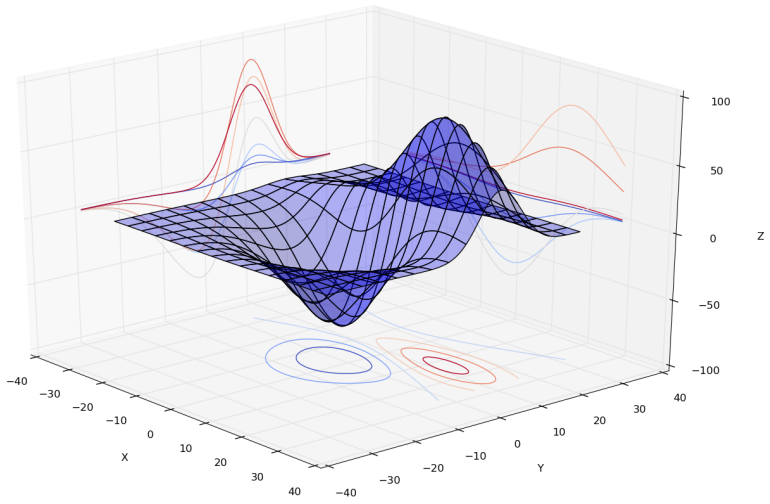
Matplotlib

Matplotlib est un module destiné à produire des graphiques de toute sorte (voir http://matplotlib.org/gallery.html pour une gallerie d'images produites avec Matplotlib). Il peut fonctionner à partir de tableau Python, mais nous l'utiliserons essentiellement à partir de tableaux Numpy. Un guide d'utilisation se trouve à l'adresse http://matplotlib.org/users/index.html. Un autre bon guide avec beaucoup d'exemples se trouve sur http://www.loria.fr/~rougier/teaching/matplotlib/



Graphe simple

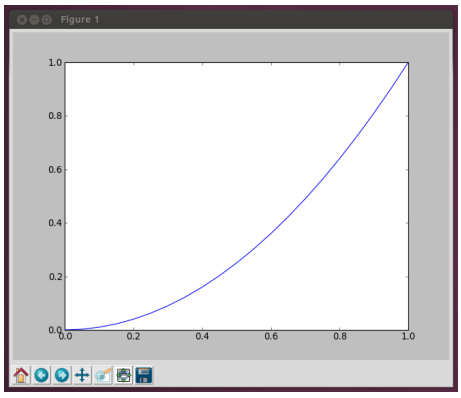
on importe le module `matplotlib.pyplot` sour le nom `plot` afin d'avoir accès aux fonctions de façon plus simple

```
>>> import matplotlib.pyplot as plt
```

la fonction la plus simple est la fonction `plot` sour la forme `plot(x,y)` où x est un tableau d'abscisses et y le tableau des ordonnées associées

```
>>> import matplotlib.pyplot as plt
>>> x=np.linspace(0,1,21)
>>> y=x*x
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0x7facbdfc2490>]
>>> plt.show()  # affiche le graphe créé
```

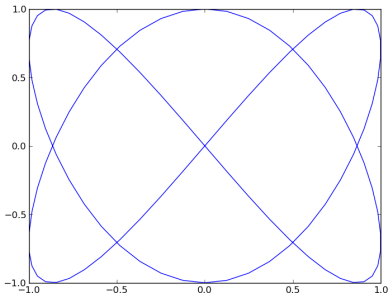
La dernière commande ouvre une fenêtre avec le graphe



De la même façon, on peut tracer une courbe paramétrée. Ici on trace la courbe définie par $f(t) = (\sin(2t), \sin(3t))$ pour $t \in [0, 2\pi]$:

```
>>> t=np.linspace(0,2*np.pi,101)# discrétisation du temps entre 0 et 2 pi.
>>> x=np.sin(2*t)       # les abscisses
>>> y=np.sin(3*t)       # les ordonnées
>>> plt.plot(x,y)       # la suite est classique
[<matplotlib.lines.Line2D object at 0x7facbd916a50>]
>>> plt.show()
```

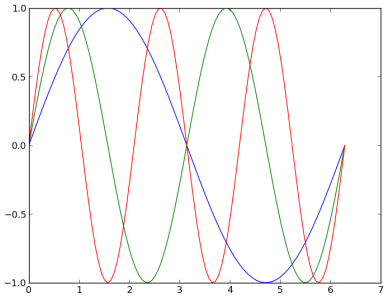
On obtient



Plusieurs courbes en même temps

Il suffit de les ajouter sur le même graphe avec `plt.plot` :

```
>>> x=np.linspace(0,2*np.pi,101)
>>> plt.plot(x,np.sin(x))
>>> plt.plot(x,np.sin(2*x))
>>> plt.plot(x,np.sin(3*x))
>>> plt.show()
```



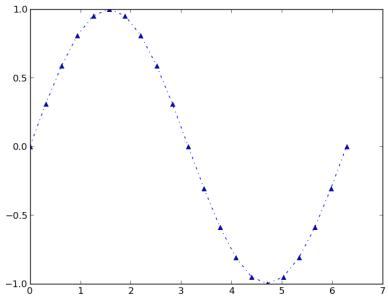
De façon plus condensée, on peut le faire en une seule instruction

```
>>> x=np.linspace(0,2*np.pi,101)
>>> plt.plot(x,np.sin(x),x,np.sin(2*x),x,np.sin(3*x))
>>> plt.show()
```

Quelques améliorations

- **Propriétés de la courbe** : lorsqu'on crée un `plot`, on peut préciser différentes options comme le style de ligne, la couleur. Pour le détail : http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot. Cela peut être fait de manière détaillée en précisant les options ou grâce à une expression condensée

```
>>> x=np.linspace(0,2*np.pi,21)
>>> plt.plot(x,np.sin(x), 'b-.^')
```



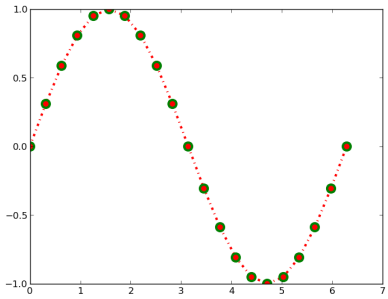
le troisième argument décrit le style de courbe

Style simplifié d'une courbe

`r,g,b,c,m,y,w,k`	couleur (k pour black)
`-  --  -.  :`	style de la ligne
`.  ,  ^  o  <  > +  x`	style des marqueurs (voir l'aide pour d'autres)

On peut également préciser les options dans une liste d'options

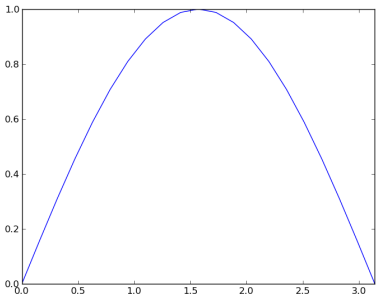
```
>>> x=np.linspace(0,2*np.pi,101)
>>> plt.plot(x,np.sin(x), color='red', linewidth=3, linestyle="-.", marker="o", markersize=10,
    markeredgecolor="g", markeredgewidth=3)
>>> plt.show()
```



On ira voir http://matplotlib.org/api/artist_api.html#matplotlib.lines.Line2D pour la description des options de style d'une ligne.

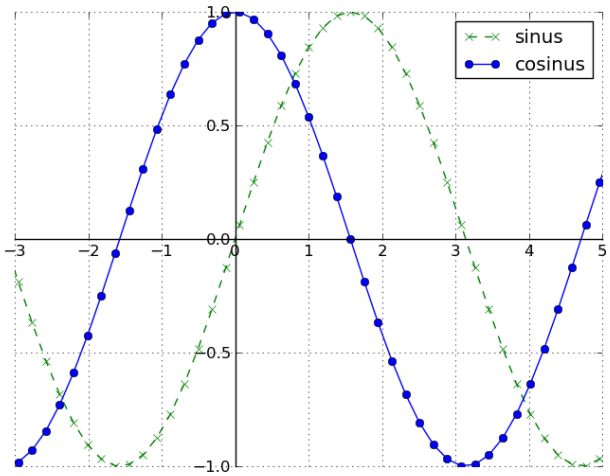
- **Axes** : les axes sont un objet créé lorsqu'on commence un graphique. On récupère cet objet avec la commande `plt.gca()` (get current axis). On peut alors modifier ses propriétés. On dispose pour commencer de la zone d'affichage qu'on peut modifier avec les commandes `xlim` et `ylim` :

```
>>> x=np.linspace(0,2*np.pi,41)
>>> plt.plot(x,np.sin(x))
>>> plt.xlim()
(0.0, 7.0)
>>> plt.xlim(0,np.pi)
(0, 3.141592653589793)
>>> plt.ylim(0,1)
(0, 1)
>>> plt.show()
```



Pour les options, il y en a beaucoup trop à détailler (aller voir sur internet si besoin). On pourra regarder l'exemple final.

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-np.pi,2*np.pi,51)
plt.plot(x,np.sin(x),color='green',label='sinus', marker='x', linestyle='--')
plt.plot(x,np.cos(x),color='blue', label='cosinus', marker='o')

axis=plt.gca()      # on récupère les axes

plt.xlim(-3,5)    # taille de la fenêtre
plt.ylim(-1,1)

axis.spines['top'].set_color('none')    # on retire l'axe supérieur
axis.spines['right'].set_color('none')  # et de droite
axis.spines['bottom'].set_position('zero')
axis.spines['left'].set_position('zero')

plt.yticks(np.linspace(-1,1,5))         # on modifie les étiquettes des ordonnées

plt.grid()
plt.legend()

plt.show()
```



Axes et axis

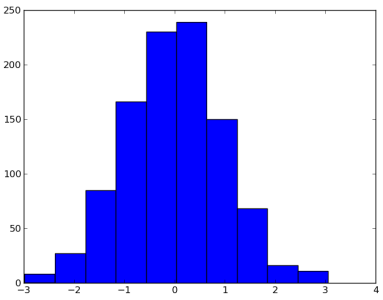
Ne pas mélanger « axis » (http://matplotlib.org/api/axis_api.html) et « axes » (http://matplotlib.org/api/axes_api.html) sous matplotlib : le premier désigne les axes, le second une fenêtre dans laquelle on travaille

(on ne rentre pas dans les détails, cela s'éloigne beaucoup de ce qu'il faut savoir). On en parlera un petit peu lors des graphiques dans l'espace

Histogrammes

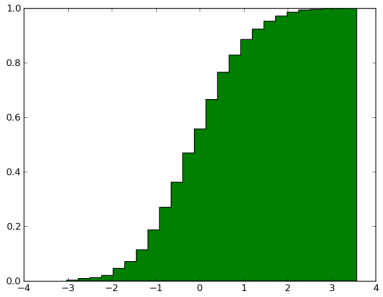
Un exemple est plus explicite

```
import numpy as np
import matplotlib.pyplot as plt
# on crée un tableau avec des valeurs aléatoires obtenues
# avec une loi normale
valeurs=np.random.randn(1000)
plt.hist(valeurs)
plt.show()
```



On peut modifier quelques options intéressantes, par exemple

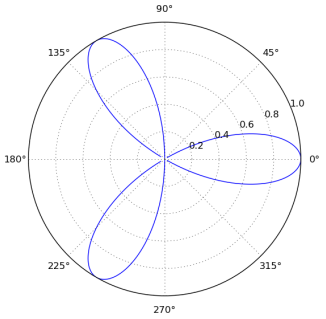
```
import numpy as np
import matplotlib.pyplot as plt
# on crée un tableau avec des valeurs aléatoires obtenues
# avec une loi normale
valeurs=np.random.randn(1000)
plt.hist(valeurs,
         25,                    # nombre de barres
         normed=True,           # données normalisées
         cumulative=True,       # histogramme cumulatif
         color='green',         # couleur
         histtype='stepfilled'  # type : bar,barstacked,step,stepfilled
         )
plt.show()
```



Graphes en coordonnées polaires

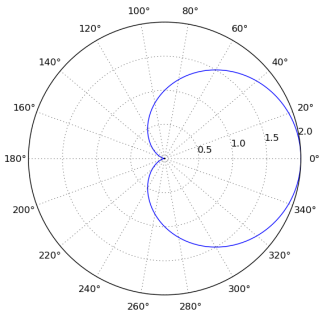
On utilise la commande `polar` :

```
theta=np.linspace(0,2*np.pi,500)
r=np.cos(3*theta)
plt.polar(theta,r)
plt.show()
```



On peut modifier quelques options

```
theta=np.linspace(0,2*np.pi,500)
r=1+np.cos(theta)
plt.polar(theta,r)
plt.thetagrids(np.arange(0,360,20))
plt.rgrids([0.5,1,1.5,2],angle=10)
plt.show()
```

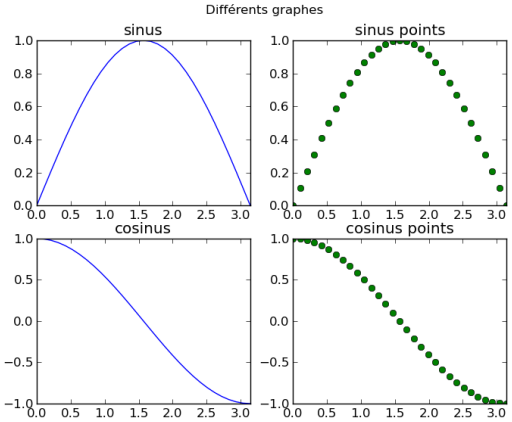


Figures et subplots

On commence une nouvelle figure avec l'instruction `figure(numero)` (le numéro n'est pas obligatoire). L'affichage avec `plt.show()` ouvrira une fenêtre pour chaque figure créée. Plus intéressant, on peut dans une même figure, créer plusieurs graphiques. On utilise pour cela la commande `subplot` sous la forme `subplot(p,q,n)` où (p,q) représente la taille du tableau des sous-graphiques et n la position dans ce tableau : si $p=2$ et $q=3$, alors on crée une grille de 6 graphiques (2 lignes et 3 colonnes) numérotés de 1 à 6.

```
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(0,np.pi,31)

plt.suptitle("Différents graphes")  # le titre général

plt.subplot(2,2,1)  # une grille 2x2 - premier graphique
plt.axis((0,np.pi,0,1))
y=np.sin(x)
plt.title('sinus')
plt.xticks([0,1,2,3])
plt.
plt.plot(x,y,"b-")
```

```
plt.subplot(2,2,2)  # second graphique
plt.axis((0,np.pi,0,1))
plt.title('sinus points')
plt.plot(x,y,"go")

plt.subplot(2,2,3)  # troisième graphique
y=np.cos(x)
plt.axis((0,np.pi,-1,1))
plt.title('cosinus')
plt.plot(x,y,"b-")

plt.subplot(2,2,4) # et le dernier
plt.axis((0,np.pi,-1,1))
plt.title('cosinus points')
plt.plot(x,y,"go")

plt.show()          # on affiche le tout
```



Contour et grille

Dans cette partie, on veut tracer les lignes de niveau d'une fonction de deux variables (ou plus généralement d'un tableau de 2 dimensions). Pour cela, on a besoin de créer un tableau en 2 dimensions avec les valeurs. On pourrait le faire avec deux boucles imbriquées mais Numpy dispose de tout ce qu'il faut pour faire les calculs vectoriellement. Admettons que l'on se situe sur le carré $[0,1]\times[0,1]$. On va illustrer sur une discrétisation régulière de pas $\frac{1}{N}$ mais on peut faire moins régulier... On peut commencer par créer

```
>>> N=50
>>> x=np.linspace(0,1,N+1)
>>> y=np.linspace(0,1,N+1)
```

Le soucis est d'évluer une fonction sur la grille complète, de façon simple. On illustre avec $f:(x,y)\mapsto 2x^2+\sin(3*y)$. On crée deux grilles de taille $(N+1)\times(N+1)$ avec les valeurs de x et y aux différents points :

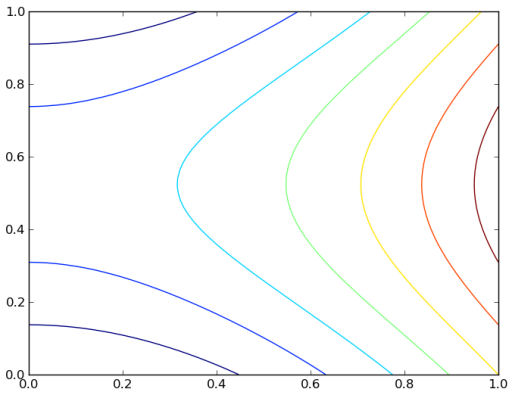
```
>>> X,Y=np.meshgrid(x,y)
>>> X
array([[ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ],
       [ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ],
       [ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ],
       ...,
       [ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ],
       [ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ],
       [ 0.  ,  0.02,  0.04, ...,  0.96,  0.98,  1.  ]])
```

On n'a plus qu'à évaluer la fonction sur cette grille

```
>>> Z=2*X*X+np.sin(3*Y)
```

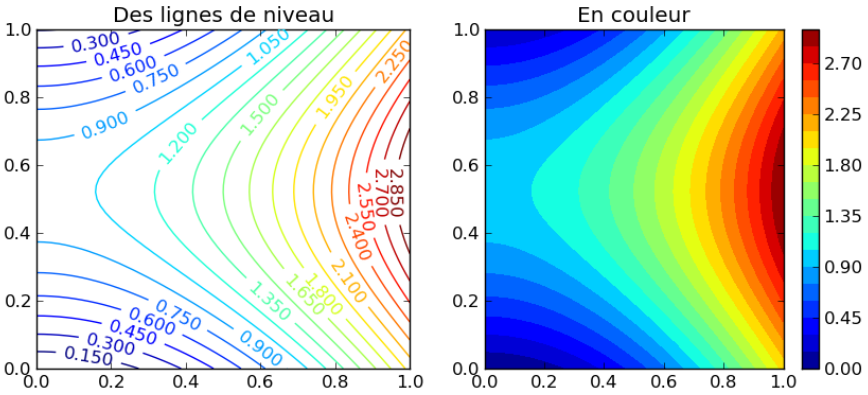
et à tracer

```
>>> plt.contour(X,Y,Z)
>>> plt.show()
```



On redonne l'exemple complet, avec quelques améliorations

```
import numpy as np
import matplotlib.pyplot as plt

plt.figure(figsize=(10,4))          # pour élargir un peu, la taille est 10*100x4*100

N=50
x=np.linspace(0,1,N+1)
y=np.linspace(0,1,N+1)

X,Y=np.meshgrid(x,y)                 # création des 2 grilles sur x et y
Z=2*X*X+np.sin(3*Y)                  # valeurs sur la grille

plt.subplot(1,2,1)                   # premier graphique
contour=plt.contour(X,Y,Z,20)        # trace le contour avec 20 lignes et le stocke dans la variable
plt.clabel(contour)                  # ajoute les étiquettes
plt.title("Des lignes de niveau")    # et un titre

plt.subplot(1,2,2)                   # deuxième graphique : on remplit
contour=plt.contourf(X,Y,Z,20)       # contour rempli
plt.colorbar()                       # on ajoute une barre latérale avec les niveaux
plt.title("En couleur")
plt.show()
```

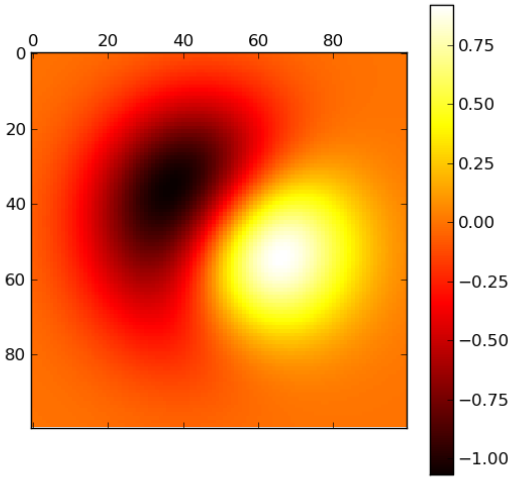


Affichage d'une matrice

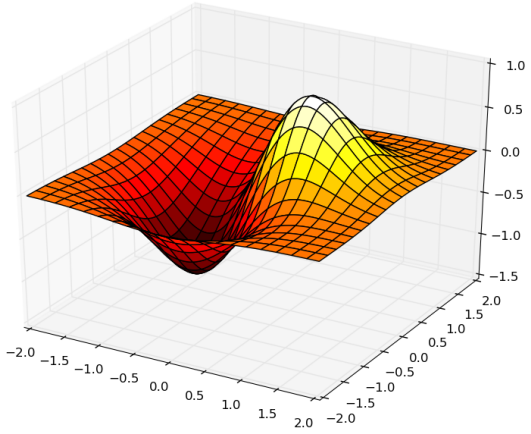
On utilise `matshow` sous la forme `imshow(matrice,options)`

```
import numpy as np
import matplotlib.pyplot as plt

def f(x,y):
    return (2*x-y**2+y)*np.exp(-x**2-y**2)

x=np.linspace(-2,2,100)      # discrétisation de [-2,2]
```

```
y=np.linspace(-2,2,100)      # idem
X,Y = np.meshgrid(x,y)       # création de la grille complète
Z=f(X,Y)                     # on évalue f sur la grille
plt.matshow(Z,cmap='hot')    # image avec les valeurs, style de couleurs 'hot'
plt.show()
```



La même vue en 3 dimensions :



On pourra aller sur http://www.loria.fr/~rougier/teaching/matplotlib/#id49 pour voir différents styles de couleurs.

Remarque : la fonction `matshow` utilise en fait une fonction plus générale, `imshow`, qui sert à afficher des images matricielles. Elle l'appelle et ajoute quelques options.

Résumé

Quelques fonctions de matplotlib.pyplot

`plot(x,y,options)`	tracé d'une graphique
`fill(x,y,options)`	avec remplissage
`hist(...)`	histogramme
`scatter(x,y,options)`	nuage de points
`contour(...), contourf(...)`	lignes de niveau, contour (plus compliqué)
`bar(x,y,width)`	diagramme en barre
`imshow(tableau,options)`	affiche une image (matricielle)
`legend()`	affichage de la légende
`grid()`	affichage de la grille
`gca()`	système d'axes
`show()`	affichage du graphique
`savefig(nom)`	sauvegarde le graphique dans le fichier `nom` (s'adapte à l'extension)
`xlabel(texte), ylabel(texte)`	étiquette abscisses et ordonnées
`text(x,y,texte)`	place un texte sur le graphique
`title(texte)`	titre du graphique
`xticks(pos,etiq), yticks(...)`	points sur l'axe des abscisses : `pos` : tableau des positions et `etiq` : tableau des étiquettes
`figure(numero)`	commence une nouvelle figure
`subplot(p,q,n)`	nouveau subplot en position n sur une grille $p \times q$
`axis(xmin,xmax,ymin,ymax)`	zone d'affichage (sous-graphiques)

Graphiques dans l'espace

Bientôt...