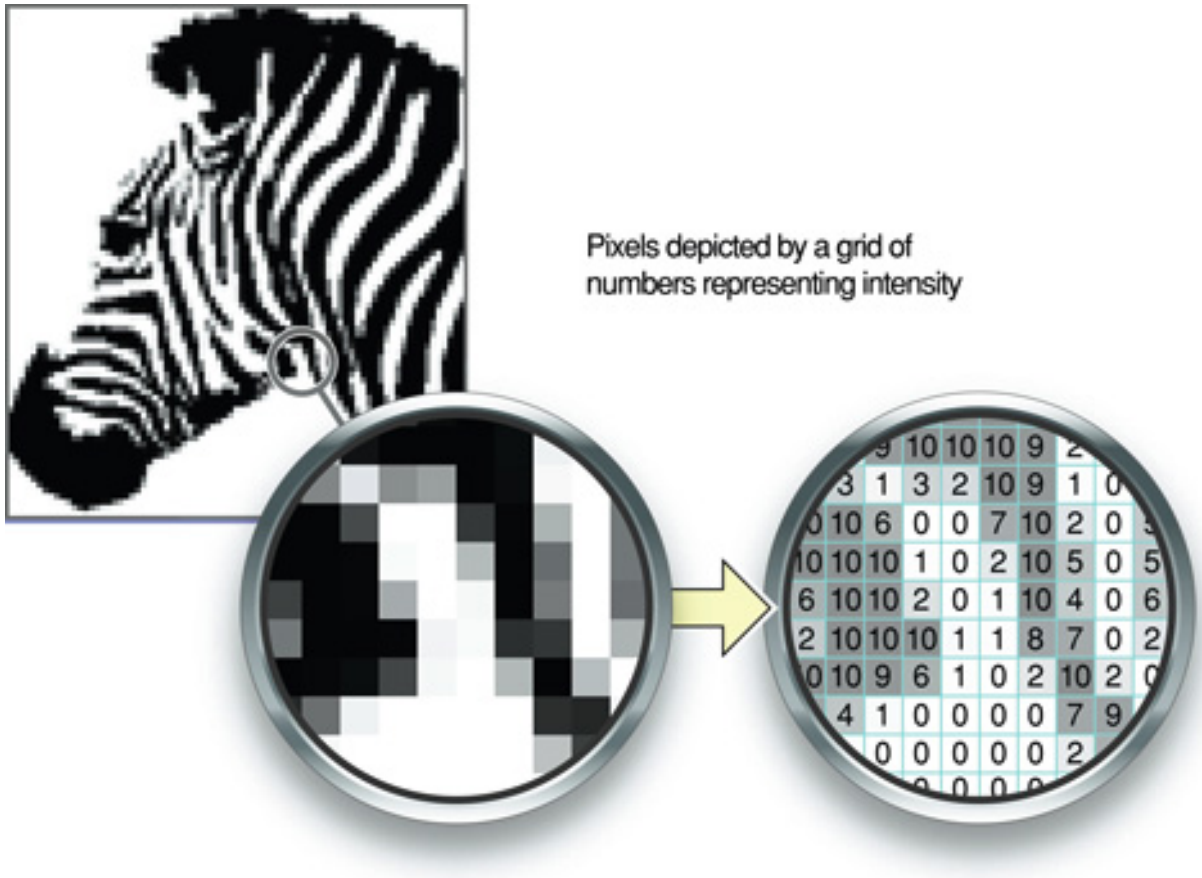
- bmp (BitMap) : le format original, sans compression de données, avec des images énormes.
- tiff (Tagged Image File Format) : sans compression, avec 16 ou 32 bits par pixel : images très volumineuses. Pour usages professionnels.
- gif (Graphics Interchange Format) : sans compression, avec seulement 8 bits par pixel (codage de 256 couleurs).
- png (Portable Network Graphic) : Compression d'images, sans perte de données. Les couleurs codées sur 24 bits (16,7 millions de couleurs). Permet de la compression en gardant une très bonne qualité d'image.
- jpeg / jpg (Joint Photographic Expert Group) : Compression d'images **avec perte de données**. Les images en noir et blanc ou couleur, avec 8, 16 ou 32 bits par pixel. Son grand avantage est de produire des fichiers de petite taille.

Lecture d'images et manipulation

Il existe beaucoup de logiciels pour lire et traiter les images, y compris pour Python. Nous allons voir essentiellement deux façons.

- Module `image` de `matplotlib` :
  - marche pour fichier .png seulement ;
  - importation facile des données dans numpy arrays.
- `PIL` (Python Imaging Library) :
  - marche pour plusieurs types de format ;
  - inclut aussi plusieurs modules de traitement d'images.
- A noter aussi pour travailler sur les images : l'existence de la librairie scikit-image : `https://scikit-image.org/`
- A vous de trouver vos préférences.

Lecture d'images et manipulation



Utilisation de matplotlib

```
1 import matplotlib.image as mpimg
2
3 img = mpimg.imread("data/saturne.png")
```

- Format de la matrice `numpy` : `img[npx,npy,3]`
  - `npx` et `npy` nombre de pixels sur axes x et y
  - 3 fait référence aux 3 composantes Red, Green, Blue
- Attention aux valeurs d'intensité de pixels ($0 \rightarrow 1$ ou $0 \rightarrow 255$)

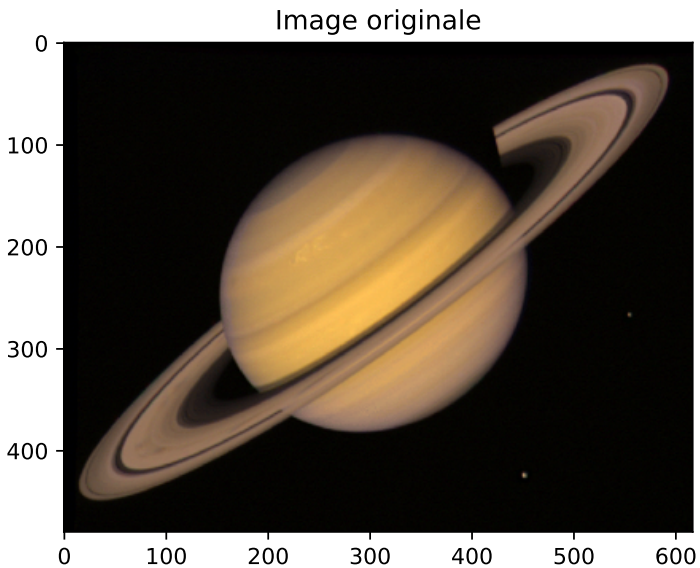
Après la mise en forme dans un tableau, libre à vous d'utiliser vos codes ou les modules spécifiques pour manipuler les images.

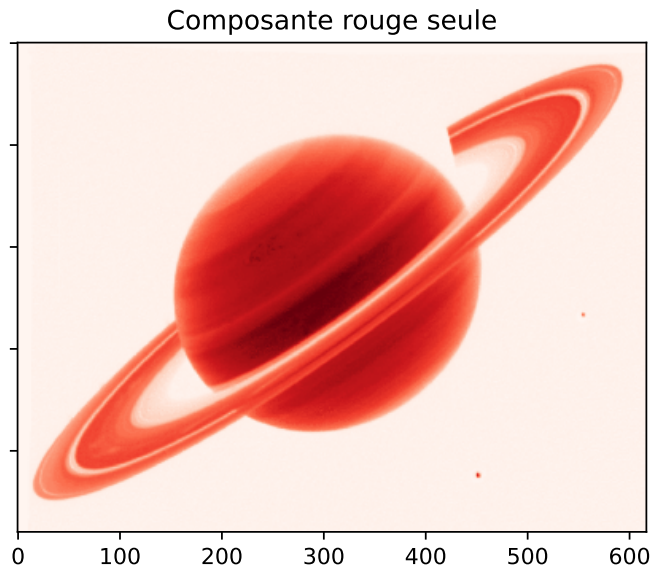
Utilisation de matplotlib

Par exemple :

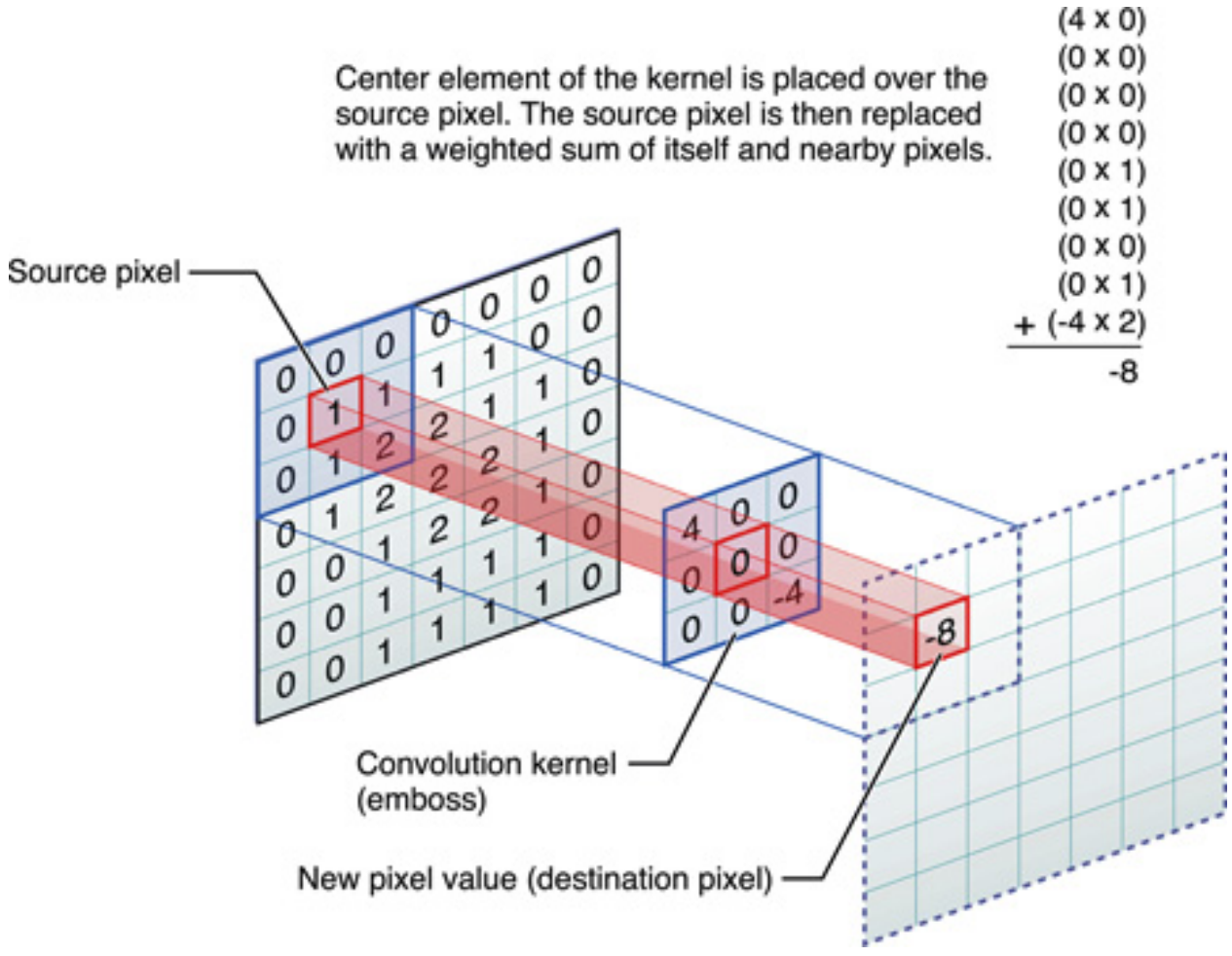
In [1]:

```
1 import matplotlib.pyplot as plt
2 fig, axs = plt.subplots(1, 2, figsize=(12,4),
      sharex = True, sharey=True)
3 axs[0].imshow(img)
4 axs[0].set_title("Image originale")
5 axs[1].imshow(img[:,:,0], cmap = "Reds")
6 axs[1].set_title("Composante rouge seule")
7 plt.show()
```





Exemples de convolution sur image



Exemples de convolution sur image

Exemple de noyau : le filtre emboss fait ressortir les contours.

Original

Emboss

Exemples de convolution sur image

La convolution d'une image nécessite la définition d'une fenêtre glissante (2D), encore appelée noyau de convolution.

Deux exemples simples pour "lisser" une image :
En pratique, atténuer la visibilité de pixels très différents des pixels environnants (et probablement parasités)

- moyenne glissante 3×3 avec le noyau :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

- gaussienne 3×3 glissante avec le noyau :

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

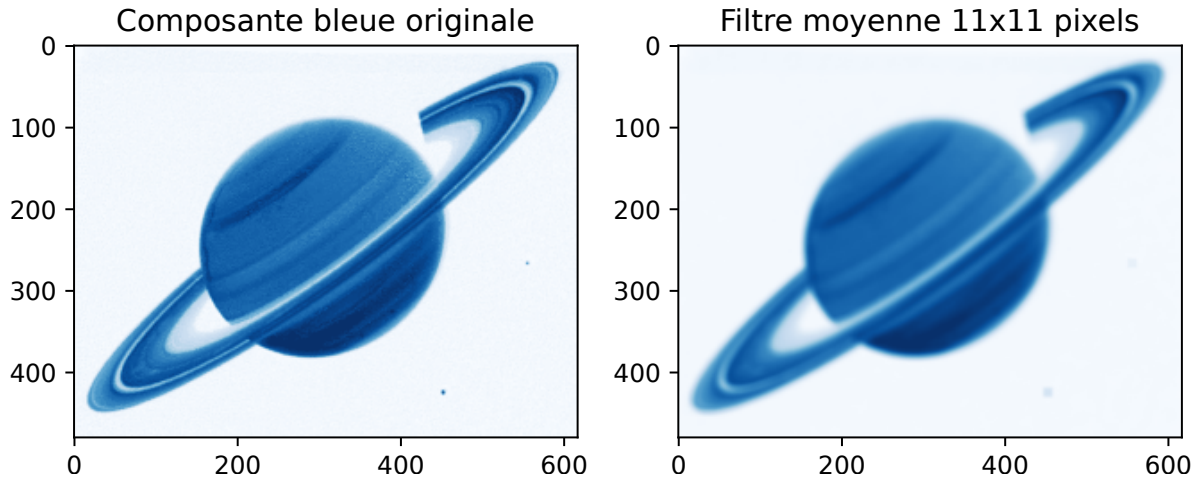
  (cela revient à faire une moyenne pondérée sur 9 pixels).

Pour s'amuser : `http://setosa.io/ev/image-kernels/`

Exemples de convolution sur image

In [2]:

```
from scipy.signal import convolve2d

img = mpimg.imread("data/saturne.png")
noyau = np.ones((21, 21)) # moyenne 21 x 21
img_conv = convolve2d(img[:,:,2],noyau,'same')

fig, ax = plt.subplots(1, 2, figsize=(8,4))
ax[0].imshow(img[:, :, 2], cmap="Blues")
ax[0].set_title("Composante bleue originale")
ax[1].imshow(img_conv, cmap="Blues")
ax[1].set_title("Filtre moyenne 21x21 pixels")
plt.show()
```



Filtre médiane glissante

Une méthode cousine de la convolution est le filtre médiane glissante. Au lieu de faire la moyenne des pixels dans une fenêtre glissante, on en réalise la **médiane**.

Ce filtre est utile pour supprimer les pixels à valeur anormalement hautes ou basses sans modifier les pixels voisins.

Exemple sur un vecteur 1D :
Vecteur original $x = (2, 80, 6, 3)$.
Vecteur résultant d'une médiane sur 3 points (il faut un nombre impair)
$y1 = \text{med}(2, 2, 80) = 2$
$y2 = \text{med}(2, 80, 6) = \text{med}(2, 6, 80) = 6$
$y3 = \text{med}(80, 6, 3) = \text{med}(3, 6, 80) = 6$
$y4 = \text{med}(6, 3, 3) = \text{med}(3, 3, 6) = 3$
Donc $y = (2, 6, 6, 3)$.

Filtre médiane glissante

La fonction est dans `scipy.signal` :

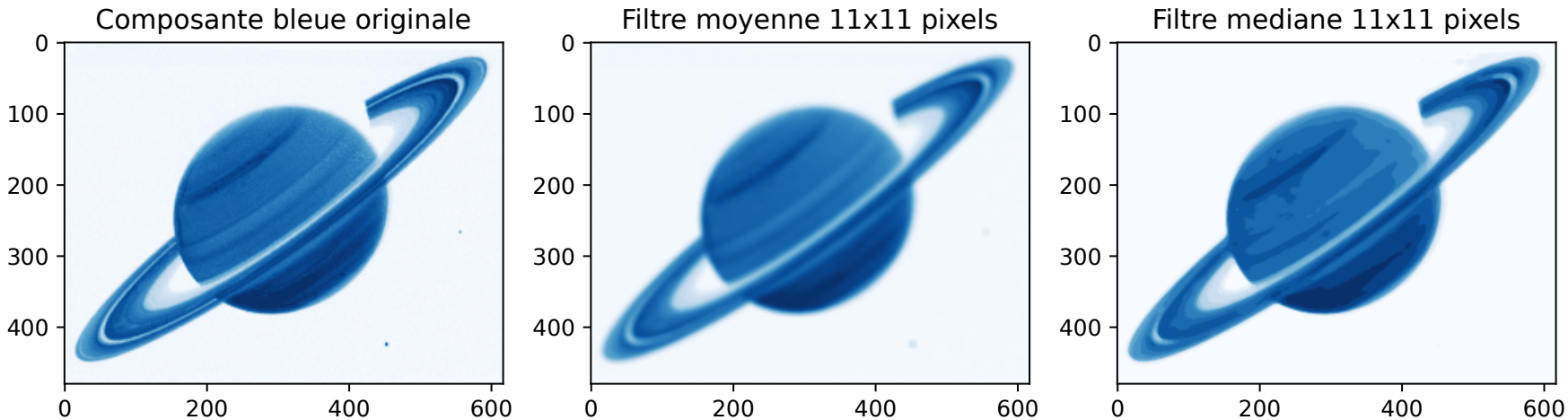
In [3]:
```
from scipy.signal import medfilt

x = np.array([2, 80, 6, 3])
medfilt(x, 3)
```

Out[3]:
```
array([2, 6, 6, 3])
```

Filtre médiane glissante

In [4]:

```
1  img_med = medfilt(img[:, :, 2], 21)
2
3  fig, ax = plt.subplots(1, 3, figsize=(12,4))
4  ax[0].imshow(img[:, :, 2], cmap="Blues")
5  ax[0].set_title("Composante bleue originale")
6  ax[1].imshow(img_conv, cmap="Blues")
7  ax[1].set_title("Filtre moyenne 21x21 pixels")
8  ax[2].imshow(img_med, cmap="Blues")
9  ax[2].set_title("Filtre mediane 21x21 pixels")
10 plt.show()
```



Autres traitements

- masquer des valeurs par des valeurs np.nan
- remplacer des valeurs par une autre
- étirer l'histogramme des pixels pour augmenter les contrastes
- multiplier par une valeur
- passer en échelle logarithmique pour faire ressortir les pixels faibles
- analyse en fréquence
- défloutage
- ...