Télédétection aérospatiale
Approfondissement : Traitement spatial
&
Programmation ENVI-IDL

Mathieu Fauvel
http://moodle-ensat.inp-toulouse.fr/

Institut National Polytechnique de Toulouse (INPT)
Ecole Nationale Supérieure Agronomique (ENSAT)
UMR 1201 DYNAFOR INRA - INPT/ENSAT
mathieu.fauvel@ensat.fr

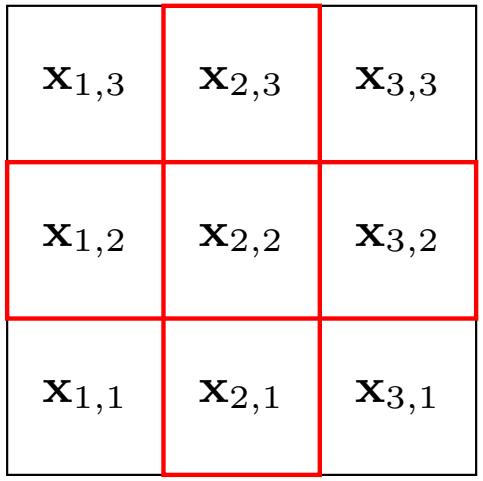
2011-2012

Traitement dans le domaine spatial

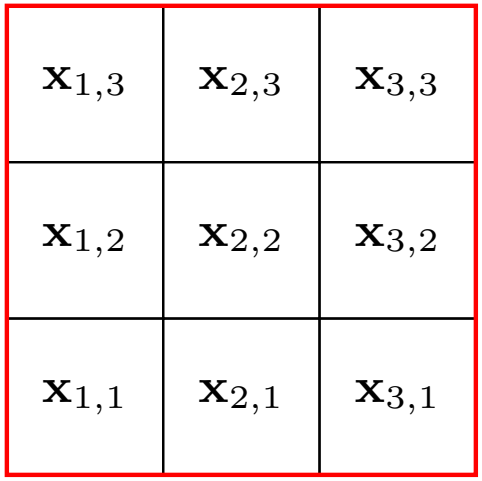
Programmation IDL-ENVI

Notion de voisinage

Le voisinage d'un pixel correspond aux pixels directement connectés à celui-ci. On définit plusieurs types de connections :



4-connexe



8-connexe

A partir du voisinage, on peut définir plusieurs opérations dans le domaine spatial :

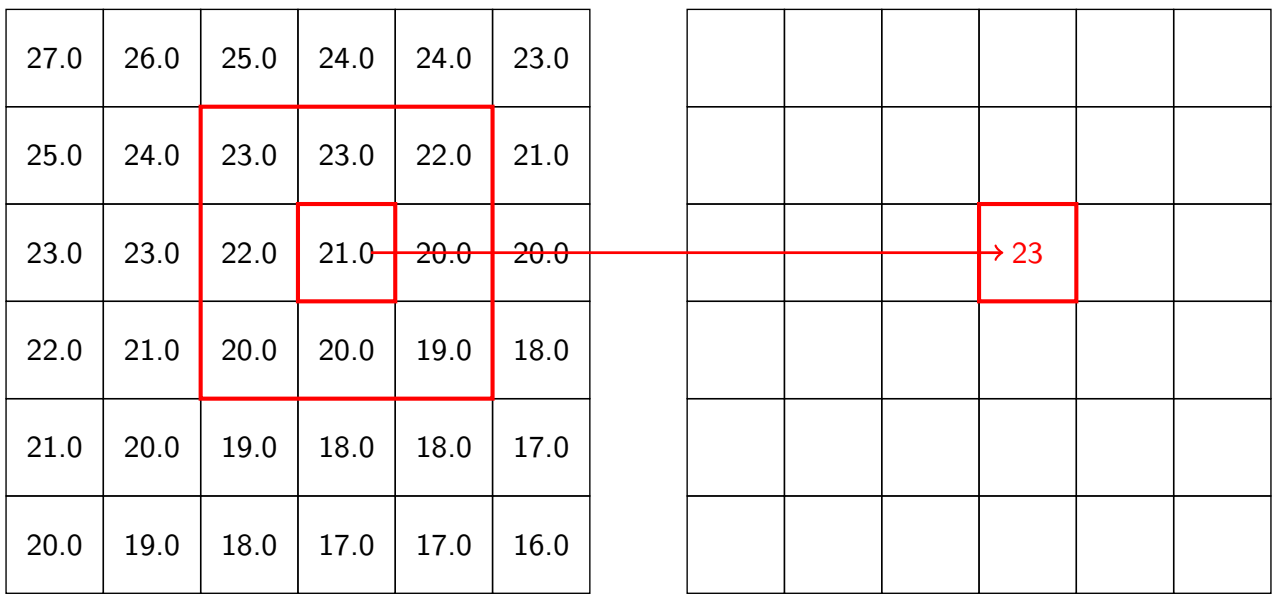
- Débruitage,
- Extraction de contours,
- Extraction de formes,
- ...

Principes des algorithmes à gabarit

Etapes :

1. Définition du gabarit G : 4/8-connexe et taille
2. Définition de l'opération f à réaliser sur le voisinage
3. Application du filtre sur tout les pixels de l'image

$$\mathbf{x}_{ij}^{f} = f(\mathbf{x}_1, \ldots, \mathbf{x}_N),\ \mathbf{x}_n \in G(i,j)$$



Exemple filtre Max

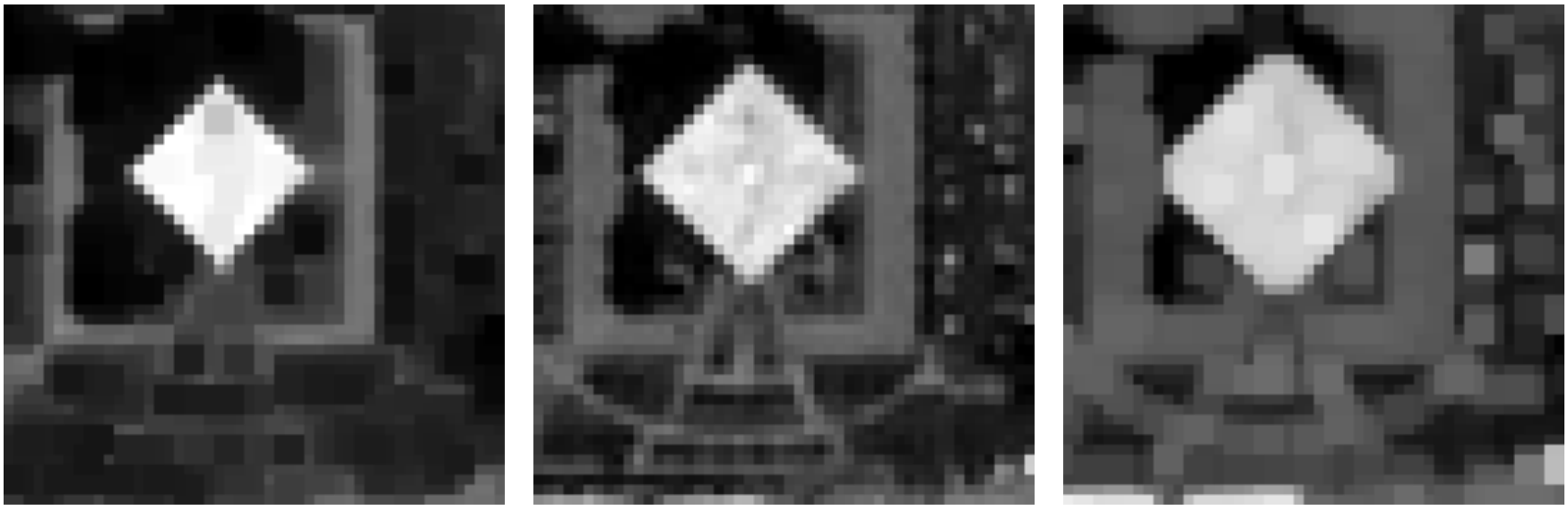
Filtre $\max$

Pseudo-code réalisant un filtre spatial $\max$: cas d'une fenêtre 3×3

```
{Parcours toute l'image}
for i = 1 → H do
   for j = 1 → W do
      temp = x[i, j] {Récupère la valeur du pixel considéré}
      for m = −1 → 1 do
         for n = −1 → 1 do
            if temp < x[i − m, j − n] then
               temp = x[i − m, j − n]{Recherche de la valeur max}
            end if
         end for
      end for
      x[i, j]^f =temp{On affecte la nouvelle valeur au pixel}
   end for
end for
```

Le filtre $\min$ est réalisé de la même façon, l'opérateur « $<$ » devient « $>$ » !

Exemple

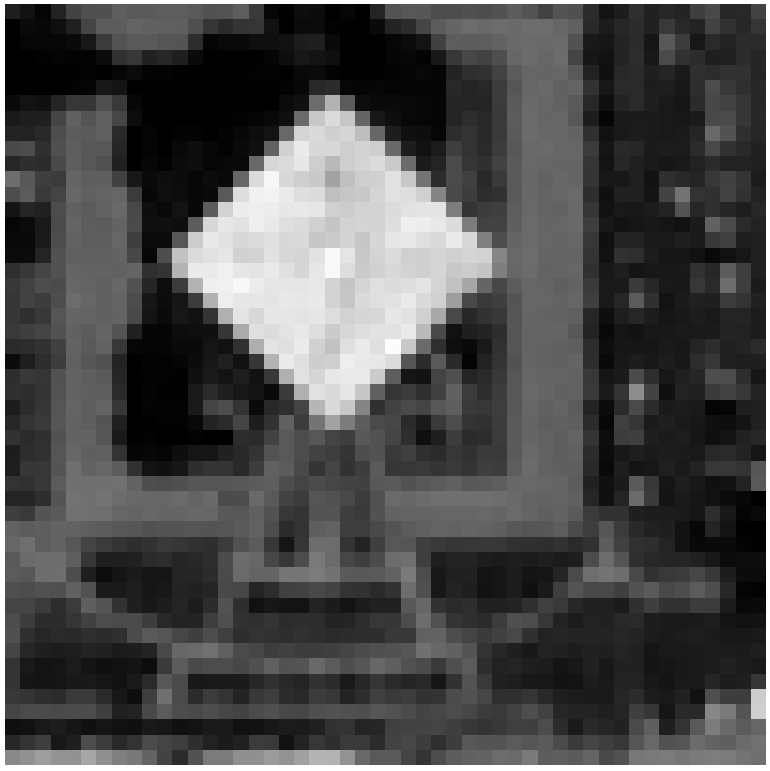


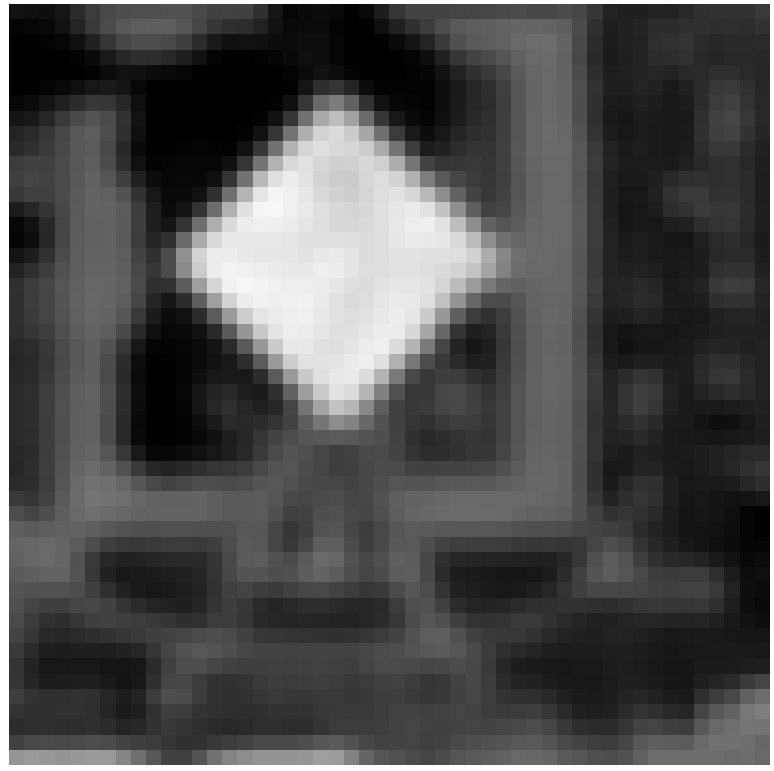
Filtre moyenneur

Pseudo-code réalisant un filtre spatial moyenneur : cas d'une fenêtre 3×3

```
{Parcours toute l'image}
for i = 1 → H do
   for j = 1 → W do
      temp = 0 {Initialisation de la variable temporaire}
      for m = −1 → 1 do
         for n = −1 → 1 do
            temp = temp + x[i − m, j − n]
         end for
      end for
      x[i, j]^f = temp / 9 {On affecte la nouvelle valeur au pixel}
   end for
end for
```

Exemple





Programmation du filtre moyen ENVI-IDL avec *Band Math*

```
function u_dilate, b, sq=sq
sb = size(b)
ns = sb[1]
nc = sb[2]

; Do some initialization
b = float(b)
bm = fltarr(ns,nc)

; Compute the moving window
st = fix((sq-1)/2)

for j=st,nc-(1+st) do begin
    for i=st,ns-(1+st) do begin
        bm[i,j]=mean(b[i-st:i+st,j-st:j+st])
    endfor
endfor

return, bm
end
```

Filtre médian

La médianne d'une série est la valeur m telle que le nombre de valeurs de l'ensemble supérieures ou égales à m est égal au nombre de valeurs inférieures ou égales à m.

3	6	6	8	6

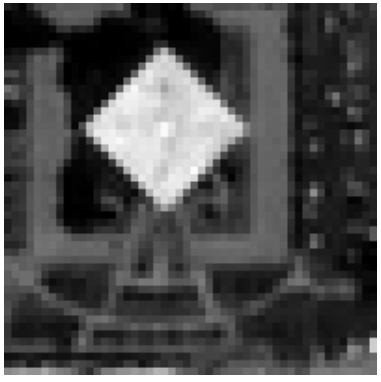
$= 6$

L'algorithme suit le même principe que ceux des filtres $\max$, $\min$ et moyenne.

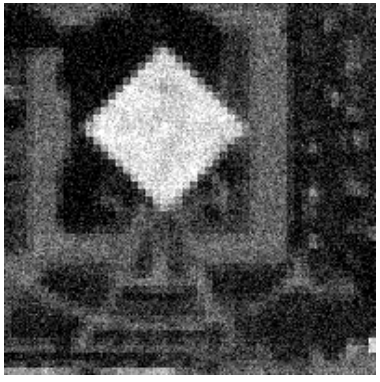
Débruitage 1/6

Il existe plusieurs types de bruits :

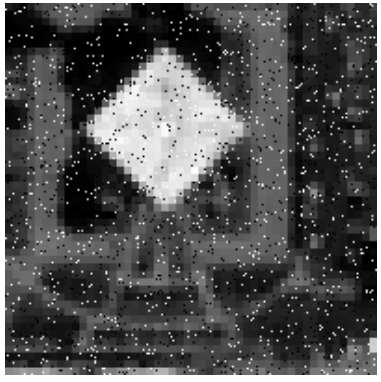
- Gausien : $\mathbf{x}_b = \mathbf{x} + b,\ b \sim \mathcal{N}(\mathbf{0}, \varepsilon)$
- Poivre et sel : $\mathbf{x}_b = \mathbf{x}$ ou 0 ou 255, de manière aléatoire dans l'image
- Speckle : $\mathbf{x}_b = (1 + b)\mathbf{x},\ b \sim \mathcal{N}(\mathbf{0}, \varepsilon)$ (concerne l'imagerie radar).



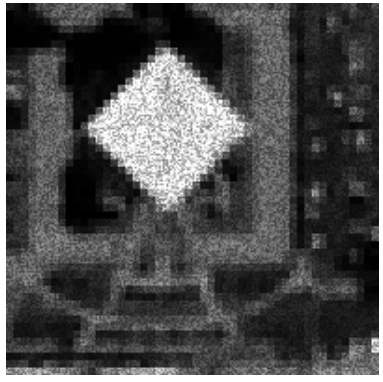
Image



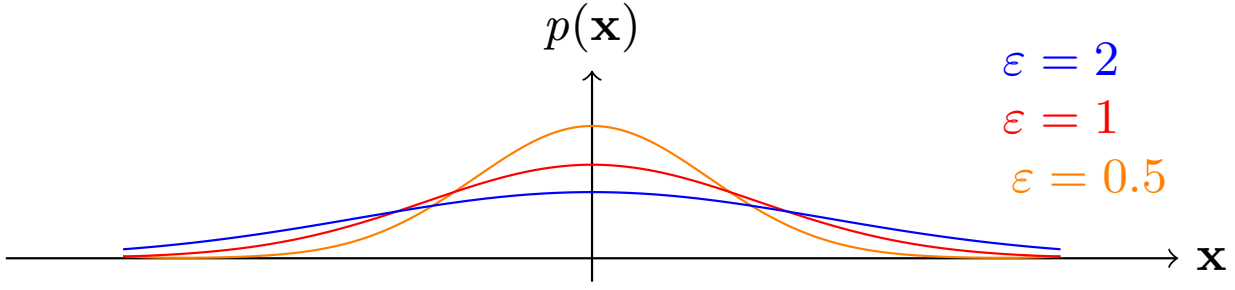
Gaussien



Poivre et Sel



Speckle



Débruitage 2/6

Cas du bruit Gaussien : filtre moyen

100	100	100	100	100	100
100	100	100	100	100	100
100	100	150	150	100	100
100	100	150	150	100	100
100	100	100	100	100	100
100	100	100	100	100	100

Image

98	104	97	102	105	101
103	99	102	98	102	105
97	104	150	147	99	98
99	106	154	152	98	102
98	101	97	97	100	100
99	100	103	99	102	97

Image bruitée

101	100	101	101	102	103
100	106	111	111	106	102
101	113	124	122	111	101
100	112	123	122	110	100
100	106	112	111	105	100
99	100	100	100	99	99

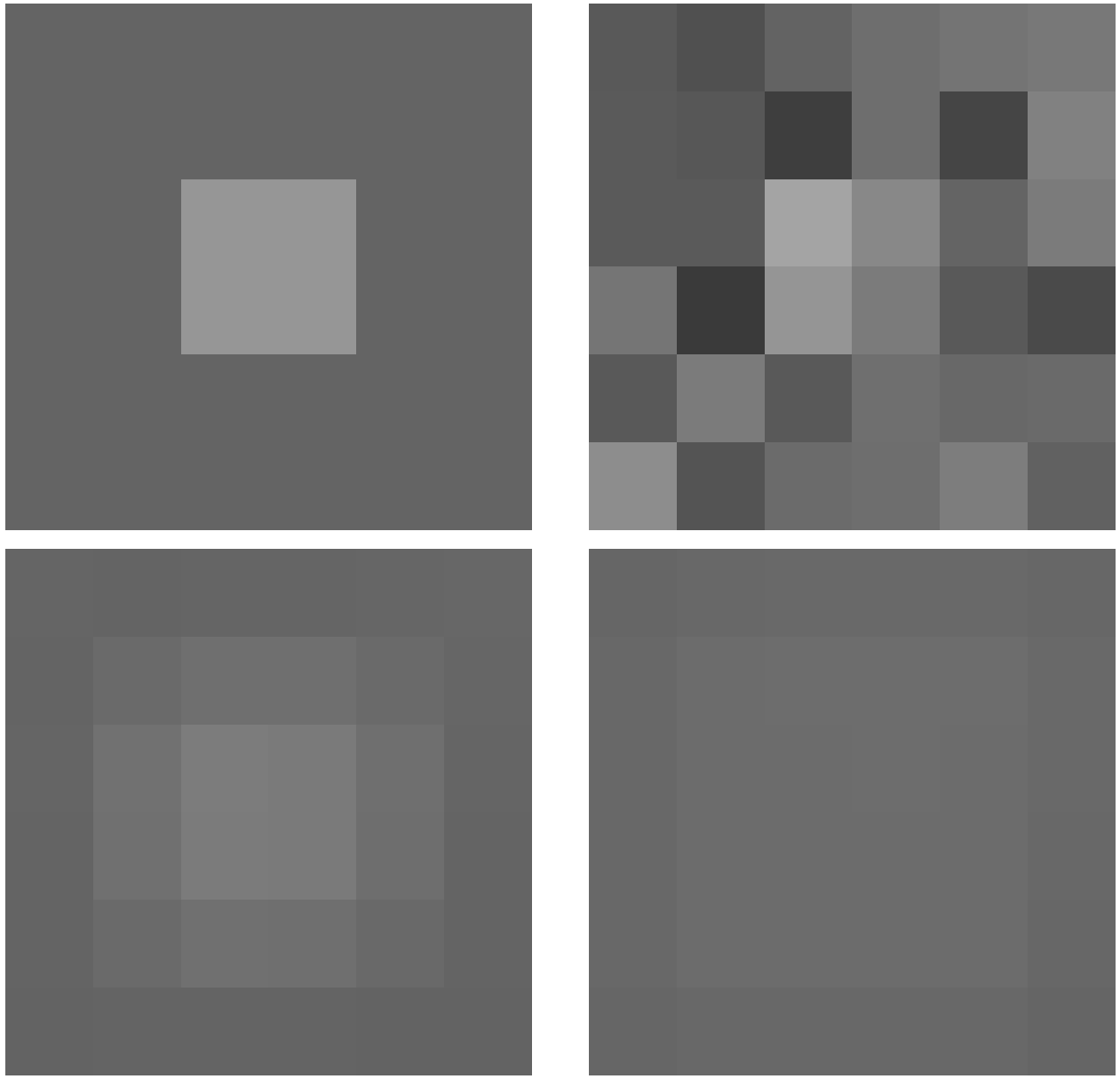
3×3

102	104	105	105	105	103
104	108	109	109	109	105
104	108	108	109	108	105
104	108	108	108	108	104
104	108	108	108	108	103
102	104	104	104	104	101

5×5

Débruitage 3/6

Cas du bruit Gaussien : filtre moyen



Débruitage 4/6

Cas du bruit poivre et sel : filtre median

100	100	100	100	100	100
100	100	100	100	100	100
100	100	150	150	100	100
100	100	150	150	100	100
100	100	100	100	100	100
100	100	100	100	100	100

Image

255	100	100	100	100	100
100	100	0	100	0	100
255	100	150	150	255	100
100	100	150	150	100	100
0	100	100	100	100	100
100	100	100	255	100	100

Image bruitée

100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	150	100	100
100	100	100	150	100	100
100	100	100	100	100	100
100	100	100	100	100	100

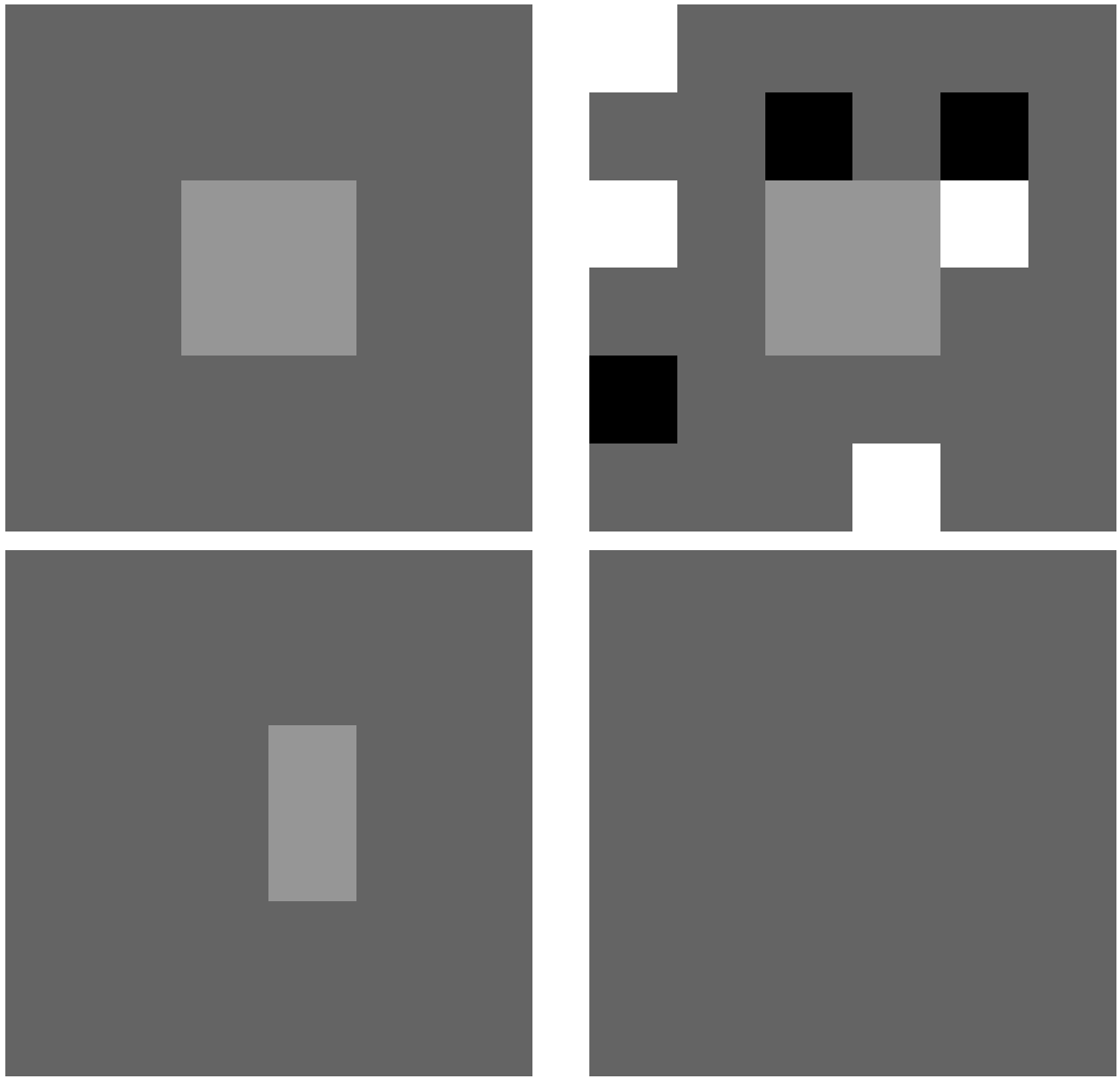
3×3

100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	100	100	100

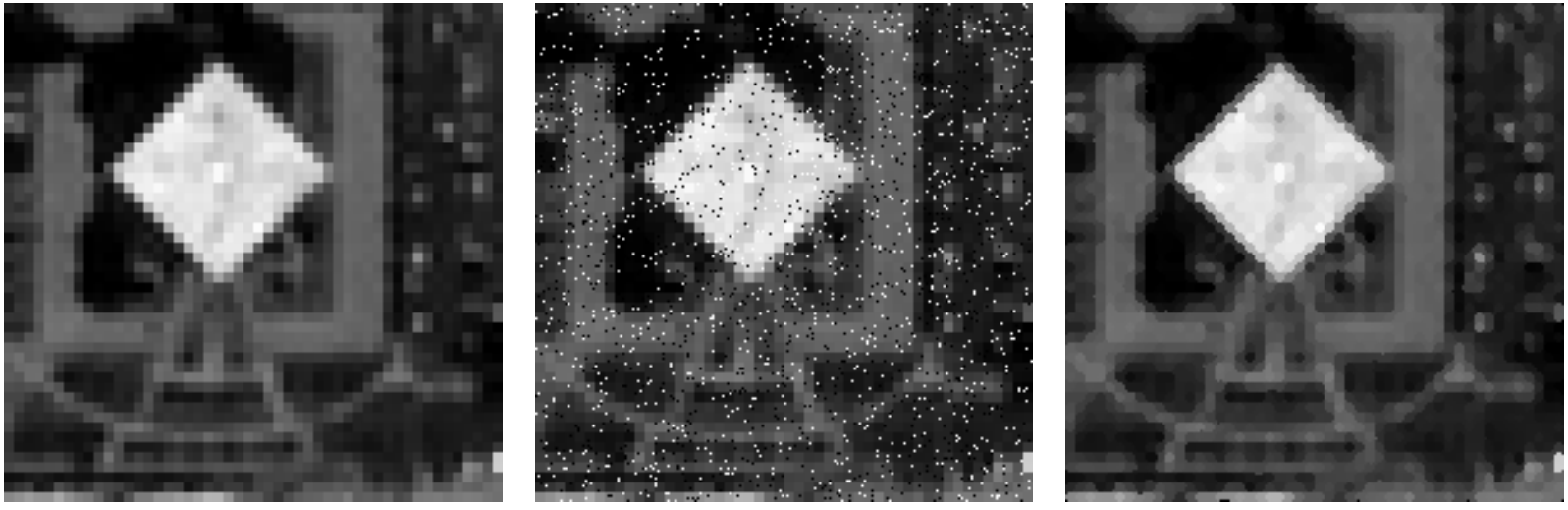
5×5

Débruitage 5/6

Cas du bruit Gaussien : filtre median

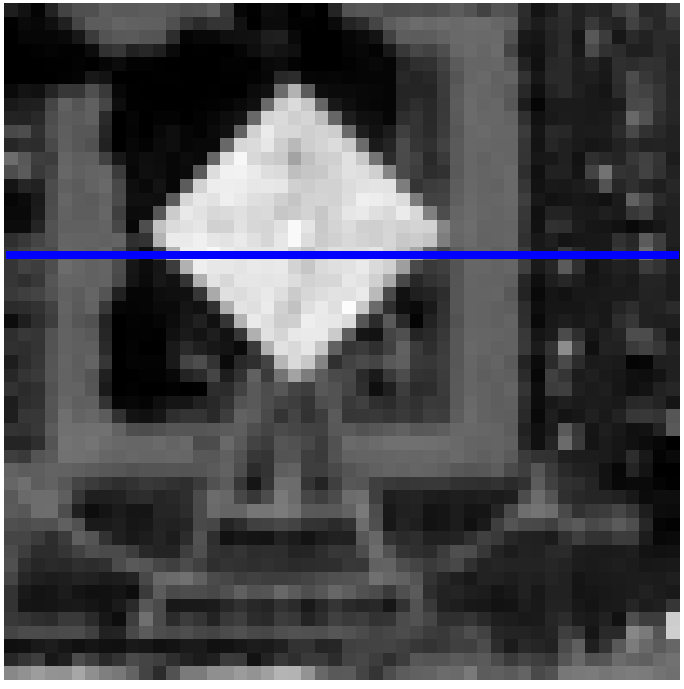


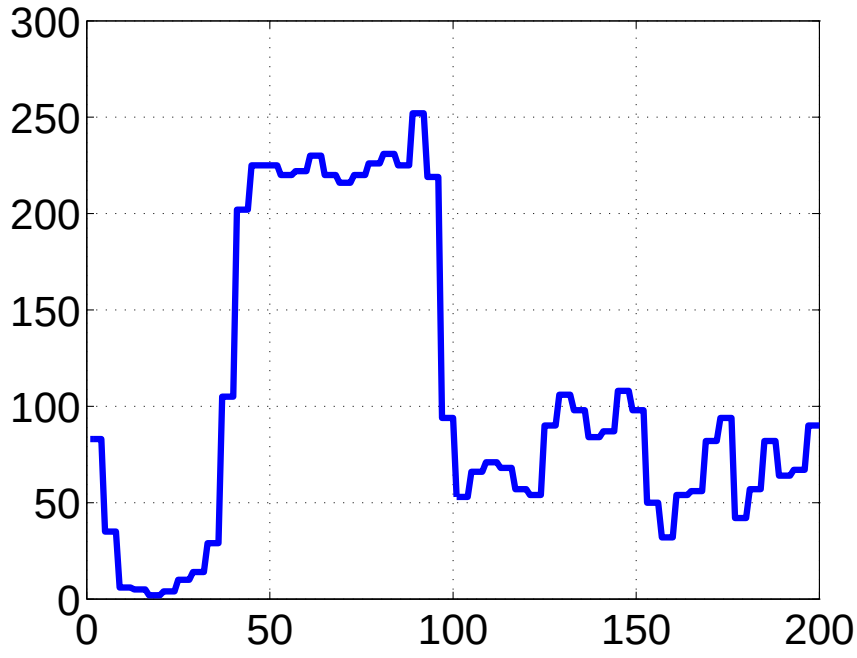
Débruitage 6/6



Détection de contours 1/3

Qu'estce qu'un contour dans une image ?





En première approximation, une variation importante du niveau de gris entre deux pixels.

Détection de contours 2/3

Pour mesurer cette variation, on peut calculer la dérivée première de l'image :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

On l'approche par plusieurs gabarit (du moins robuste au plus robuste) :

0	0	0
0	-1	1
0	0	0

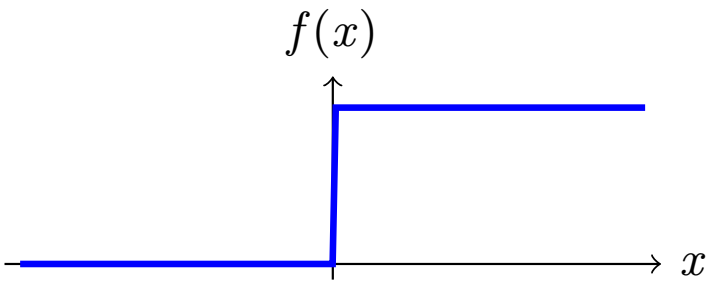
Gradient

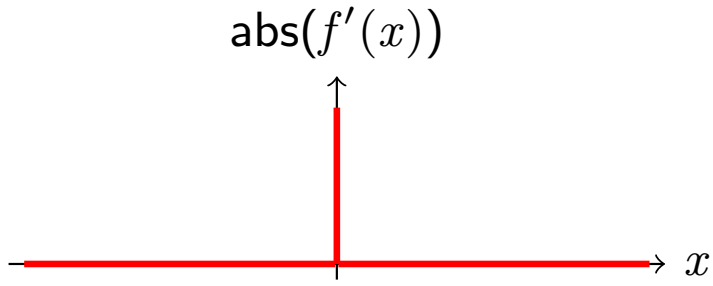
-1	0	1
-1	0	1
-1	0	1

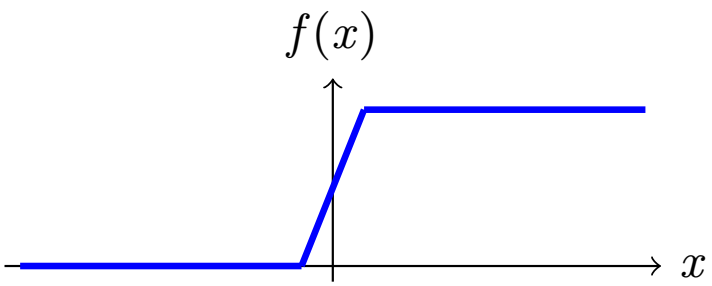
Prewitt

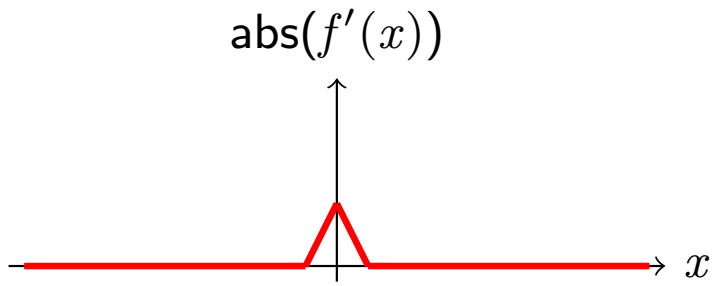
-1	0	1
-2	0	2
-1	0	1

Sobel









Détection de contours 3/3

A pratique, on applique les gabarits dans plusieurs directions :

-1	0	1
-2	0	2
-1	0	1

Verticale

-1	-2	-1
0	0	0
1	2	1

Horizontale

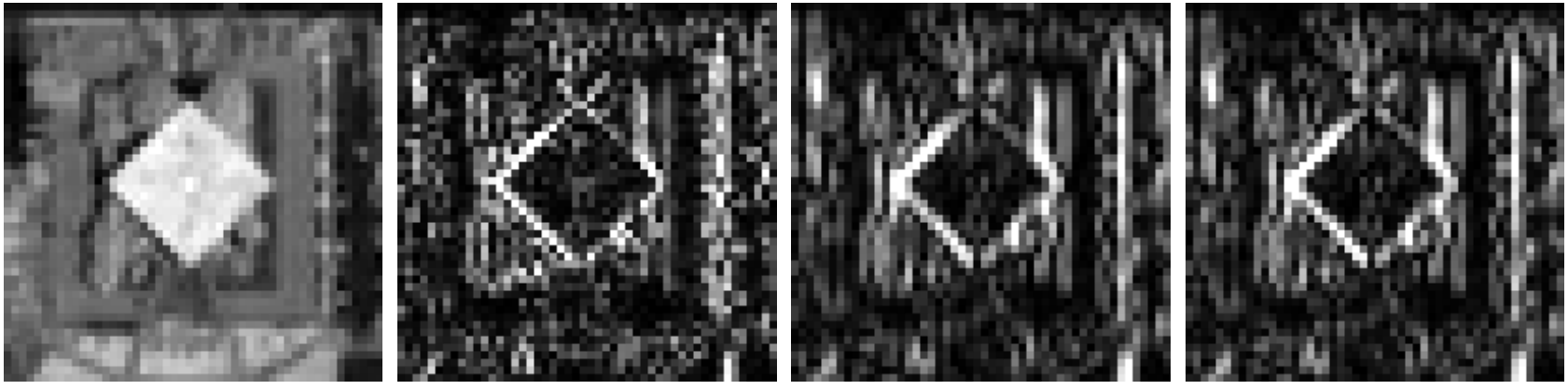
-2	-1	0
-1	0	1
0	1	2

Diagonale : NE-SW

0	-1	-2
1	0	-1
2	1	0

Diagonale : NW-SE

et on somme la valeut absolue des réponses des filtres dans toutes les directions.



Traitement dans le domaine spatial

Programmation IDL-ENVI

IDL ?

- Interactive Data Language (IDL).
- Laguage de programation propriétaire (payant) de Research Systems, Inc. (RSI) de Boulder, Colorado EU (Kodak).
- Au départ : ensemble de programmes et fonctions pour l'analyse de données issues de missions de la NASA.
- Actuellement, utilisé pour l'analyse et la visualisation de données multi-dimensionnelles.
- ENVI est codé en IDL : possibilité de coder simplement des algorithmes en utilisant les fonctions « offertes » par IDL.
  - Fonction simple $\rightarrow$ *Band Math*
  - Fonction plus avancée $\rightarrow$ création de widget et interface utilisateur.

IDL interactif

Il est possible d'utiliser IDL de manière interactive :

```
; les point-virgules sont utilises pour les commentaires
IDL> a = 3 ; assignement de valeur

IDL> print, a + 4 ; On affiche la valeur de a+4
                  ; avec la commande "print" (voir doc)
7

IDL> print, a + 4. ; le type  est automatiquement ajuste
7.00000            ; sur la precision la plus haute

IDL> print, a + 4D ; double precision
7.0000000

IDL> print, a*1e-9
3.00000e-09

IDL> print, A ; Insensible a la case
3
```

Type IDL

Type	Octet	Dynamique	Déclaration	Conversion
Byte	1	0-255	byarr	byte
integer	2	$\pm 2^{15-1}$	intarr	fix
unsigned integer	2	$0 - 2^{16-1}$	uintarr	uint
floating-point	4	$\pm 10^{38}$	fltarr	float
double-precision	8	$\pm 10^{308}$	dblarr	double

A priori, IDL s'occupe de tout :

```
IDL> a=2

IDL> b=5.0

IDL> c = a+b

IDL> help, a, b, c
A       INT       = 2
B       FLOAT     = 5.00000
C       FLOAT     = 7.00000
```

Tableau en IDL

- IDL est un language orienté « tableau ».
- Les opérations sont optimisées sur les tableaux (mémoire et temps de clacul)
- Allocation automatique et dynamique
- Il faut absolument éviter les boucles sur les tableaux !
- Pour cela, il faut utiliser les fonctions pré-définies d'IDL (il y en a plein !)

Vecteurs en IDL

```
IDL> vec = [3,5,1,4,5] ; definition d'un vecteur (tableau 1D)

IDL> print, vec          ; on affiche les
3       5       1       4       5
vec[0]  vec[1]  vec[2]  vec[3]  vec[4]

IDL> vec[3] = 9 ; On peut changer chaque variable

IDL> print, vec[3]
9
```

Voir sur le Guide d'IDL, ce que font les fonctions : *size*, *max*, *min*, *mean* et *std*.

Matrices en IDL

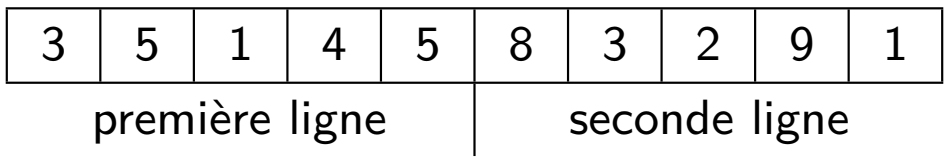
```
IDL> mat = [[3,5,1,4,5],[8,3,2,9,1]]
IDL> print, mat ; 5 colonnes et 2 lignes
3       5       1       4       5
8       3       2       9       1

IDL> help, mat
MAT             INT       = Array[5, 2]

IDL> print, mat[3,1]
9 ; Colonne 3 et ligne 2 -> inverse du C/C++, matlab etc...

IDL> print, mat[3+5*1] ; ecriture equivalente
9
```

Une matrice dans IDL :



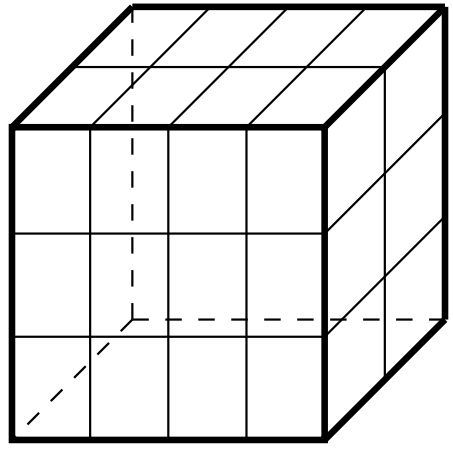
Pour une matrice $M \times N$: mat$[m, n]$ = mat$[m + n * M]$ (il y a M éléments par ligne).

Tableaux n-D en IDL

```
IDL> a = indgen(4,3,2)

IDL> print, a
       0       1       2       3
       4       5       6       7
       8       9      10      11

      12      13      14      15
      16      17      18      19
      20      21      22      23
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$n=0, p=0$				$n=1, p=0$				$n=2, p=0$				$n=0, p=1$				$n=1, p=1$				

Pour un tableau $M \times N \times P$, on a $a[m, n, p] = a[m + M * n + M * N * p]$

Accéder aux éléments d'un tableau

```
IDL> mat = [[3,5,1,4,5],[8,3,2,9,1]]

IDL> print, mat
       3       5       1       4       5
       8       3       2       9       1

IDL> print, mat[3,*] ; affiche toute la colonne 4 (4-1)
          4
          9

IDL> print, mat[2:4,0] ; elements des colonnes 3 a 5
          1       4       5 ; de la ligne 1

IDL> q = [3,2,0]

IDL> print, mat[q,0]
          4       1       3
```

Recherche dans un tableau - Exemple de fonction IDL

```
IDL> mat = [[3,5,1,4,5],[8,3,2,9,1]]
IDL> q = where(mat le 2, nq) ; trouve les elements <= 2
IDL> print, q ; indice du tableau
           2           7           9
IDL> print, mat[q] ; valeur du tableau
           1           2           1
IDL> print, nq ; nq est le nombre d'element trouve
           3
```

La fonction *where* est environ 4 fois plus rapide qu'une boucle sur un vecteur !

Opérations sur les tableaux

```
IDL> n = 100 ; Nombre d element
IDL> x = findgen(n+1)*2*!pi/n ; x=[0,2pi] par pas de 2pi/100
IDL> y = cos(x) ; directement sur toute les variables !
IDL> m = mean(y)
IDL> print, m
   0.00990102
IDL> m = max(y)
IDL> print, m
   1.00000
IDL> m = median(y)
IDL> print, m
   1.19249e-08
IDL> dx = x+x
IDL> print, dx[1]
   0.125664
IDL> print, x[1]
   0.0628319
```