

## Cartes Topologiques

### 1 Chargement de la somtoolbox

- Adresse de la somtoolbox → <http://www.cis.hut.fi/projects/somtoolbox/download/>
- Charger le fichier : **somtoolbox2\_Mar\_17\_2005.zip**

Vous pouvez cliquer directement sur le lien suivant :

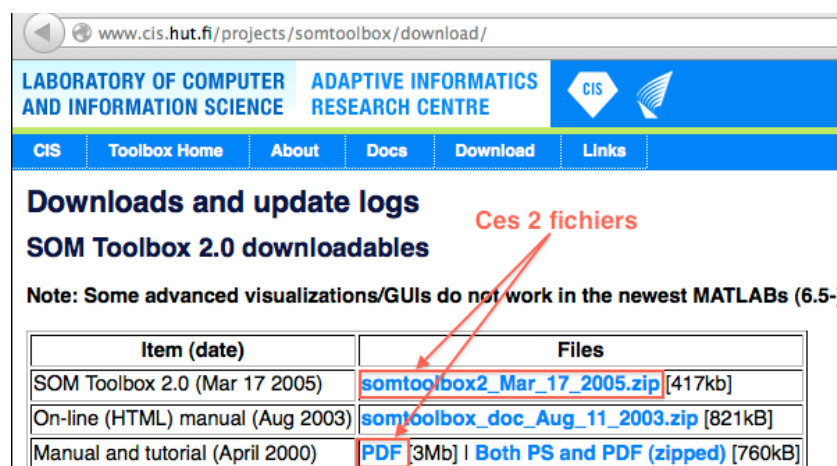
[http://www.cis.hut.fi/projects/somtoolbox/package/somtoolbox2\\_Mar\\_17\\_2005.zip](http://www.cis.hut.fi/projects/somtoolbox/package/somtoolbox2_Mar_17_2005.zip)

↪ Décompresser le fichier (sous linux : `unzip somtoolbox2_Mar_17_2005.zip`)

↪ Vous avez alors un répertoire **somtoolbox** contenant les fonctions du logiciel

- Charger aussi la documentation : **Manual and tutorial (April 2000)**

↪ <http://www.cis.hut.fi/projects/somtoolbox/package/papers/techrep.pdf>



### Utilisation de somtoolbox avec octave, qui est compatible avec Matlab et libre/open-source

Si vous utilisez **octave** alors il faut changer tous les `'&'` en `'&&'` et tous les `'|'` en `'||'`. Apparemment Octave n'aime pas qu'on utilise un opérateur binaire là où on devrait utiliser un opérateur logique.

Donc vous pouvez procéder comme suit :

- Créer un répertoire **somtoolboxOctave**
- Copier tous les fichiers contenus dans le répertoire **somtoolbox** dans **somtoolboxOctave**
- Faire les changements suivants dans le répertoire **somtoolboxOctave**
  - 1) Dans les fichiers suivants, remplacer tous les `'&'` en `'&&'` et tous les `'|'` en `'||'`

som\_batchtrain.m

som\_bmus.m

som\_connection.m

som\_grid.m

som\_map\_struct.m

som\_neighborhood.m

sompak\_train.m

som\_randinit.m

som\_seqtrain.m

som\_set.m

som\_topol\_struct.m

som\_train\_struct.m

som\_unit\_coords.m

som\_unit\_dists.m

som\_unit\_neighs.m

vis\_valuetype.m
  - 2) Commenter la ligne 525 (transformer en commentaire en ajoutant `%` au début de la ligne), `"fprintf(1,..."`, du fichier **som\_batchtrain.m** pour la désactiver. Sinon, dans le terminal, le programme commence à afficher les données et attend qu'on appuie sur 'f' pour continuer.

Ensuite, il faut utiliser **somtoolboxOctave** à la place de **somtoolbox**

## 2 Approximation de fonctions de densité

### Les données

Utiliser les instructions suivantes pour générer les données Data1

```
n=300;  
D = rand(n,2);  
save Data1 D
```

### Apprentissage

Utiliser les données Data1 pour vérifier visuellement si les référents de la carte topologique obtenue par apprentissage approximent la densité de probabilité de ces données.

Exécuter les instructions suivantes une à une pour comprendre les différentes étapes de l'apprentissage.

- **Ajouter le chemin d'accès aux fonctions de la somtoolbox (ou somtoolboxOctave)**  
>> addpath somtoolbox → si vous utilisez octave alors : addpath somtoolboxOctave  
→ à modifier selon l'emplacement de somtoolbox (ou somtoolboxOctave)
- **Charger les données**  
>> load Data1 → à modifier selon l'emplacement de Data1

#### Définition de la structure des données

```
>> sData = som_data_struct(D,'name','donnees1');
```

☞ Pour voir les différents champs de cette structure, taper :

```
>> sData
```

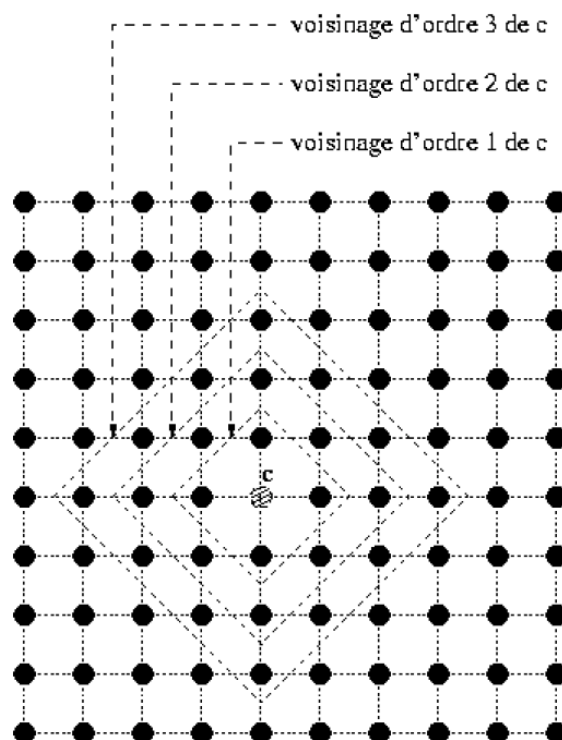
☞ Pour plus de détails sur la commande **som\_data\_struct**, taper :

```
>> help som_data_struct
```

---

#### Rappels

L'algorithme proposé par Kohonen est un algorithme d'auto-organisation qui projette l'espace des données, de grande dimension, sur un espace de faible dimension (en général 1, 2 ou 3), appelé **carte topologique**.



⇒ Cette carte ( $\mathcal{C}$ ) est constituée d'un **ensemble de neurones interconnectés selon une structure de graphe non orienté**.

- ⇒ La structure de la carte induit une **distance discrète ( $\delta$ ) sur la carte ( $\mathcal{C}$ )** de sorte que la distance **entre toute paire de neurone  $c$  et  $r$  est définie comme étant la longueur du plus court chemin entre  $c$  et  $r$ .**

$$\delta(c, r) = \text{longueur du plus court chemin entre } c \text{ et } r \text{ sur le graphe}$$

- ⇒ Pour chaque **neurone  $c$** , cette distance permet de définir la notion de **voisinage d'ordre  $d$  de  $c$**  comme étant **le sous ensemble des neurones dont la distance avec  $c$  est inférieure ou égale à  $d$ .**

$$\text{Voisinage d'ordre } d \text{ de } c : V_c(d) = \{r \in \mathcal{C}, \delta(c, r) \leq d\}$$

- ⇒ Sur la figure ci-dessus on a représenté les voisinages d'ordre 1, 2 et 3 du neurone  $c$ .

### Initialisation de la structure de la carte topologique

>> msize = [6 6]; → choix d'une carte 2D de 6×6

>> insize = size(sData.data,2); → dimension des données d'apprentissage

#### Rappels sur Matlab

si  $M$  est une matrice contenant  **$n$**  exemples, chacun de dimension  **$m$**  alors

↪ `size(M,1)` donne le nombre de lignes de  $M$  (donc  **$n$** , le nombre d'exemples)

↪ `size(M,2)` donne le nombre de colonnes de  $M$  (donc  **$m$** , la dimension des exemples)

>> lattice = 'rect'; → **choix du maillage** (voir Figure 1 ci-dessous)

☞ maillage rectangulaire : lattice = 'rect'

☞ maillage hexagonale : lattice = 'hexa'

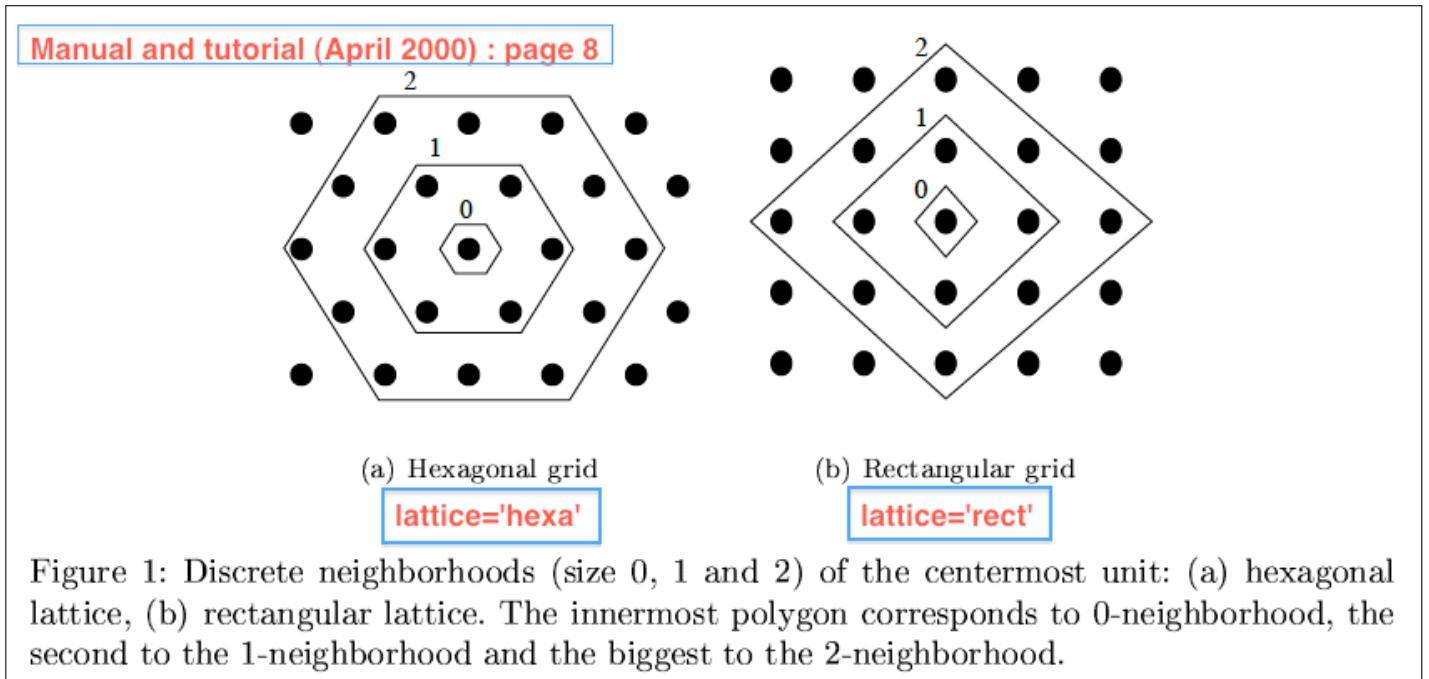
>> shape = 'sheet'; → **choix de la forme de la structure du graphe** (voir Figure 2 ci-dessous)

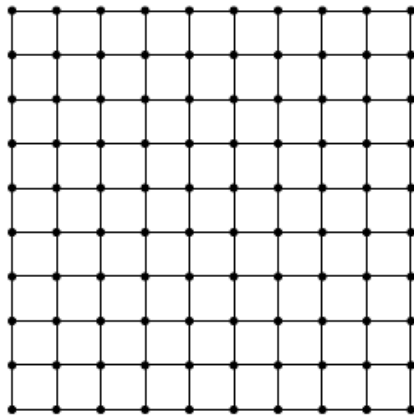
☞ en forme de plan : shape = 'sheet'

☞ en forme de cylindre : shape = 'cyl'

☞ en forme de tore : shape = 'toroid'

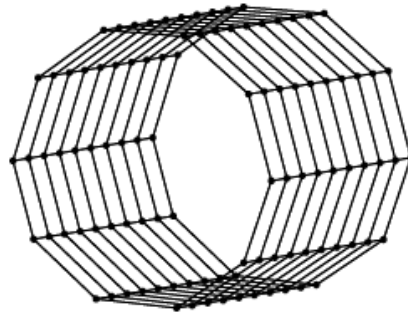
☞ Voir aussi la documentation (techrep.pdf) Manual and tutorial (April 2000) page 8





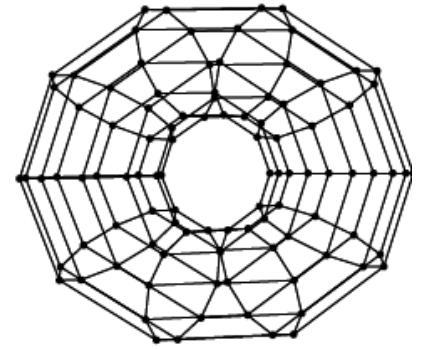
(a) Sheet

`shape='sheet'`



(b) Cylinder

`shape='cyl'`



(c) Toroid

`shape='toroid'`

Figure 2: Different map shapes. The default sheet shape (a), and two shapes where the map topology accommodates circular data: cylinder (b) and toroid (c).

```
>> sMap = som_map_struct(insize,'msize',msize, lattice, shape);
```

☞ Pour voir les différents champs de cette structure, taper :

```
>> sMap
```

☞ Pour plus de détails sur la commande `som_map_struct`, taper :

```
>> help som_map_struct
```

**Initialisation des poids de la carte topologique (des vecteurs référents)**

```
>> sMap = som_randinit(sData, sMap); → som_randinit : initialisation aléatoire
```

**Visualisation des données et de la carte initiale**

```
>> figure
```

```
>> plot(D(:,1),D(:,2),'b+');
```

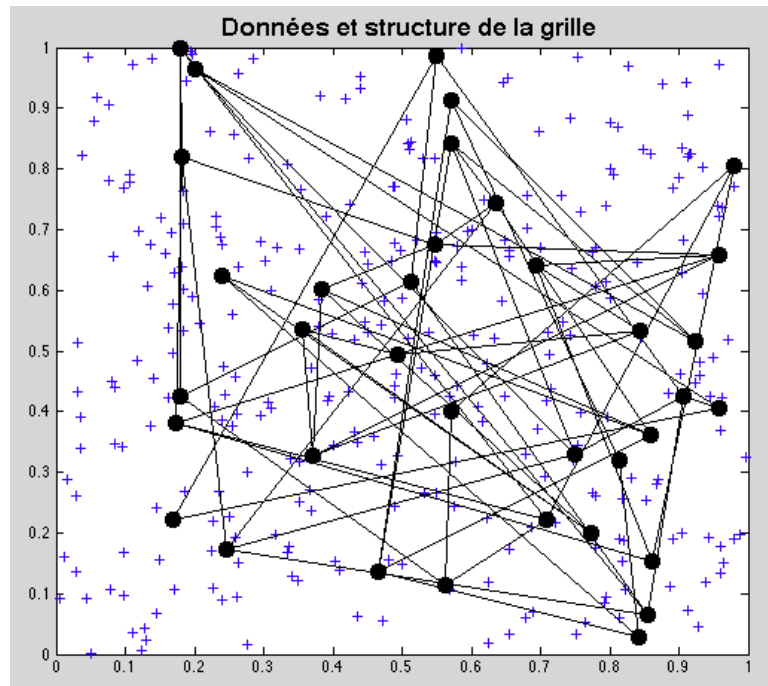
```
>> hold on
```

```
>> som_grid(sMap,'Coord',sMap.codebook)
```

```
>> axis on
```

```
>> title('Données et structure de la grille');
```

On obtient la carte suivante :



**Remarque :** comme les poids de la carte (les vecteurs référents) sont initialisés aléatoirement, il y a beaucoup de neurones qui sont très proches au niveau du graphe (en utilisant la distance  $\delta$ ) mais très loin au niveau géométrique.

```

☞ Essayer aussi l'initialisation som_lininit (qui tient compte des données)
>> sMap = som_lininit(sData, sMap)
☞ Pour plus de détails sur la commande som_lininit, taper :
>> help som_lininit → pour voir les détails sur cette initialisation
Visualisation des données et de la carte initiale avec cette nouvelle initialisation
>> figure
>> plot(D(:,1),D(:,2),'b+');
>> hold on
>> som_grid(sMap,'Coord',sMap.codebook)
>> axis on
>> title('Données et structure de la grille');

```

---

### Rappels

---

- L'apprentissage effectué par les cartes auto-organisatrices cherche à ce que ces vecteurs référents captent au mieux la densité de probabilité sous-jacente aux observations tout en cherchant à respecter une contrainte de conservation de la topologie de la carte :
  - ☞ deux neurones voisins par rapport à la topologie discrète de la carte
    - ↪ doivent être associés à deux vecteurs référents proches dans l'espace des données.
  - ☞ **C'est la notion de conservation de la topologie.**
- L'apprentissage d'une carte topologique revient donc à **minimiser une fonction de coût**
  - ↪ **qui respecte l'ordre topologique.**
- Cette fonction de coût est une extension de celle des k-moyennes dans laquelle
  - ☞ la **distance euclidienne** est remplacée par une mesure que nous appelons **distance généralisée** qui fait intervenir tous les neurones de la cartes via **une fonction de voisinage**  $K^T$ .

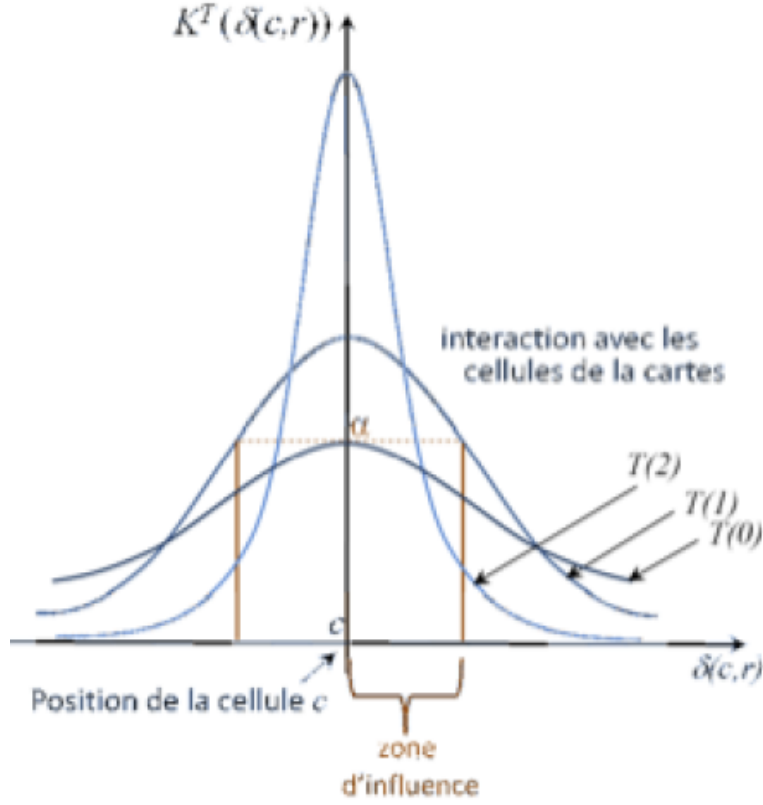
$$J_{som}^T(\chi, \mathcal{W}) = \sum_{\mathbf{z}_i \in \mathcal{A}} \sum_{c \in C} K^T(\delta(c, \chi(\mathbf{z}_i)) \|\mathbf{z}_i - \mathbf{w}_c\|^2$$

- La fonction de coût  $J_{som}^T$  est donc une extension de l'inertie des k-moyennes dans laquelle la distance euclidienne entre une observation  $\mathbf{z}_i$  et son référent  $\mathbf{w}_{\chi(\mathbf{z}_i)}$  est remplacée par une **distance généralisée** (notée  $d^T$ ) qui fait intervenir tous les neurones de la cartes via **une fonction de voisinage**  $K^T$

$$d^T(\mathbf{z}_i, \mathbf{w}_{\chi(\mathbf{z}_i)}) = \sum_{c \in C} K^T(\delta(c, \chi(\mathbf{z}_i)) \|\mathbf{z}_i - \mathbf{w}_c\|^2$$

- ☞  $\delta(c, \chi(\mathbf{z}_i))$  représente la distance sur la carte entre le neurone  $c$  et le neurone affecté à l'observation  $\mathbf{z}_i$ .
  - ↪  $\chi(\mathbf{z}_i)$  représente le neurone affecté à l'observation  $\mathbf{z}_i$  (selon la fonction d'affectation  $\chi$ )
- Cette **fonction de voisinage**  $K^T$  est régie par un paramètre  $T$ , appelé **température**.
  - ☞ qui est déterminant dans l'importance de l'influence entre les neurones (La valeur de  $T$  détermine la taille du voisinage).
  - ⇒ L'influence ou degré de voisinage entre 2 neurones  $c$  et  $r$  sera donc calculée par une fonction  $K^T(\delta(c, r))$
- Grâce à cette fonction  $K^T$ , la contrainte de voisinage introduite par la topologie de la carte pourra être d'autant plus forte que 2 neurones sont proches sur la carte.
- Quand la valeur de  $T$  est suffisamment petite,
  - ↪ cette fonction de coût ( $J_{som}^T$ ) coïncide avec celle des k-moyennes.

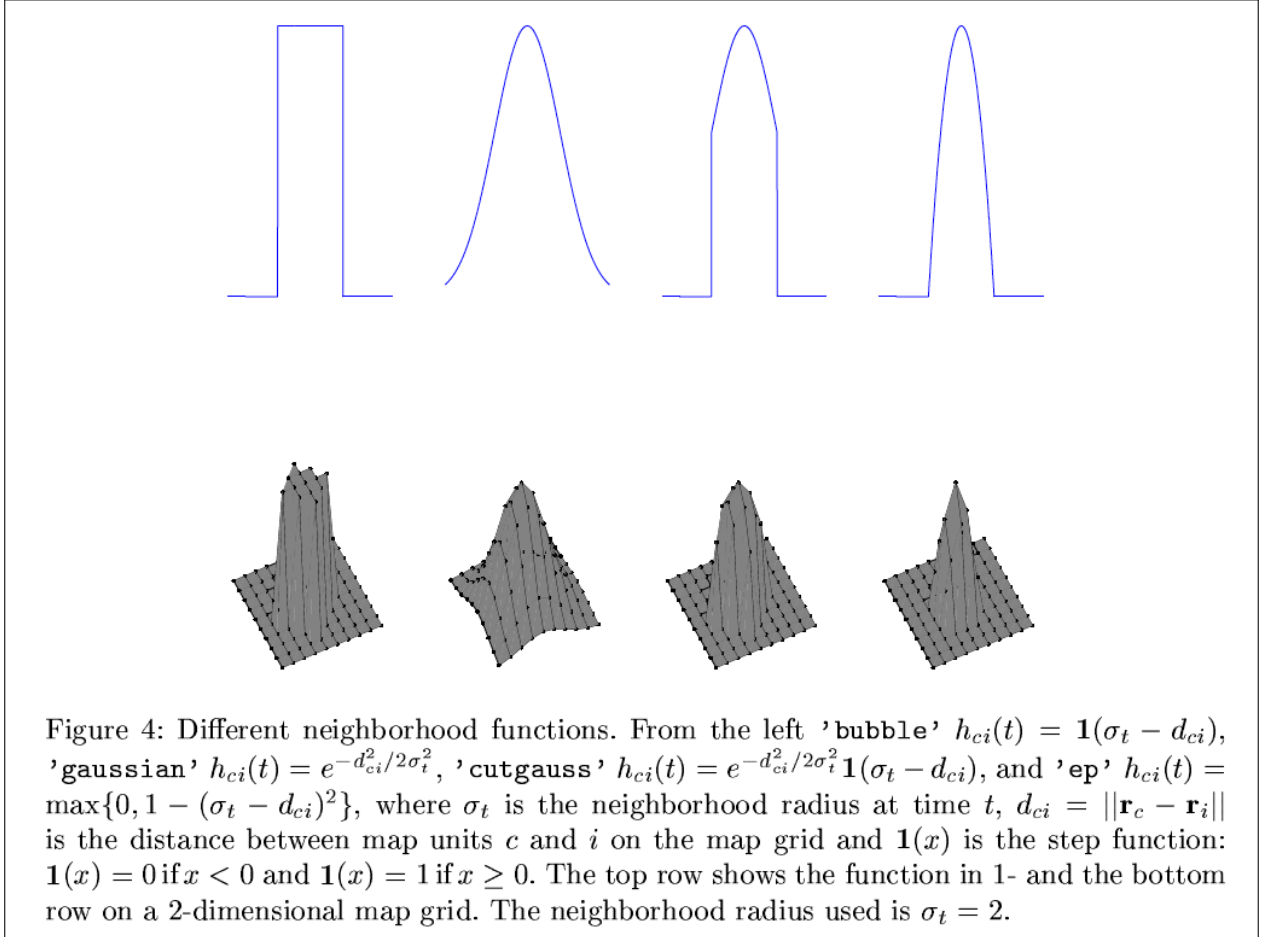
- Exemple de fonction de voisinage (de type gaussien) :  $K^T(\delta) = \exp(\frac{-|\delta|}{T})$  ou  $K^T(\delta) = \exp(-\frac{\delta^2}{T^2})$



- ⇒ Plus  $T$  est grand, plus l'influence des neurones proches est importante.
- ⇒ On peut aussi faire intervenir un paramètre supplémentaire  $\alpha$  pour limiter la zone d'influence.  
Le voisinage d'un neurone  $c$  se définit alors par l'ensemble des neurones  $r$  pour lesquels  
↪ la fonction  $K^T(\delta(c, r))$  est supérieure à ce paramètre  $\alpha$ .

Pour plus de détails sur d'autres fonctions de voisinage

☞ Voir la documentation (techrep.pdf) Manual and tutorial (April 2000) page 10 (figure 4 ci-dessous)



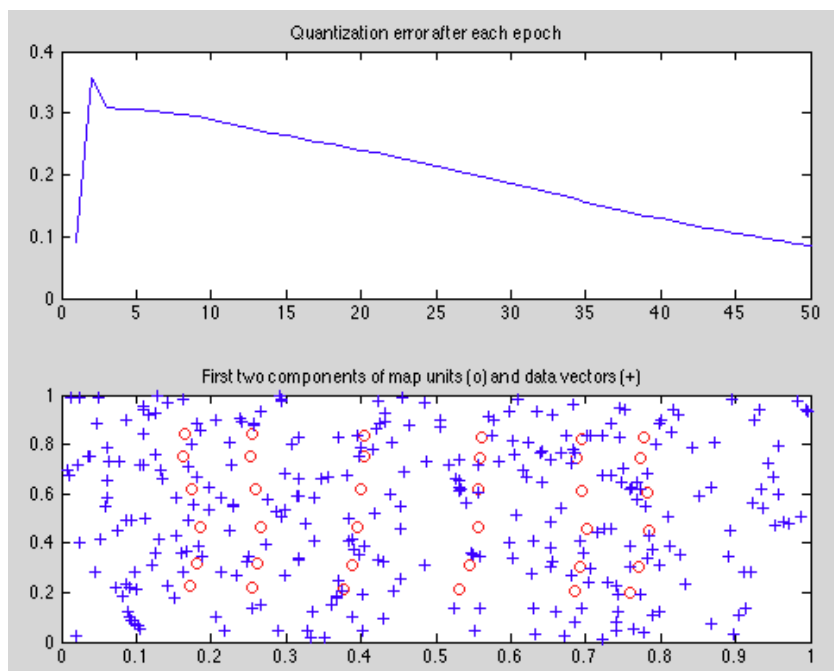
- Lorsqu'on utilisait une valeur de **T fixée**, on s'est aperçu,
    - ☞ qu'une valeur élevée de **T** favorisait la formation d'un ordre de la carte
      - ↪ mais que cette dernière ne parvenait pas à se déployer sur l'ensemble des données
    - ☞ à contrario, une petite valeur de **T** permettait le déploiement de la carte sur les données,
      - ↪ mais la carte obtenue n'était pas nécessairement bien ordonnée.
  - ⇒ Pour répondre à cet antagonisme, la procédure utilisée consiste à initialiser la température **T** à une valeur élevée, favorisant ainsi l'apparition de l'ordre, puis à la faire décroître progressivement au cours des itérations de minimisation permettant à la carte de recouvrir peu à peu la distribution réelle des observations.
  - ⇒ Les paramètres déterminants de cette méthode sont :
    - l'intervalle de variation de **T**
      - ↪ c'est-à-dire la valeur initiale de **T** (Tmax) et sa valeur finale (Tmin)
    - le nombre d'itérations de minimisation,
    - la manière dont le paramètre **T** décroît dans l'intervalle [Tmax, Tmin] au cours des itérations.
- Selon le réglage de ces paramètres, l'algorithme d'optimisation peut aboutir à des résultats différents
- L'apprentissage des cartes topologiques fait décroître le paramètre **T** dans un intervalle.
    - ⇒ La convergence vers la solution peut se décomposer en deux phases :
      - **Phase 1 (phase d'auto-organisation)** : correspond aux grandes valeurs de **T**.
        - ↪ Elle a tendance à assurer la conservation de l'ordre topologique.
      - **Phase 2 (phase de convergence)** : a lieu pour les petites valeurs de **T**.
        - ↪ l'algorithme commence à se rapprocher de l'algorithme des k-moyennes.

---

### **Entraînement de la carte : Phase 1 (Auto organisation)**

```
>> figure
>> epochs = 50;  → nombre d'itérations de la phase d'auto-organisation
>> radius_ini = 5; → valeur initiale de T (Tmax) (pour la phase d'auto-organisation)
>> radius_fin = 1; → valeur finale de T (Tmin) (pour la phase d'auto-organisation)
>> Neigh = 'gaussian'; → choix de la fonction de voisinage (voir figure 4 ci-dessus)
    ☞ Neigh = 'gaussian', 'cutgauss', 'bubble' ou 'ep'

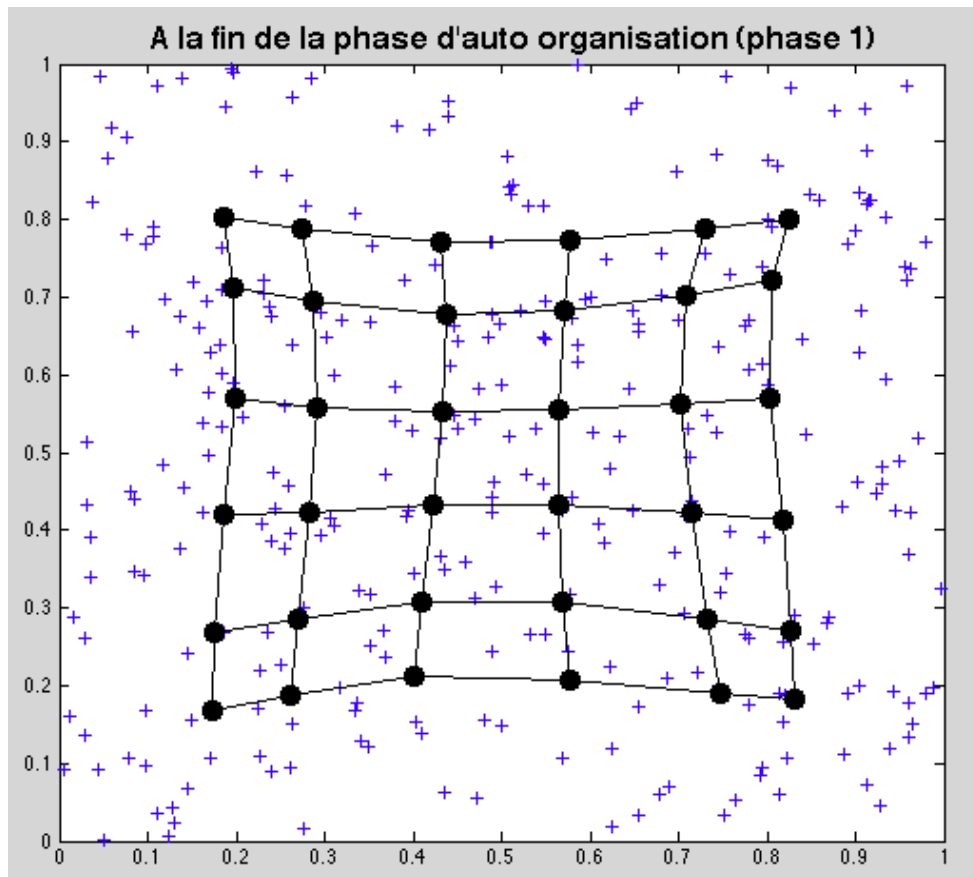
>> tr_lev = 3;
>> [sMap,sT] = som_batchtrain(sMap, sData,'trainlen',epochs, ... ← ne pas oublier ces 3 points
>> 'radius_ini',radius_ini,'radius_fin',radius_fin, 'neigh',Neigh,'tracking',tr_lev);
```





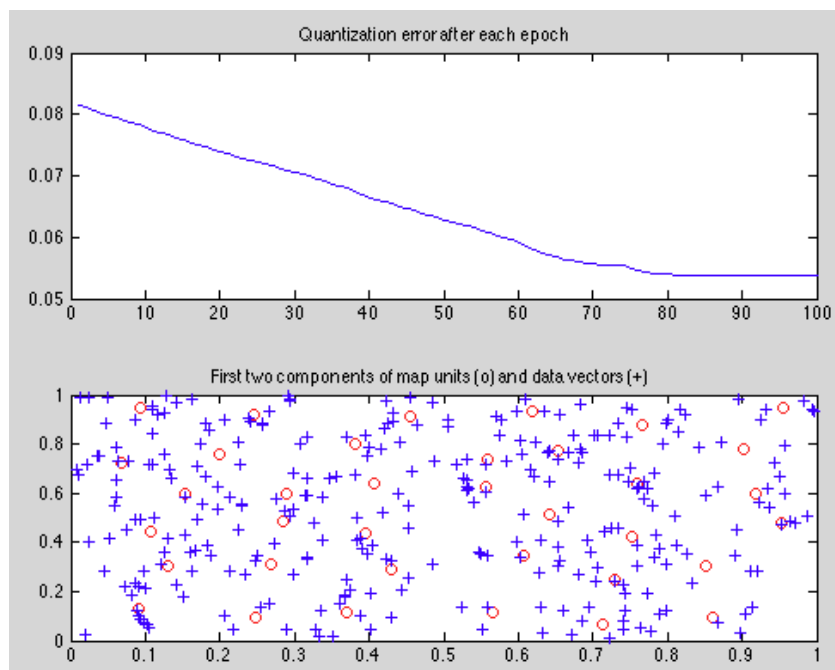
## Visualisation des données et de la carte à la fin de la phase 1

```
>> figure
>> plot(D(:,1),D(:,2),'b+')
>> hold on
>> som_grid(sMap,'Coord',sMap.codebook), hold off, axis on
>> title('A la fin de la phase d'auto organisation (phase 1)');
```



## Entraînement de la carte : Phase 2 (Convergence)

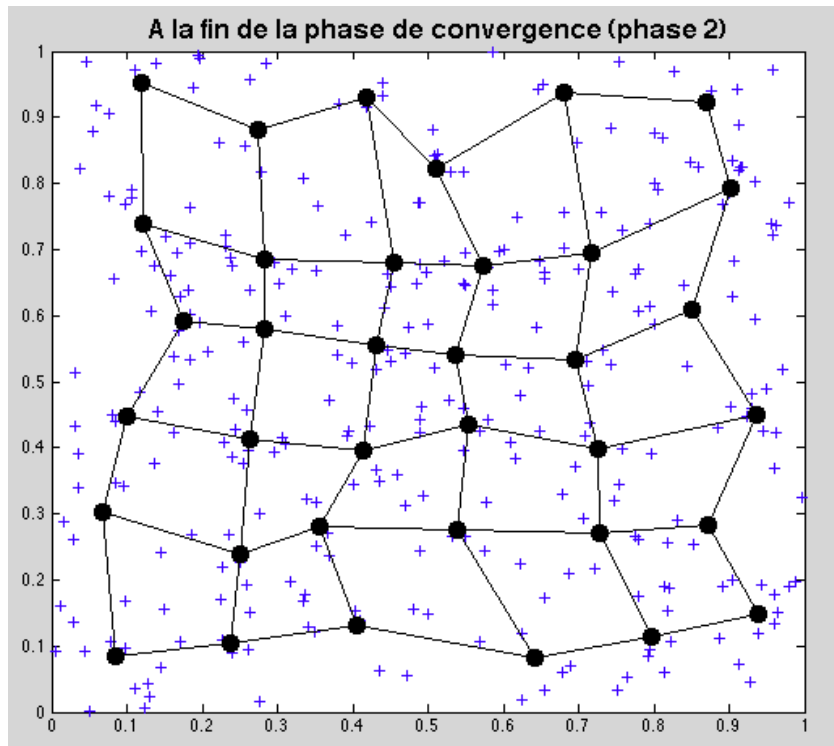
```
>> epochs = 100; radius_ini = 1; radius_fin = 0.1;
>> figure
>> [sMap,sT] = som_batchtrain(sMap, sData,'trainlen',epochs, ... ← ne pas oublier ces 3 points
>> 'radius_ini',radius_ini,'radius_fin',radius_fin, 'neigh',Neigh,'tracking',tr_lev);
```





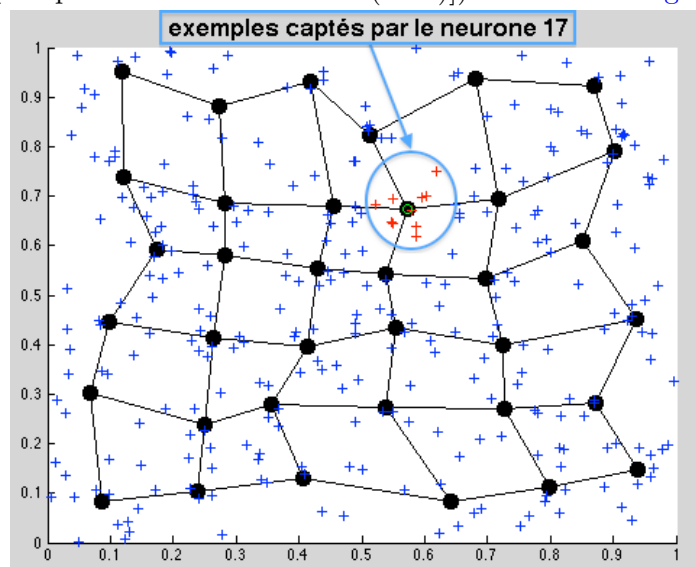
## Visualisation des données et de la carte à la fin de la phase 2

```
>> figure
>> plot(D(:,1),D(:,2),'b+')
>> hold on
>> som_grid(sMap,'Coord',sMap.codebook)
>> title('A la fin de la phase de convergence (phase 2)');
```



### • Visualisation des exemples captés par un neurone

```
>> [Bmus, Qerrors] = som_bmus(sMap, sData);
    ⇨ Taper help som_bmus pour plus de détails
    ⇨ Lire aussi page 42 de la documentation "Manual and tutorial (April 2000)"
>> neur=17; % neurone choisi
>> exemples_captés = find(Bmus == neur); % données captées par le neurone choisi
>> figure
>> som_grid(sMap,'Coord',sMap.codebook) % visualisation de la carte
>> hold on
>> plot(D(:,1),D(:,2),'b+') % données en bleu
>> plot(D(exemples_captés,1),D(exemples_captés,2),'r+') % données captées par neur en rouge
>> plot(sMap.codebook(neur,1),sMap.codebook(neur,2),'go') % neurone neur en vert
>> title(['exemples captés par le neurone ' num2str(neur)]) % titre de la figure
```



L'ordre topologique obtenu est très sensible à l'ensemble des paramètres qui interviennent dans l'algorithme. Il n'existe malheureusement pas de règle qui nous assure que l'ordre obtenu est parfaitement adéquat. Certains indicateurs de qualité sont cependant proposés comme :

- **l'erreur de quantification**

↪ elle est définie par la distance moyenne des données à leurs référents

- **l'erreur topographique** (une mesure de la préservation de la topologie)

↪ elle correspond à la proportion des données pour lesquelles les 2 référents les plus proches ne correspondent pas à des neurones adjacents sur la carte.

⇒ Ces indicateurs sont d'une interprétation délicate, et une démarche empirique de validation reste recommandée avant d'interpréter et d'utiliser les résultats définitivement.

### Qualité de la carte

```
>> [qe,te]=som_quality(sMap,sData)
```

☞ Pour plus de détails sur la commande **som\_quality**, taper :

```
>> help som_quality
```

```
>> help som_quality
SOM_QUALITY Calculate the mean quantization and topographic error.

[qe,te] = som_quality(sMap, D)

qe = som_quality(sMap,D);
[qe,te] = som_quality(sMap,sD);

Input and output arguments:
sMap      (struct) a map struct
D          (struct) a data struct
           (matrix) a data matrix, size dlen x dim

qe        (scalar) mean quantization error
te        (scalar) topographic error

The issue of SOM quality is a complicated one. Typically two
evaluation criterias are used: resolution and topology preservation.
If the dimension of the data set is higher than the dimension of the
map grid, these usually become contradictory goals.

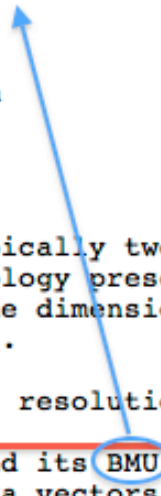
The first value returned by this function measures resolution and the
second the topology preservation.
qe : Average distance between each data vector and its BMU
te : Topographic error, the proportion of all data vectors
    for which first and second BMUs are not adjacent units.

NOTE: when calculating BMUs of data vectors, the mask of the given
      map is used.

For more help, try 'type som_quality' or check out the online documentation.
See also som\_bmus.

>>
```

**BMU : Best-Matching Unit**



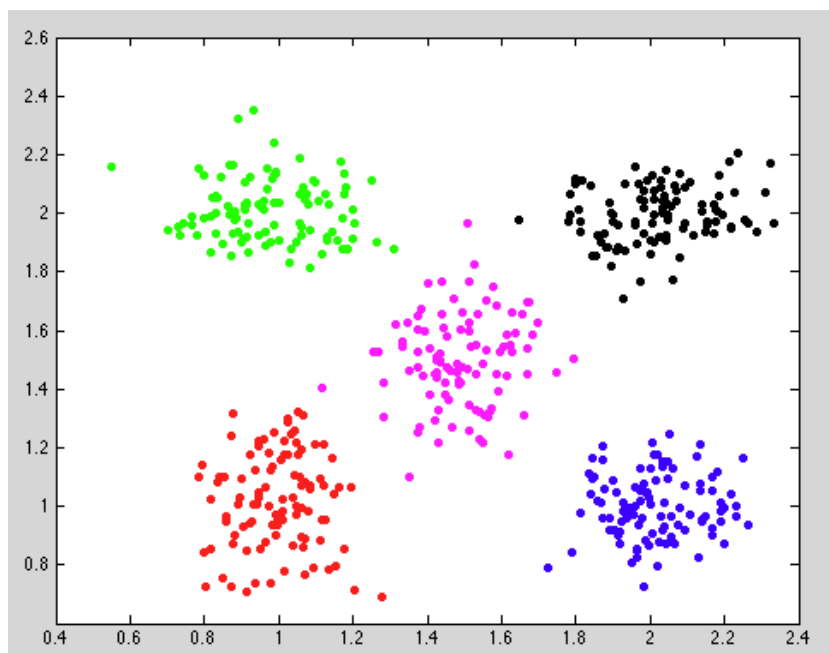
# Problème de classification

## Les données

- Charger le fichier [RCP208Data5Classes.zip](http://deptinfo.cnam.fr/new/spip.php?article851) depuis :  
<http://deptinfo.cnam.fr/new/spip.php?article851>  
Vous pouvez charger directement ce fichier en cliquant sur le lien suivant :  
<http://deptinfo.cnam.fr/new/spip.php?pdoc6299>
- Décompresser ce fichier (sous linux : `unzip RCP208Data5Classes.zip`)
  - ☞ Vous obtenez **Data2.mat** qui contient 4 variables :
    - *D* : les données
    - *classes* : les classes des données
    - *cnames* : les noms des variables
    - *labs* : les étiquettes des classes ou labels
- Visualisation des données
  - ☞ Utiliser les instructions suivantes (sous matlab ou octave):

```
load Data2.mat
i1=find(classes==1);
i2=find(classes==2);
i3=find(classes==3);
i4=find(classes==4);
i5=find(classes==5);
figure
plot(D(i1,1),D(i1,2),'r.')
hold on
plot(D(i2,1),D(i2,2),'g.')
plot(D(i3,1),D(i3,2),'b.')
plot(D(i4,1),D(i4,2),'k.')
plot(D(i5,1),D(i5,2),'m.')
```

On obtient la figure suivante :



## Apprentissage

### Initialisation de la structure de la carte topologique

```
>> addpath somtoolbox
>> sData = som_data_struct(D,'name','Donnees2','labels',labs,'comp_names',cnames);
Taper sData.comp_names pour voir son contenu
>> sData.comp_names → sData.comp_names contient cnames
Taper sData.labels pour voir son contenu
>> sData.labels → sData.labels contient labs
```

⇒ En utilisant les instructions vues dans la section précédente, entraîner une carte.

## Labellisation des neurones de la cartes en utilisant le vote majoritaire

- Comme on le sait maintenant, à l'issu de l'apprentissage,  
↪ chaque observation  $\mathbf{z}_i$  est affectée à un neurone  $c$  selon la fonction d'affectation  $\chi : c = \chi(\mathbf{z}_i)$ .
- On projette alors l'ensemble des données labellisées (étiquetées) ,  
↪ chaque neurone  $c$  en récupère une partie.
- Le neurone  $c$  est donc associé au sous-ensemble  $P_c$  constitué, pour partie, des données labélisées qui lui ont été affectées.  
Dès lors, pour déterminer la classe du neurone  $c$ , on peut utiliser un vote majoritaire  
↪ qui consiste à attribuer au neurone  $c$  la classe qui apparaît le plus souvent parmi le sous- ensemble  $P_c$ .  
(Dans cet exemple, toutes les données sont labellisées)
- Donc on peut utiliser **le vote majoritaire** pour attribuer une classe à chaque neurone  $c$   
↪ le principe étant que le neurone prendra l'étiquette de la classe la plus représentée dans son sous ensemble  $P_c$ .

Il est à noter qu'il est possible qu'un neurone n'ait capté aucune donnée (révélant ainsi des zones de l'espace mal connues)

↪ dans ce cas aucune classe n'est attribuée au neurone.

A la fin de l'apprentissage, taper les instructions suivantes une à une pour les comprendre

### Labellisation des neurones de la cartes en utilisant le vote majoritaire

```
>> Labl = cell(size(sMap.codebook,1),1);
>> Labl → pour voir le contenu de Labl
>> for ii = 1:size(sMap.codebook,1), Labl{ii} = num2str(ii); end
>> Labl → pour voir le nouveau contenu de Labl
>> sMap = som_label(sMap,'clear','all')
>> sMap.labels → pour voir son contenu
>> sMap = som_label(sMap,'add','all', Labl);
>> sMap.labels → pour voir son nouveau contenu
>> sMap = som_autolabel(sMap,sData,'vote'); → labellisation des neurones de la carte en utilisant
                                             le vote majoritaire
>> sMap.labels → pour voir son nouveau contenu
```

### Visualisation du résultat obtenu par vote majoritaire

```
>> figure
>> som_show(sMap,'empty','numéro de chaque neurone et son étiquette');
>> som_show_add('label',sMap,'subplot',1);
```

On obtient la figure ci-dessous

numéro (ou indice) du neurone	n° de chaque neurone et son label obtenu par vote majoritaire						
	① deux	8 deux	15 deux	22	29 un	36 un	43 un
	2 deux	9 deux	16 deux	23 cinq	30 un	37 un	44 un
	3 deux	10 deux	17 cinq	24 cinq	31 cinq	38 un	45 un
	4 quatre	11 cinq	18 cinq	25 cinq	32 cinq	39 cinq	46 trois
	5 quatre	12 quatre	19 cinq	26 cinq	33 cinq	40 trois	47 trois
	6 quatre	13 quatre	20 quatre	27 cinq	34 trois	41 trois	48 trois
	7 quatre	14 quatre	21 quatre	28	35 trois	42 trois	49 trois

- Dans cette figure, chaque neurone contient deux informations :
  - ↪ son numéro (son indice dans la carte) et
  - ↪ son étiquette (ou label) obtenue suite au vote majoritaire.
- Pour plus de détails sur **som\_label**, **som\_autolabel** et **som\_show**
  - ↪ utiliser la commande **help**
    - >> help som\_label
    - >> help som\_autolabel
    - >> help som\_show
  - ↪ voir aussi documentation (techrep.pdf) **Manual and tutorial (April 2000)**