6. Graphiques, images, animations

6.1 Concepts de base

Les fonctionnalités décrites dans cette page web se réfèrent aux logiciels/versions suivants :

- **MATLAB 7**, avec son moteur de graphiques intégré
- **GNU Octave-Forge 3.4.2**, avec les backends **Gnuplot 4.4** (backend traditionnel) et OpenGL/**FLTK** (qui fait son apparition avec Octave 3.4)

Nous ne décrivons plus ici le backend Octave **JHandles**, celui-ci n'étant plus supporté depuis Octave 3.4.

L'**aide en ligne** relative aux fonctions de réalisation de graphiques s'obtient, de façon classique, en frappant `help fonction_graphique` (Ex: `help plot`). En outre :

- sous **MATLAB**: les commandes `help graph2d`, `help graph3d` et `help specgraph` affichent la liste des fonctions graphiques disponibles
- sous **Octave**: on se réfèrera au Manuel Octave (HTML ou PDF) au chapitre "Plotting", ou via la commande `doc fonction_graphique`

Pour une **comparaison** des possibilités graphiques entre Octave/FLTK, Octave/Gnuplot et MATLAB, voyez cette intéressante page : **http://octave.sourceforge.net/compare_plots/**

6.1.1 Notion de "backends graphiques" sous Octave

MATLAB, de par sa nature commerciale, est monolithique et intègre son propre moteur d'affichage de graphiques.

GNU Octave est conçu de façon modulaire (voir chapitre "**Packages Octave-Forge**") et s'appuie également sur des logiciels externes. C'est ainsi que le logiciel libre de visualisation **Gnuplot** a longtemps été utilisé par Octave comme générateur de graphiques standard. Ce n'est que depuis la version 3.4 (en 2011) que Octave intègre son propre moteur graphique basé OpenGL/**FLTK**, ce qui n'empêche pas l'utilisateur de recourir à d'autres "backends" graphiques.

Parmi les autres projets de couplage (bindings) avec des grapheurs existants, ou de développement de backends graphiques propres à Octave, on peut citer :

- **Octaviz** : 3D, assez complet (wrapper donnant accès aux classes **VTK**, Visualization ToolKit) (voir article **FI-EPFL 5/07**)
- **OctPlot** : 2D (ultérieurement 3D ?)
- **epsTK** : fonctions spécifiques pour graphiques 2D très sophistiqués (était intégré à la distribution Octave-Forge 2.1.42 Windows)

Quant aux anciens projets suivants, ils sont (ou semblent) arrêtés : **JHandles** (packetage Octave-Forge, développement interrompu depuis 2010), **Yapso** (Yet Another Plotting System for Octave, 2D et 3D, basé OpenGL), **PLplot** (2D et 3D), **Oplot++** (2D et 3D, seulement sous Linux et MacOSX), **KMatplot** (2D et 3D, ancien, nécessitant Qt/KDE), **KNewPlot** (2D et 3D, ancien, nécessitant Qt et OpenGL), **Grace** (2D).

6.1.2 Les backends OpenGL/FLTK et Gnuplot sous GNU Octave-Forge 3.4

La version **3.4** constitue une avancée majeure de **Octave** avec l'arrivée d'un moteur graphique spécifique et l'implémentation avancée du mécanisme MATLAB des "handles graphics". Nous avons ainsi actuellement le choix entre deux backends principaux :

a. le backend traditionnel **Gnuplot** : logiciel de visualisation libre développé indépendemment de Octave, à l'origine essentiellement orienté tracé de courbes 2D et de surfaces 3D en mode "filaire". Devenu capable, depuis la version 4.2, de remplir des surfaces colorées, cela a permis, depuis Octave 3, l'implémentation de fonctions graphiques 2D/3D classiques MATLAB (fill, pie, bar, surf...). Les "handles graphics" ont commencé à être implémentés avec Gnuplot depuis Octave 2.9 !
b. le nouveau backend basé sur **OpenGL** et **FLTK** (Fast Light Toolkit), qui offre davantage de performances et permet davantage d'interactivité

Choix du backend graphique depuis Octave-Forge 3.4 :

Pour basculer d'un backend à l'autre, il faut commencer par fermer les éventuelles fenêtres de graphiques ouvertes avec `close('all')`, puis :

- pour passer de FLTK à **Gnuplot**, passer la commande: `graphics_toolkit('gnuplot')`
- pour passer de Gnuplot à **FLTK**, passer la commande: `graphics_toolkit('fltk')`

En outre la commande `available graphics_toolkits` montre quels sont les backends disponibles.

6.1.3 Fenêtres de graphiques

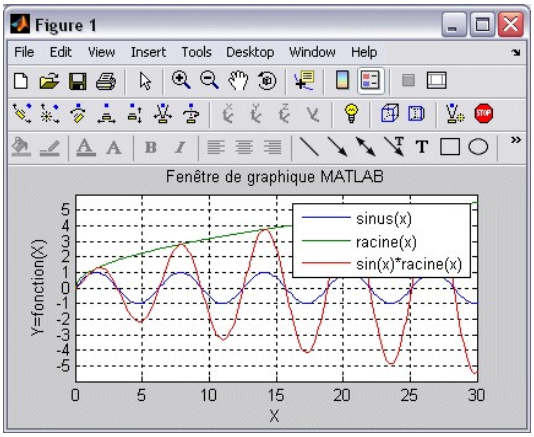
Les graphiques MATLAB/Octave sont élaborés dans des **fenêtres de graphiques** spécifiques appelées "**figures**". Celles-ci apparaissent lorsqu'on fait usage des commandes `figure`, `subplot`, ou automatiquement lors de toute commande produisant un tracé (graphique 2D ou 3D).

De façon analogue au workspace avec la commande `save`, **MATLAB** permet de **sauvegarder une figure** en tant qu'**objet** avec la commande `saveas(handle,'fichier','fig')`, par exemple en vue de la récupérer et la compléter dans une session MATLAB ultérieure... Il n'y a pas d'équivalence sous Octave.

On présente ci-dessous l'aspect et les fonctionnalités des fenêtres graphiques correspondant aux différentes versions de backends. Le code qui a été utilisé pour produire les illustrations est le suivant :

```
x=0:0.1:10*pi;
y1=sin(x); y2=sqrt(x); y3=sin(x).*sqrt(x);
plot(x,y1,x,y2,x,y3);
grid('on');
axis([0 30 -6 6]);
set(gca,'Xtick',0:5:30);  set(gca,'Ytick',-5:1:5);
title('Fenêtre de graphique MATLAB / FLTK / Gnuplot');
xlabel('X'); ylabel('Y=fonction(X)');
legend('sinus(x)','racine(x)','sin(x)*racine(x)');
```

Fenêtre graphique MATLAB 7

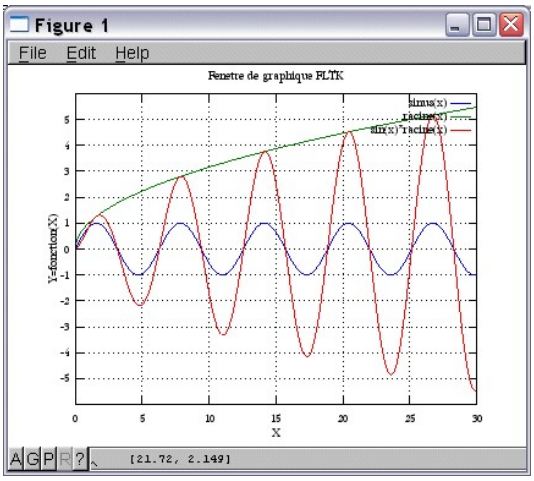


Les caractéristiques principales des fenêtres de graphiques **MATLAB** sont :

- Une **barre de menus** comportant notamment :
  - `Edit>Copy Figure` : copie de la figure dans le presse-papier (pour la "coller" ensuite dans un autre document) ; voyez `Edit>Copy Options` qui permet notamment d'indiquer si vous prenez l'image au format vecteur (défaut => bonne qualité, redimensionnable...) ou raster, background coloré ou transparent...
  - `Tools>Edit Plot`, ou commande `plotedit`, ou bouton-curseur [Edit Plot] de la barre d'outils : permet de sélectionner les différents objets du graphique (courbes, axes, textes...) et, en double-cliquant dessus ou via les articles du menu `Tools`, d'éditer leurs propriétés (couleur, épaisseur/type de trait, symbole, graduation/sens des axes...)
  - `File>Save as` : exportation du graphique sous forme de fichier en différents formats raster (JPEG, TIFF, PNG, BMP...) ou vecteur (EPS...)
  - `File>Page/Print Setup`, `File>Print Preview`, `File>Print` : mise en page, prévisualisation et impression d'un graphique (lorsque vous ne le "collez" pas dans un autre document)
  - Affichage de palettes d'outils supplémentaires avec `View>Plot Edit Toolbar` et `View>Camera Toolbar`
  - `View>Property Editor`, ou dans le menu `Edit` les articles `Figure Properties`, `Axes Properties`, `Current Object Properties` et `Colormap`, puis le bouton [Inspector] (ou commande `propedit`) : pour modifier de façon très fine les propriétés d'un graphique (via ses handles...)
  - Ajout/dessin d'objets depuis le menu `Insert`
  - Un menu `Camera` apparaît lorsque l'on passe la commande `cameramenu`
- La **barre d'outils** principale, comportant notamment :
  - bouton-curseur [Edit Plot] décrit plus haut
  - boutons-loupes [+] et [-] (équivalents à `Tools>Zoom In|Out`) pour zoomer/dézoomer interactivement dans le graphique ; voir aussi les commandes `zoom on` (puis cliquer-glisser, puis `zoom off`), `zoom out` et `zoom(facteur)`
  - bouton [Rotate 3D] (équivalent à `Tools>Rotate 3D`) permettant de faire des **rotations 3D**, par un cliquer-glisser avec le bouton <gauche>, y compris sur des graphiques 2D !
  - boutons [Insert Colorbar] (équivalent à la commande `colorbar`) et [Insert Legend] (équivalent à la commande `legend`)
  - boutons [Show|Hide Plot Tools] (ou voir menu `View`) affichant/masquant des sous-fenêtres de dialogues supplémentaires (Figure Palette, Plot Browser, Property Editor)

Fenêtre graphique OpenGL/FLTK (depuis Octave 3.4)

Les caractéristiques principales des fenêtres de graphiques **FLTK** sous Octave sont :

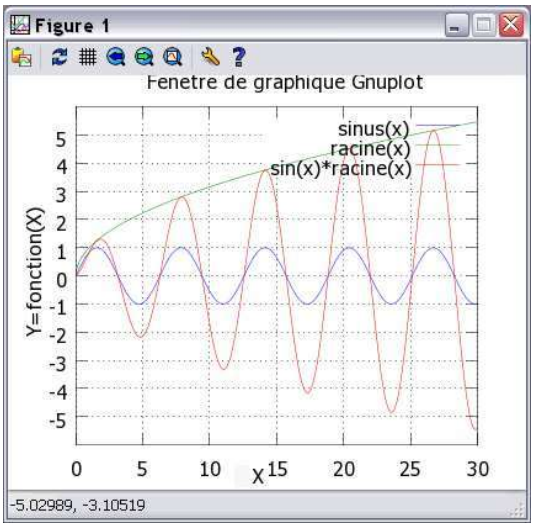


- Une **barre de menus** comportant :
  - `File>Save {as}` : **sauvegarder** la figure sur un fichier de type (selon l'extension spécifiée):
    • vectorisé: PDF, PS (PostScript)
    • raster: GIF, PNG, JPG
  - `File>Close` : fermer la fenêtre de figure (identique à la case de fermeture `[X]` ou à la commande `close`)
  - `Edit>Grid` : bascule d'activation/désactivation de l'affichage de la **grille** (équivalent à la commande `grid('on|off')`)
  - `Edit>Autoscale` : se remet en mode "**autoscaling**", c-à-d. ajustement dynamique des limites inférieures et supérieures des axes X, Y pour afficher l'intégralité des données (équivalent à la commande `axis('auto')`)
  - `Edit>GUI Mode>Pan+Zoom` ou bouton `[P]` : dans des fenêtres de graphiques 3D, le bouton <gauche> de la souris fera du "pan" (déplacement)
  - `Edit>GUI Mode>Rotate+Zoom` ou bouton `[R]` : dans des fenêtres de graphiques 3D, le bouton <gauche> de la souris fera du "rotate"
  - `Edit>GUI Mode>None` : désactive l'usage du bouton <gauche> de la souris
- Les **boutons** de la souris et la **barre d'outils** en bas à gauche s'utilisent ainsi :
  - <gauche>-glisser :
    • graphiques 2D : **pan** (déplacement horizontal et/ou vertical)
    • graphiques 3D : **pan** ou **rotate**, selon le mode défini plus haut
  - <droite>-glisser : faire un **rectangle-zoom**
  - <roulette>-tourner : faire un **zoom** avant/arrière
  - <double-clic-gauche> ou <milieu> ou clavier `<a>` ou bouton `[A]` : **autoscaling**
  - clavier `<g>` ou bouton `[G]` : bascule d'affichage/masquage de la **grille**
  - bouton `[?]` : affichage aide sur les raccourcis clavier et l'usage de la souris

- Il est en outre possible de compléter cette fenêtre par des **menus personnalisés** (articles et raccourcis associés à fonctions callback...) à l'aide de la fonction `uimenu`.

Fenêtre graphique Gnuplot 4.4 sous Octave 3.2.4 MinGW

ATTENTION: au cas où la fenêtre Gnuplot sous Windows ne réagirait plus (impossible de la déplacer, curseur en "sablier"...), vous pouvez la "réveiller" en passant simplement la commande `refresh` (ou `grid`, bascule d'activation/désactivation grille). Ce bug (qui existe depuis Octave 3.0) ne se produit que dans certaines configuration de Windows.



Les caractéristiques principales de la fenêtres de graphiques **Gnuplot 4.4** sous Octave sont :

- Une **barre d'outils** (pour autant qu'elle aie été activée avec la commande `putenv('GNUTERM','wxt')`) comportant :
  - bouton [Copy the plot to clipboard] : **copie** la figure dans le "presse-papier" (pour pouvoir la "coller" ensuite dans un autre document)
  - bouton [Replot] : raffraîchit l'affichage du graphique
  - bouton [Toggle grid] : affichage/masquage de la **grille** (bascule) (équivalent à la commande `grid('on|off')`)
  - boutons [Apply the previous/next zoom settings] : pour fenêtre **2D** seulement, revient au facteur de zoom précédent/suivant
  - bouton [Apply autoscale] : pour fenêtre **2D** seulement, se remet en mode "**autoscaling**", c-à-d. ajustement dynamique des limites inférieures et supérieures des axes X, Y pour afficher l'intégralité des données (équivalent à la commande `axis('auto')`)
  - bouton [Open configuration dialog] : accès aux préférences Gnuplot :
    nous vous conseillons de désactiver l'option "put the window at the top of your desktop after each plot", sinon il faut remettre la fenêtre de commande Octave au premier plan après chaque commande de graphique
  - bouton [Open help dialog] : informations d'aide

- En outre :
  - dans l'angle inférieur gauche : s'affichent, en temps réel :

        - fenêtre **2D**: les **coordonnées X/Y** précises du **curseur**, que vous pouvez inscrire dans le graphique en cliquant avec `<milieu>`
        - fenêtre **3D**: l'orientation de la vue (angle d'**élévation** par rapport au nadir, et **azimuth**) et les facteurs d'**échelle** en X/Y et en Z
    - par des cliquer-glisser avec la souris :
        - fenêtre **2D**: **zooms interactifs** précis avec <droite> glisser <droite> (outre la commande `axis` présentée plus bas) ; pour faire un **zoom out**, on passera la commande `axis('auto')`
        - fenêtre **3D**:
            - <gauche>-glisser : **rotation 3D**
            - <milieu>-mvmt horizontal : **zoom** avant/arrière (utiliser <ctrl> pour graphiques complexes)
            - <milieu>-mvmt vertical : changement **échelle en Z** (utiliser <ctrl> pour graphiques complexes)
            - <maj-milieu>-mvmt vertical : changement **origine Z** (utiliser <ctrl> pour graphiques complexes)
    - en mode terminal XWT (avec barre d'icônes), Gnuplot n'a plus de sous-menu Options dans le menu contextuel de la barre de titre

- Quelques fonctionnalités plus avancées de cette fenêtre graphique Gnuplot :
    - Octave active automatiquement le "**mode souris**" de Gnuplot ; on peut aussi faire cela manuellement en frappant `m` (bascule d'activation/désactivation) dans la fenêtre graphique Gnuplot
    - d'autres **raccourci-clavier** sont possibles dans la fenêtre graphique Gnuplot (la liste de ceux-ci apparaît dans la fenêtre de commande Octave lorsque vous frappez `h` dans la fenêtre graphique), notamment :
        - `g` : affichage/masquage de la **g**rille (bascule)
        - `l` ☒ (pas possible sous Gnuplot 4.4) : axe Y (2D) ou Z (3D) **l**ogarithmique/linéaire (bascule)
        - `L` ☒ (pas possible sous Gnuplot 4.4) : axe se trouvant le plus proche du curseur **l**ogarithmique/linéaire (bascule)
        - `b` : affichage/masquage d'une **b**ox dans les graphiques **3D** (bascule)
        - `a` : pour fenêtre **2D** seulement, **a**utoscaling des axes (utile après un zoom !)
        - `7` : pour fenêtre **2D** seulement, même échelle (ratio) pour les axes X et Y (bascule)
        - `p` et `n` : pour fenêtre **2D** seulement, facteur de zoom **p**récédent, respectivement suivant (**n**ext)
        - `u` : pour fenêtre **2D** seulement, dé-zoomer (**u**nzoom)
        - `e` : r**e**plot

Pour mémoire, suivre **ce lien** pour accéder aux informations relatives aux anciennes versions de : • Gnuplot 3.x à 4.0 embarqué dans Octave-Forge 2.x Windows, • Gnuplot 4.2.2/4.3 embarqué dans Octave 3.0.1/3.0.3 MSVC

6.1.4 Axes, échelle, quadrillage, légende, titre, annotations

Les fonctions décrites dans ce chapitre doivent être **utilisées après** qu'**une fonction de dessin** de graphique ait été passée (et non avant). Elles agissent immédiatement sur le graphique courant.

Fonction et description	
Exemple	**Illustration**

Lorsque l'on trace un graphique, MATLAB/Octave détermine automatiquement les **limites inférieures et supérieurs** des axes X, Y {et Z} sur la base des valeurs qui sont graphées, de façon que le tracé occupe toute la fenêtre graphique (en hauteur et largeur). Les **rapports d'échelle** des axes sont donc différents les uns des autres. Les commandes `axis` et `xlim` / `ylim` / `zlim` permettent de modifier ces réglages.

a) `axis([Xmin Xmax Ymin Ymax { Zmin Zmax }])`
b) `axis('auto')`
c) `axis('manual')`
d) *lim_xyz* = `axis`

Modification des valeurs limites (sans que l'un des "aspect ratio" `equal` ou `square`, qui aurait été activé, soit annulé) :
a) recadre le graphique en utilisant les valeurs spécifiées des limites inférieures/supérieures des axes X, Y {et Z}, réalisant ainsi un **zoom** avant/arrière dans le graphique ; sous MATLAB il est possible de définir les valeurs `-inf` et `inf` pour faire déterminer les valeurs min et max (équivalent de `auto`)
b) se remet en mode "autoscaling", c-à-d. définit dynamiquement les limites inférieures/supérieurs des axes X, Y {et Z} pour faire apparaître l'intégralité des données ; sous MATLAB on peut spécifier '`auto x`' ou '`auto y`' pour n'agir que sur un axe
c) verrouille les limites d'axes courantes de façon que les graphiques subséquents (en mode `hold on`) ne les modifient pas lorsque les plages de valeurs changent
d) passée sans paramètre, la fonction `axis` retourne le vecteur-ligne *lim_xyz* contenant les limites [*Xmin Xmax Ymin Ymax* { *Zmin Zmax* }]

a) `xlim([Xmin Xmax])` , `ylim([Ymin Ymax])` , `zlim([Zmin Zmax])`
b) `xlim('auto')` , `ylim('auto')` , `zlim('auto')`
d) `xlim('manual')` , `ylim('manual')` , `zlim('manual')`
d) *lim_x* = `xlim` , *lim_y* = `ylim` , *lim_z* = `zlim`

Même fonctionnement que la fonction `axis`, sauf que l'on n'agit ici que **sur 1 axe** à la fois

a) `axis('equal')` ou `axis('image')` ou `axis('tight')`
b) `axis('square')`
c) `axis('normal')`
d) `axis('vis3d')`

Modification des rapports d'échelle ("aspect ratio") (sans que les limites inf. et sup. des axes X, Y {et Z} soient affectées) :
a) définit le même rapport d'échelle pour les axes X et Y
b) définit les rapports d'échelle en X et Y de façon la zone graphée soit carrée
c) annule l'effet des "aspect ratio" `equal` ou `square`
d) sous MATLAB, bloque le rapport d'échelle pour rotation 3D

a) *ratio* = `daspect()`
b) `daspect(ratio)`
c) `daspect('auto')`

Rapport d'échelle entre les axes X-Y{-Z} (data aspect ratio) (voir aussi la commande `pbaspect` relatif au "plot box")
a) récupère, sur le vecteur *ratio* (3 éléments), le rapport d'échelle courant entre les axes du graphique
c) modifie le rapport d'échelle entre les axes selon le vecteur *ratio* spécifié
b) le rapport d'échelle est mis en mode automatique, s'adaptant dès lors à la dimension de la fenêtre de graphique

a) `axis('off | on')`
b) `axis('nolabel | labelx | labely | labelz')`
c) `axis('ticx | ticy | ticz')`

Désactivation/réactivation affichage cadre/axes/graduation, quadrillage et labels :
a) désactive/rétablit l'affichage du cadre/axes/graduation et quadrillage du graphique ; sous MATLAB (mais pas Octave) agit en outre également sur les étiquettes des axes (labels)
b) désactive l'affichage des graduations des axes (ticks), respectivement rétablit cet affichage de façon différenciée en x, y et/ou z
c) active l'affichage des graduations des axes (ticks) et du quadrillage (grid) de façon différenciée en x, y et/ou z

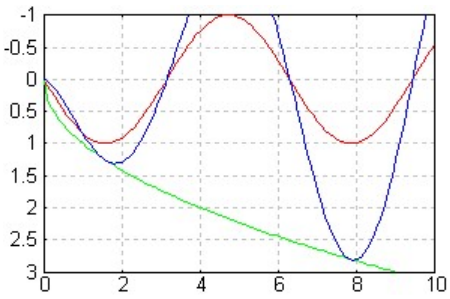
a) `axis('xy')`
b) `axis('ij')`

Inversion du sens de l'axe Y :
a) origine en bas à gauche, valeurs Y croissant de bas en haut (par défaut)
b) origine en haut à gauche, valeurs Y croissant de haut en bas

Ex 1: (graphique ci-contre réalisé avec Octave/Gnuplot)

Ne vous attardez pas sur la syntaxe de la commande plot qui sera décrite plus loin au chapitre "Graphiques 2D"

```
x=0:0.1:10*pi;
y1=sin(x); y2=sqrt(x); y3=sin(x).*sqrt(x);
plot(x,y1,x,y2,x,y3);
legend('off');
grid('on');
axis([0 10 -1 3]);  % changement limites (zoom)
axis('ij');         % inversion de l'axe Y
```



a) **`set(gca,'Xtick | Ytick | Ztick', [`***valeurs***`] )`**
b) **`set(gca,'XTickLabel | YTickLabel | ZTickLabel',`** *labels***`)`**
c) **`tics('x|y|z',`** *valeurs* **`{,`** *labels***`})`**

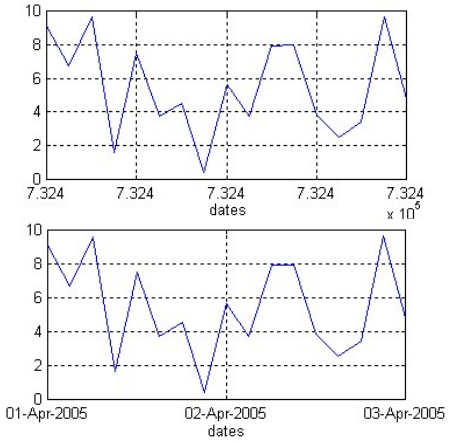
Graduation des axes et lignes de grille :
a) et b) Commandes basées sur la technique des "Handle Graphics". On utilise ici la fonction **`gca`** (get current axes) qui retourne le "handle" du graphique courant
a) Spécifie les *valeurs* (suite de valeurs en ordre coissant), sur l'*axe* indiqué, auxquelles il faut : dessiner un 'tick' sur l'axe, afficher la valeur, et faire partir une ligne de grille
b) Spécifie le texte à afficher (label) en regard de chaque tick. Le paramètre *labels* peut être un vecteur de nombres, une matrice de chaînes , un tableau cellulaire de chaînes. Commande particulièrement utile si l'on veut graduer l'axe avec des chaînes (p.ex. des dates formatées...). **Important** : le nombre de *valeurs* et de *labels* doit être identique !
c) Propre à Octave (implémentée dans package "plot"), cette fonction permet de redéfinir très simplement la graduation et les labels (sans passer par les "Handle Graphics") :
- le premier paramètre définit l'axe sur lequel on veut agir (x, y ou z)
- le vecteur *valeurs* (ligne ou colonne) définit les emplacements des 'ticks' et départ de lignes de grille
- et *labels* est ici un vecteur cellulaire (ligne ou colonne) de chaînes contenant les textes à afficher à coté de chaque tick (en lieu et place des valeurs)
En ne passant que le permier paramètre (x, y ou z), la fonction restaure la graduation par défaut

Voir aussi la fonction **`datetick`** pour graduer/formater les axes temporels

Ex 2: (graphique ci-contre réalisé avec MATLAB ou Octave)

Ne vous attardez pas sur la syntaxe de la commande plot qui sera décrite plus loin au chapitre "Graphiques 2D"

```
date_debut = datenum('01-Apr-2005');
date_fin   = date_debut + 2;  % 2 jours plus tard

x=date_debut:0.125:date_fin;  % série toutes 3 h.
y=10*rand(1,length(x));

plot(x,y);
grid('on');   % => premier graphique ci-contre

xlabel('dates');

x_tick=date_debut:1:date_fin; % tous 1 jours (24 h)
set(gca,'Xtick',x_tick)
set(gca,'XTickLabel', ...
    datestr(x_tick,'dd-mmm-yyyy'))
          % => second graphique ci-contre
```



a) **`grid('on | off')`** ou **`grid on | off`**
b) **`grid`**

a) Activation/désactivation de l'affichage du **quadrillage** (grid). Par défaut le quadrillage d'un nouveau graphique n'est pas affiché.
b) Sans paramètre, cette fonction agit comme une bascule on/off.

Ex: voir l'exemple 1 ci-dessus

`box('on|off')`
Activation/désactivation de l'affichage, autour du graphique, d'un **cadre** (graphiques 2D) ou d'une "**boîte**" (graphiques 3D).
Sans paramètre, cette fonction agit comme une bascule on/off.

`zoom('on | xon | yon | off | out')`
`zoom(`*facteur***`)`**
Pour **zoomer** d'un *facteur* donné dans la figure courante, globalement, en X ou Y...

`xlabel('`*label_x***`')`** , **`ylabel('`***label_y***`')`** , **`zlabel('`***label_z***`')`**
Définit et affiche le texte de **légende des axes** X, Y et Z (étiquettes, labels). Par défaut les axes d'un nouveau graphique n'ont pas de labels.

Ex: voir l'exemple 3 ci-dessous

a) `legend('legende_t1 ','legende_t2'... {,pos} )`
b) `legend('off')`

a) Définit et place une **légende** sur le graphique en utilisant les textes spécifiés pour les tracés *t1*, *t2*... La position de la légende est définie par le paramètre `pos` : **0**= Automatic (le moins de conflit avec tracés), **1**= angle haut/droite, **2**= haut/gauche, **3**= bas/gauche, **4**=bas/droite, **-1**= en dehors à droite de la zone graphée. Avec MATLAB, la légende peut ensuite être déplacée interactivement à l'aide de la souris.
b) Désactive l'affichage de la légende

Ex: voir l'exemple 3 ci-dessous

`title('titre')`

Définit un **titre de graphique** qui est placé au-dessus de la zone graphée. Un nouveau graphique n'a par défaut pas de titre. Pour effacer le titre, définir une chaîne *titre* vide.

Ex: voir l'exemple 3 ci-dessous

a) `text(x, y, { z,} 'chaîne' {,'propriété','valeur'...} )`
b) `gtext('chaîne' {,'propriété','valeur'...} )`

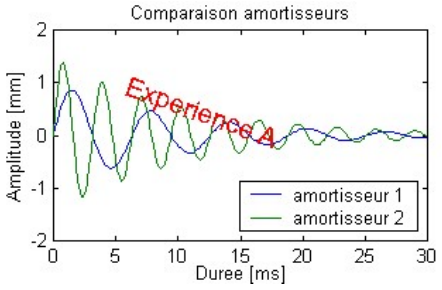
a) Définit l'**annotation** *chaîne* qui est placés sur le graphique aux coordonnées *x*, *y* {,*z*} spécifiées. Des attributs (police, taille, couleur, orientation...) peuvent être spécifiés par des couples *propriété*/*valeur* (voir exemple ci-dessous et aide en ligne). Lorsqu'on utilise plusieurs fois cette fonctions, cela ajoute à chaque fois un nouveau texte.
b) L'emplacement du texte dans le graphique est défini interactivement à l'aide de la souris.
F ☒ La propriété `Rotation` n'est pas encore implémentée sous Octave 3.4.2/FLTK

Ex: voir l'exemple 3 ci-dessous

Ex 3: (graphique ci-contre réalisé avec MATLAB ou Octave)

Ne vous attardez pas sur la syntaxe de la commande plot qui sera décrite plus loin au chapitre "Graphiques 2D"

```
x=linspace(0,30,200);
y1=sin(x)./exp(x/10); y2=1.5*sin(2*x)./exp(x/10);
plot(x,y1,x,y2);
xlabel('Duree [ms]'); ylabel('Amplitude [mm]');
title('Comparaison amortisseurs');
legend('amortisseur 1','amortisseur 2',4);
text(6,1,'Experience A', ...
        'FontSize',14,'Rotation',-20, ...
        'Color','red');
```



M `texlabel('expression')`

Convertit au format TeX l'*expression* spécifiée. Cette fonction est généralement utilisée comme argument dans les commandes `title`, `xlabel`, `ylabel`, `zlabel`, et `text` pour afficher du texte incorporant des indices, exposants, caractères grecs...

Ex: M `text(15,0.8,texlabel('alpha*sin(sqrt(x^2 + y^2))/sqrt(x^2 + y^2)'));` affiche:
$\alpha \sin(\sqrt{x^2 + y^2})/\sqrt{x^2 + y^2}$

a) `whitebg()`
b) `whitebg(couleur)`
c) `whitebg('none')`

Change la **couleur de fond** du graphique :
a) Inversion du schéma de couleur, agissant comme une bascule
b) Le fond est mis à la *couleur* spécifiée sous forme de nom (p.ex. `'yellow'`) ou de triplet RGB (p.ex. `[0.95 0.95 0.1]`)
c) Rétablit le schéma de couleur par défaut

6.1.5 Graphiques superposés, côte-à-côte, ou fenêtres graphiques multiples

Par défaut, MATLAB/Octave envoie tous les ordres graphiques à la **même fenêtre** graphique (appelée "figure"), et chaque fois que l'on dessine un **nouveau graphique** celui-ci **écrase le graphique précédent**. Si l'on désire tracer **plusieurs graphiques**, MATLAB/Octave offrent les possibilités suivantes :

A. **Superposition** de plusieurs tracés de type analogue dans le même graphique en utilisant le même système d'axes (*overlay plots*)
B. Tracer les différents graphiques **côte-à-côte**, dans la même fenêtre mais dans des axes distincts (*multiple plots*)
C. Utiliser des **fenêtres distinctes** pour chacun des graphiques (*multiple windows*)

A) Superposition de graphiques dans le même système d'axes ("overlay plots")

Fonction et description

Exemple | **Illustration**

a) `hold('on')` ou `hold on`
b) `hold('off')` ou `hold off`

a) Cette commande indique à MATLAB/Octave d'accumuler (**superposer**) les ordres de dessin qui suivent dans la même figure (pour empêcher qu'un nouveau tracé efface le précédent). Elle peut être passée avant tout tracé ou après le premier ordre de dessin. Dans les modes "multiple plots" ou "multiple windows" (voir plus bas), l'état on/off de hold est mémorisé indépendemment pour chaque sous-graphique, resp. chaque fenêtre de figure
b) Après cette commande, MATLAB/Octave est remis dans le mode par défaut, c'est-à-dire que tout nouveau graphique effacera le précédent. En outre, les annotations et attributs de graphique précédemment définis (labels x/y/z, titre, légende, état on/off de la grille...) sont bien évidemment effacés.

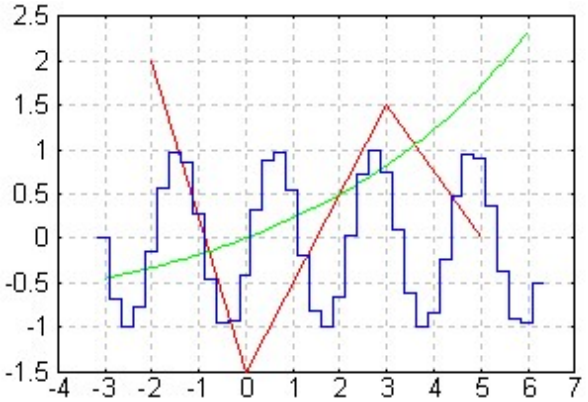
Remarque: les 2 primitives de base de tracé de lignes `line` et de surfaces remplies `patch` (présentées plus bas) permettent de dessiner par "accumulation" dans un graphique sans que `hold` soit à `on` !

`ishold`
Retourne l'état courant du mode hold pour la figure active ou le sous-graphique actif : `0` (false) si hold est off, `1` (true) si hold est on.

Ex: (graphique ci-contre réalisé avec MATLAB ou Octave)

Ne vous attardez pas sur la syntaxe des commandes plot, fplot et stairs qui seront décrites plus loin au chapitre "Graphiques 2D"

```
x1=[-2 0 3 5]; y1=[2 -1.5 1.5 0];
plot(x1,y1,'r');                % rouge
hold('on');
fplot('exp(x/5)-1',[-3 6],'g');  % vert
x3=-pi:0.25:2*pi; y3=sin(3*x3);
stairs(x3,y3,'b');              % bleu
grid('on');
```



Vous constaterez que :
• on superpose des graphiques de types différents (plot, fplot, stairs...)
• ces graphiques ont, en X, des plages et des nombres de valeurs différentes

B) Graphiques côte-à-côte dans la même fenêtre ("multiple plots")

Fonction et description

Exemple | **Illustration**

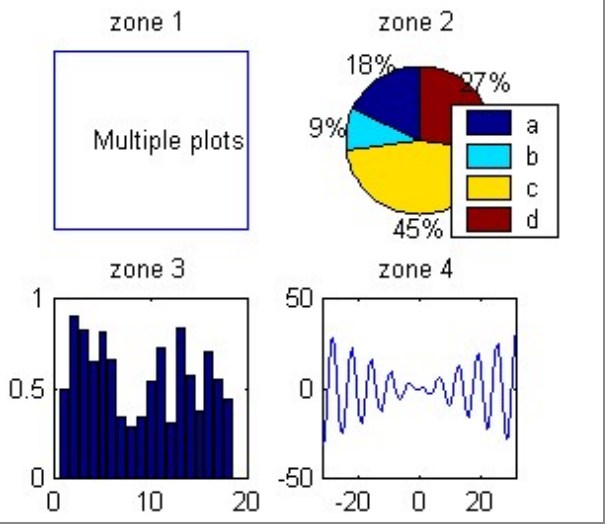
`subplot(L,C,i)`
Découpe la fenêtre graphique courante (créée ou sélectionnée par la commande `figure(numero)`, dans le cas où l'on fait du "multiple windows") en `L` lignes et `C` colonnes, c'est-à-dire en *L* x *C* espaces qui disposeront chacun leur propre système d'axes (mini graphiques). **Sélectionne** en outre la `i`-ème zone (celles-ci étant numérotées ligne après ligne) comme espace de tracé courant.
• Si aucune fenêtre graphique n'existe, cette fonction en ouvre automatiquement une
• Si l'on a déjà une fenêtre graphique simple (i.e. avec 1 graphique occupant tout l'espace), le graphique sera effacé !
• Dans une fenêtre donnée, une fois le "partitionnement" effectué (par la 1ère commande `subplot`), on ne

devrait plus changer les valeurs *L* et *C* lors des appels subséquents à `subplot`, faute de quoi on risque d'écraser certains sous-graphiques déjà réalisés !

Ex: (graphique ci-contre réalisé avec MATLAB ou Octave)

Ne vous attardez pas sur la syntaxe des commandes plot, pie, bar et fplot qui seront décrites plus loin au chapitre "Graphiques 2D"

```
subplot(2,2,1);
plot([0 1 1 0 0],[0 0 1 1 0]);
text(0.2,0.5,'Multiple plots');
axis('off'); legend('off'); title('zone 1');
subplot(2,2,2);
pie([2 1 5 3]); legend('a','b','c','d');
title('zone 2');
subplot(2,2,3);
bar(rand(18,1)); title('zone 3');
subplot(2,2,4);
fplot('x*cos(x)',[-10*pi 10*pi]);
title('zone 4');
```



C) Graphiques multiples dans des fenêtres distinctes ("multiple windows")

Fonction et description
a) `figure` b) `figure(`*numero*`)` **a)** Ouvre une **nouvelle fenêtre** de graphique (figure), et en fait la fenêtre de tracé active (dans laquelle on peut ensuite faire du "single plot" ou du "multiple plots"). Ces fenêtres sont automatiquement numérotées 1, 2, 3... **b)** Si la fenêtre de *numero* spécifié existe, en fait la **fenêtre de tracé active**. Si elle n'existe pas, ouvre une nouvelle fenêtre de graphique portant ce *numero*.
`gcf` (***g**et **c**urrent **f**igure*) Retourne le *numero* de la fenêtre de graphique active (qui correspond, dans ce cas là, au *handle* de la figure)

6.1.6 Autres commandes de manipulation de fenêtres graphiques ("figures")

Fonction et description
`refresh` ou `refresh(`*numero*`)` **Raffraîchit** (redessine) le(s) graphique(s) dans la fenêtre de figure courante, respectivement la fenêtre de *numéro* spécifié
`clf` ou `clf(`*numero*`)` (***cl**ear **f**igure*) **Efface** le(s) graphique(s) dans la fenêtre de figure courante, respectivement la fenêtre de *numéro* spécifié Remet en outre `hold` à **off** s'il était à **on**, mais conserve la table de couleurs courante. `cla` (***cl**ear **a**xis*) Dans le cas d'une fenêtre de graphique en mode "multiple plots", cette commande n'efface que le sous-graphique courant.
a) `close` b) `close(`*numero*`)` c) `close all` **a)** Referme la fenêtre graphique active (figure courante) **b)** Referme la fenêtre graphique de *numero* spécifié **c)** Referme toutes les fenêtre graphique ! Met `hold` à **off** s'il n'y a plus de fenêtre graphique
`shg` (***sh**ow **g**raphic*) Fait passer la fenêtre de figure MATLAB courante **au premier plan**. ☒ Cette commande est sans effet avec Octave sous Windows.

6.1.7 Traits, symboles et couleurs de base par 'linespec'

Plusieurs types de graphiques présentés plus bas utilisent une syntaxe, initialement définie par MATLAB et maintenant aussi reprise par Octave 3, pour spécifier le **type**, la **couleur** et l'**épaisseur** ou **dimension** de **trait** et de **symbole**. Il s'agit du paramètre `linespec` qui est une combinaison des caractères définis dans le tableau ci-dessous (voir M **`help linespec`**).

Le symbole M indique que la spécification n'est valable que pour **MATLAB**, le symbole G indique qu'elle n'est valable que pour Octave/**Gnuplot**, le symbole F indique qu'elle n'est valable que pour Octave/**FLTK** ; sinon c'est valable pour les 3 grapheurs/backends !

Il est possible d'utiliser la fonction **`[L,C,M,err]=colstyle('linespec')`** pour tester un `linespec` et le décoder sur 3 variables séparées `L` (type de ligne), `C` (couleur) et `M` (marker). Si `linespec` est erroné, une erreur `err` est retournée.

Pour un rappel sur l'ancienne façon de spécifier les propriétés de lignes sous Octave 2.x, suivre **ce lien**.

Couleur ligne et/ou symbole

Caractère	Effet
`y`	jaune (yellow)
`m`	magenta
`c`	cyan
`r`	rouge (red)
`g`	vert clair (green)
`b`	bleu (blue)
`w`	blanc (white)
`k`	noir (black)

Type de ligne

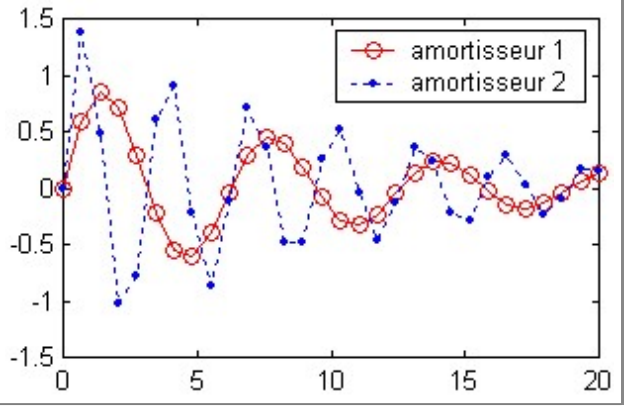
Caractère	Effet
(rien)	affichage d'une ligne continue, sauf si un symbole est spécifié (auquel cas le symbole est affiché et pas la ligne)
`-`	ligne continue
M F `--`	ligne traitillée
M F `:`	ligne pointillée
M F `-.`	ligne trait-point

Symbole (marker)

Caractère	Effet
(rien)	pas de symbole
`o`	cercle
`*`	étoile de type astérisque
`+`	signe plus
`x`	croix oblique (signe fois)
`.` (point)	M petit disque rempli F G symbole point
M F `^` `<` `>` `v` G `<` `v` G `>` `^`	M F triangle (orienté selon symbole) G triangle pointé vers le bas vide/rempli G triangle pointé vers le haut vide/rempli
`s`	carré vide (square) (G rempli)
`d`	losange vide (diamond) (G rempli)
`p`	M F étoile à 5 branches (pentagram) G carré vide
`h`	M F étoile à 6 branches (hexagram) G losange vide

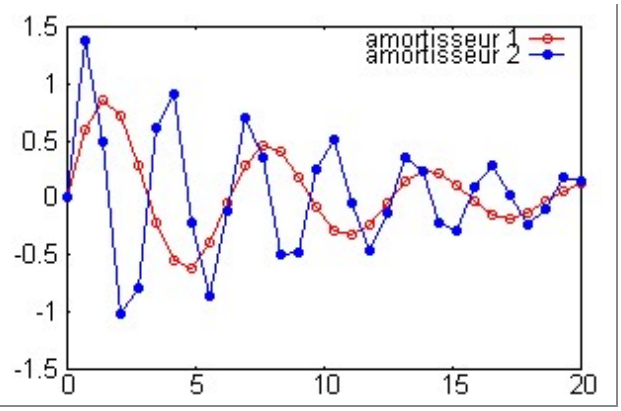
Ci-dessous, exemples d'utilisation de ces spécifications `linespec`.

Exemple	Illustration

Ex 1 : sous **MATLAB** ou **Octave/FLTK**
`x=linspace(0,20,30);`
`y1=sin(x)./exp(x/10); y2=1.5*sin(2*x)./exp(x/10);`
`plot(x,y1, 'r-o' ,x,y2, 'b:.' );`
`legend('amortisseur 1','amortisseur 2');`
	

Ex 2: sous **Octave/Gnuplot**

```
% même code que ci-dessus
```



On verra plus loin (chapitre 3D "Vraies couleurs, tables de couleurs et couleurs indexées") qu'il est possible d'utiliser beaucoup plus de couleurs en spécifiant des "**vraies couleurs**" sous forme de triplets RGB (valeurs d'intensités `[red green blue]` de 0.0 à 1.0), ou en travaillant en mode "**couleurs indexées**" via une "**table de couleurs**" (colormap). Les couleurs ainsi spécifiées peuvent être utilisées avec la propriété `'color'` de la commande `set` (voir chapitre qui suit), commande qui permet de définir également plus librement l'épaisseur et le type de **trait**, ainsi que le type de **symbole** et sa dimension.

6.1.8 Traits, symboles et couleurs spécifiques via les 'handles graphics'

Sous **MATLAB**, et sous Octave également avec le backend **FLTK** (donc depuis **Octave 3.4**), on peut aussi **modifier les attributs** de lignes et symboles avec la technique des "**Handle Graphics**" très sommairement décrite ci-dessous (développée plus en détail dans un chapitre indépendant).

Fonction et description	
Exemple	**Illustration**

`handle = fonction_graphique(...)`
Trace le graphique correspondant à la *fonction_graphique* spécifiée, et **mémorise** son *handle* (qui, selon le type de graphique, sera un scalaire ou un vecteur de handles)

a) *handle* = `gcf`
b) *handle* = `gca`
Au cas où l'on n'aurait pas mémorisé le *handle* lors du tracé de la fonction (voir ci-dessus), on peut l'obtenir après coup :
a) récupère le handle de la figure active (qui correspond, dans ce cas là, au numéro de figure)
b) récupère le handle du graphique courant dans la figure active

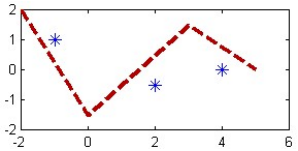
a) `get(handle)`
b) *structure* = `get(handle)`
c) *var* = `get(handle, 'PropertyName')`
a) Affiche les valeurs courantes des différentes propriétés de l'objet spécifié par son *handle*
b) Récupère sur une *structure* les valeurs courantes des différentes propriétés de l'objet spécifié par son *handle*
c) Récupère sur *var* la valeur (*PropertyValue*) courante correspondant à la propriété *PropertyName* spécifiée

Il est important de noter que dans la liste des propriétés on peut trouver des handles (de type "enfants" ou "parent") ! On est donc en présence d'une arborescence de handles (qui pourraient être assimilés à des "branches") et de propriétés ("feuilles" au bout de ces branches).

`set(handle, 'PropertyName', PropertyValue, ...)`
Modifications des propriétés d'un objet :
Pour l'objet désigné par *handle*, modifie la propriété *PropertyName* à la valeur spécifiée *PropertyValue*. Les objets MATLAB et Octave (depuis Octave 2.9/3) sont organisés en hiérarchie, et leurs propriétés sont extrêmement nombreuses... ce qui rend la technique basée sur les Handle Graphics assez complexe mais offre des possibilité d'**édition** et **animation** très nombreuses !
L'exemple ci-dessous illustre un cas simple d'utilisation des handles pour changer de type de trait et de symbole.

Ex: (graphique ci-contre réalisé avec MATLAB ou Octave/FLTK ;
sous Octave/Gnuplot, la seule différence est qu'on a une ligne continue)

```
x1=[-2 0 3 5]; y1=[2 -1.5 1.5 0];
x2=[-1 2 4];   y2=[1 -0.5 0];

 h1= plot(x1,y1);      % dessin trait, mémoris. handle
hold('on');
 h2= plot(x2,y2,'o'); % dessin symboles, mémoris. handle
 set(h1,'linewidth',3,'color',[0.7 0 0],'linestyle','--');
```



```
   % ci-dessus, spécification d'une "vraie couleur"
set(h2,'marker','*','markersize',10);
```

Mais on pourrait aussi, sans utiliser directement les "handles", passer directement les propriétés et valeurs en paramètres lors de l'appel à la fonction graphique (... mais toutes les fonctions n'acceptent pas cette façon de faire !) :

```
plot(x1,y1, 'linewidth',3,'color',[0.7 0 0],'linestyle','--' );
hold('on');
plot(x2,y2, 'marker','*','markersize',10,'linestyle','none' );
```

6.1.9 Interaction souris avec une fenêtre graphique

Il est possible d'interagir entre MATLAB/Octave et un graphique à l'aide de la souris.

On a déjà vu plus haut la fonction `gtext('chaîne')` qui permet de **placer interactivement** (à l'aide de la souris) une **chaîne** de caractère dans un graphique.

`[x, y {,bouton}] = ginput(n)`

Attend que l'on clique *n* fois dans le graphique à l'aide de la souris, et retourne les vecteurs-colonne des **coordonnées** *x* et *y* des endroits où l'on a cliqué, et facultativement le numéro de *bouton* de la souris qui a été actionné (1 pour <gauche>, 2 pour <milieu>, 3 pour <droite>).
Si l'on omet le paramètre *n*, cette fonction attend jusqu'à ce que l'on frappe <enter> dans la figure.
Remarque: sous Octave, fonction implémentée dans le package "plot"
Sous Octave 3.4 avec FLTK, les boutons 2 et 3 ne semblent pas interprétés

6.2 Graphiques 2D

Sous **MATLAB**, la liste des fonctions relatives aux graphiques 2D est accessible via `help graph2d` et `help specgraph`. Concernant **Octave/Gnuplot**, on se réfèrera au chapitre "Plotting" du Manuel Octave (HTML ou PDF).

6.2.1 Dessin de graphiques 2D

Fonction et description	
Exemple	**Illustration**

a) `plot(x1, y1 {,linespec} {, x2, y2 {,linespec} ...} )`
`plot(x1, y1 {,'PropertyName',PropertyValue} ... )`
b) `plot(vect)`
c) `plot(mat)`
d) `plot(var1,var2 ...)`

Graphique 2D de lignes et/ou semis de points sur axes linéaires :
a) Dans cette forme (la plus courante), *xi* et *yi* sont des **vecteurs** (ligne ou colonne), et le graphique comportera autant de courbes indépendantes que de paires *xi*/*yi*. Pour une paire donnée, les vecteurs *xi* et *yi* doivent avoir le même nombre d'éléments (qui peut cependant être différent du nombre d'éléments d'une autre paire). Il s'agit d'un 'vrai graphique X/Y' (graduation de l'axe X selon les valeurs fournie par l'utilisateur).

Avec la seconde forme, définition de propriétés du graphiques plus spécifiques (voir l'exemple parlant du chapitre précédent !)

b) Lorsqu'une seule variable *vect* (de type **vecteur**) est définie pour la courbe, les valeurs *vect* sont graphées en Y, et c'est l'indice de chaque valeur qui est utilisé en X (1, 2, 3 ... n). Ce n'est donc plus un 'vrai graphique X/Y' mais un graphique dont les points sont uniformément répartis selon X.
c) Lorsqu'une seule variable *mat* (de type **matrice**) est passée, chaque **colonne** de *mat* fera l'objet d'une courbe, et chacune des courbes s'appuiera en X sur les valeurs 1, 2, 3 ... n (ce ne sera donc pas non plus un 'vrai graphique X/Y')
d) Lorsque l'on passe des paires de valeurs de type **vecteur/matrice**, **matrice/vecteur** ou **matrice/matrice** :

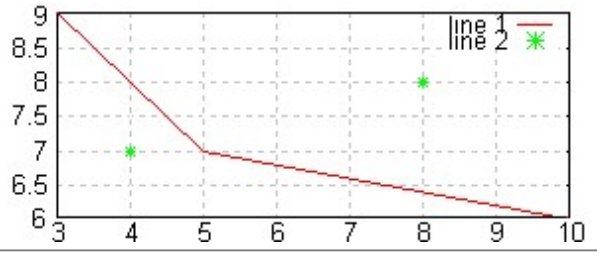
- si *var1* est un vecteur (ligne ou colonne) et *var2* une matrice :
  - si le nombre d'éléments de *var1* correspond au nombre de colonnes de la matrice *var2*, chaque **ligne** de *var2* fera l'objet d'une courbe, et chaque courbe utilisera le vecteur *var1* en X
  - si le nombre d'éléments de *var1* correspond au nombre de lignes de la matrice, chaque **colonne** de *var2* fera l'objet d'une courbe, et chaque courbe utilisera le vecteur *var1* en X
  - sinon, erreur !
- nous ne décrivons pas les autres cas (*var1* est une matrice et *var2* un vecteur, ou tous deux sont une matrice) qui sont très rares

Voir en outre (plus bas dans ce support de cours) :
- pour des graphiques à 2 axes Y : fonction `plotyy`
- pour des graphiques avec axes logarithmiques : les fonctions `semilogx`, `semilogy` et `loglog`
- pour des graphiques en semis de point avec différenciation de symboles sur chaque point : fonction `scatter`
- pour tracer des courbes 2D/3D dans un fichier au format AutoCAD DXF : fonction `dxfwrite`

Ex 1: selon forme a) ci-dessus

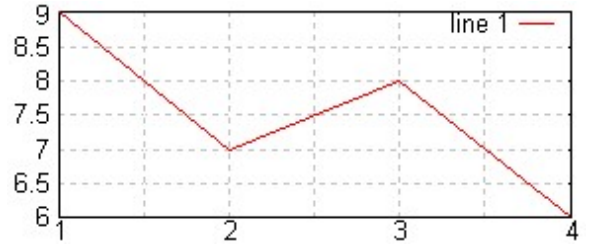
```
plot([3 5 6 10], [9 7 NaN 6], ...
     [4 8], [7 8], 'g*')
```

Remarque importante : lorsque l'on a des valeurs manquantes, on utilise `NaN`



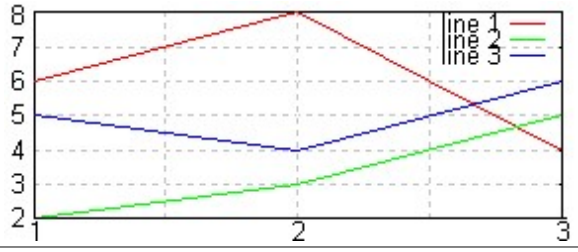
Ex 2: selon forme b) ci-dessus

```
plot([9 ; 7 ; 8 ; 6]);
```



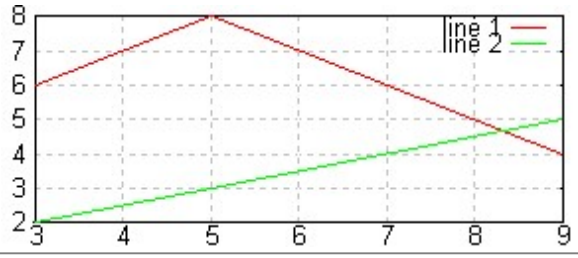
Ex 3: selon forme c) ci-dessus

```
plot([6 2 5 ; 8 3 4 ; 4 5 6]);
```



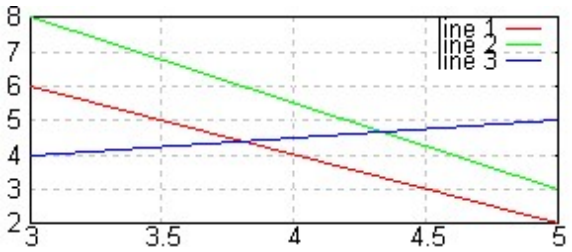
Ex 4.1: selon forme d)1 ci-dessus

```
plot([3 5 9], [6 8 4 ; 2 3 5]);
```



Ex 4.2: selon forme d)2 ci-dessus

```
plot([3 5], [6 8 4 ; 2 3 5]);
```



a) `fplot('fonction', [xmin xmax] {, nb_points } {, linespec } )` (***f**unction **plot**)*
b) `fplot('[fonction1, fonction2, fonction3 ...]', [xmin xmax] ...)`

Graphique 2D de fonctions y=fct(x) :
a) Trace la *fonction* fct(x) spécifiée entre les limites *xmin* et *xmax*.
b) Trace simultanément les différentes *fonctions* spécifiées (remarquez bien la notation entre crochets)
Par rapport à `plot`, il n'y a dans ce cas pas besoin d'échantillonner les valeurs x et y de la fonction (i.e. définition d'un vecteur x puis du vecteur y=fct(x) ...), car `fplot` accepte en argument les 2 méthodes de définition de *fonction* suivantes :

- chaîne de caractère exprimant une **fonction de x** (voir **Ex 1**)
- nom d'une **fonction MATLAB/Octave** existante (voir **Ex 2.1**),
  ou nom d'une **fonction utilisateur** (définie sous forme de M-file, voir chapitre **fonctions**) (voir **Ex 2.2**)

Le paramètre optionnel *linespec* permet de spécifier un type particulier de lignes et/ou symboles.
O Sous **Octave**, la fonction est échantillonnée (de façon interne) par défaut sur 100 points, ou sur le nombre *nb_points* spécifiés
M Sous **MATLAB**, la fonction est échantillonnée (de façon interne) par défaut sur un nombre de points qui varie selon la fonction et l'intervalle ; l'usage de *nb_points*, en-dessous d'une certaine valeur, n'a pas d'effet.

Voir encore la fonction `ezplot` (*easy plot*) qui permet de dessiner une fonction 2D définie sous sa **forme paramétrique**. A titre d'exemple, voyez la fonction `ezplot3` plus bas.

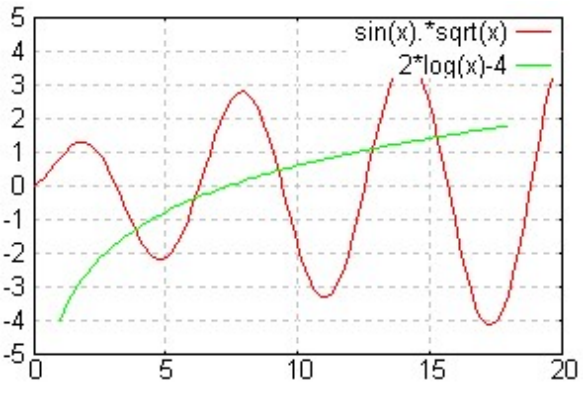
Ex 1:

```
fplot('sin(x)*sqrt(x)', [0 20],'r');
hold('on');
fplot('2*log(x)-4',[1 18],'g');
grid('on');
```

ou

```
fplot('[sin(x)*sqrt(x),2*log(x)-4]', ...
      [0 20],'b');
grid('on');
ylim([-5 5])
```

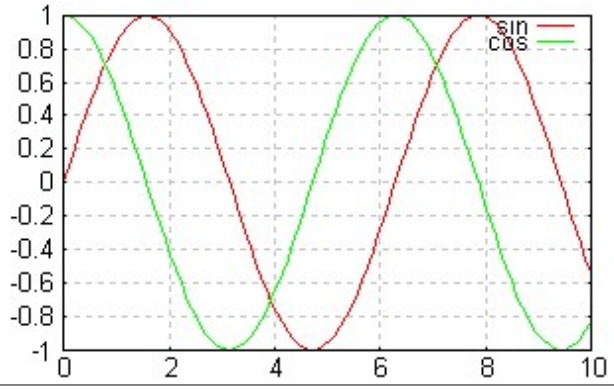
Remarque : constatez, dans la 1ère solution, que l'on a superposé les graphiques des 2 fonctions dans des plages de valeurs en X qui sont différentes !



Ex 2.1:

Grapher des fonctions built-in MATLAB/Octave :

```
fplot('sin',[0 10],'r');
hold('on');
fplot('cos',[0 10],'g');
grid('on');
```



Ex 2.2:

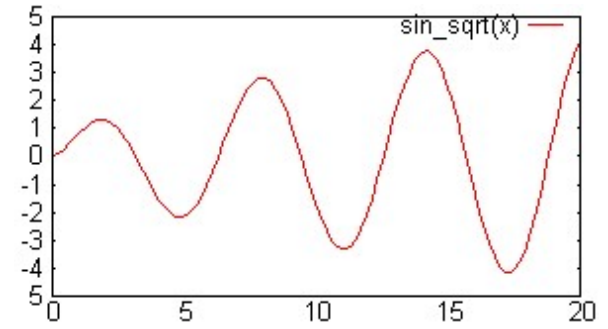
Définir une fonction utilisateur, puis la grapher :

1) Définition, dans un fichier nommé `sin_sqrt.m`, de la fonction suivante (voir chapitre **fonctions**) :

```
function [Y]=sin_sqrt(X)
  Y=sin(X).*sqrt(X);
return
```

Remarque: sous **Octave**, on pourrait aussi, au lieu de saisir le code de la fonction ci-dessus dans un M-file, l'entrer **interactivement** dans la fenêtre de commande Octave en terminant la saisie par `endfunction` (au lieu de `return`) ; ce qui donne lieu à une "compiled function".

2) Puis la grapher simplement avec :
`fplot('sin_sqrt(x)',[0 20],'r')`



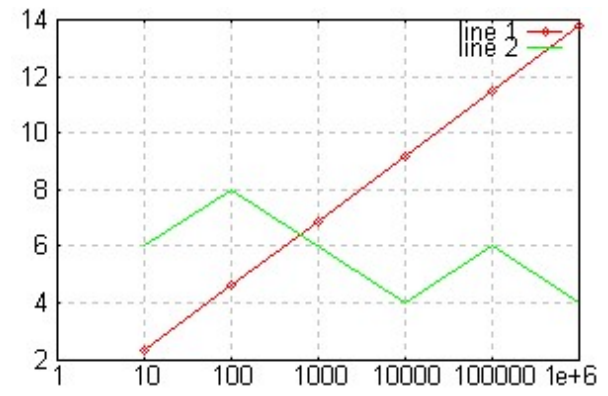
a) `semilogx(...)`
b) `semilogy(...)`
c) `loglog(...)`

Graphique 2D de lignes et/ou semis de points sur axes logarithmiques :
Ces 3 fonctions, qui s'utilisent exactement comme la fonction `plot` (mêmes paramètres...), permettent de grapher dans des systèmes d'axes logarithmiques :
a) axe X logarithmique, axe Y linéaire
b) axe X linéaire, axe Y logarithmique
c) axes X et Y logarithmiques

Voir aussi, plus bas, la fonction `plotyy` pour graphiques 2D à 2 axes Y qui peuvent être logarithmiques

Ex:

```
x1=logspace(1,6,6); y1=log(x1);
semilogx(x1,y1,'r-o', ...
         [10 100 1e4 1e5 1e6],[6 8 4 6 4],'g');
grid('on');
```



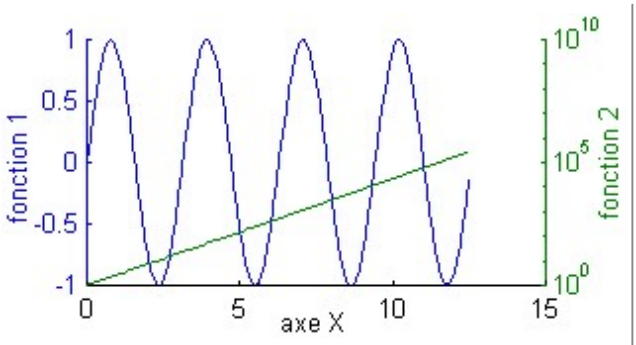
`plotyy(x1, y1, x2, y2 {,'type1' {,'type2'}} )`

Graphique avec 2 axes Y distincts :
Trace la courbe définie par les vecteurs *x1* et *y1* relativement à l'axe Y de gauche, et la courbe définie par les vecteurs *x2* et *y2* relativement à l'axe Y de droite.
Les paramètres optionnels *type1* et *type2* permettent de définir le type de primitive de tracé 2D utiliser. Ils peuvent notamment être : `plot`, `semilogx`, `semilogy`, `loglog`, `stem`...

Ex:

```
x=0:0.1:4*pi;
h= plotyy(x, sin(2*x), x, exp(x), ...
          'plot', 'semilogy');
xlabel('axe X');

hy1=get(h(1),'ylabel');
hy2=get(h(2),'ylabel');
set(hy1,'string','fonction 1');
set(hy2,'string','fonction 2');
```

Remarque : nous devons ici utiliser la technique des 'handles' (ici variables h, hy1 et hy2) pour étiqueter les 2 axes Y



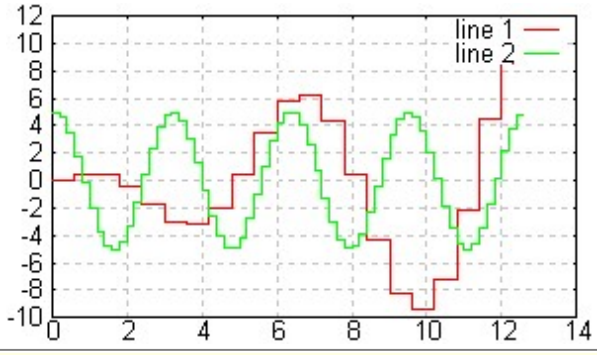
a) `stairs( {x,} y )`
b) `stairs( {x,} ymat {, linespec } )`

Graphique 2D en escaliers :
a) Dessine une ligne en escaliers pour la courbe définie par les vecteurs (ligne ou colonne) *x* et *y*. Si l'on ne fournit pas de vecteur *x*, la fonction utilise en X les indices de *y* (donc les valeurs 1 à length(*y*)).
b) Traçage de plusieurs courbes sur le même graphique en passant à cette fonction une matrice *ymat* dans laquelle chaque courbe fait l'objet d'une colonne. Sous Octave 3.4.2, on peut spécifier une couleur avec le paramètre *linespec*, mais pas un type de ligne.

Remarque : on peut aussi calculer le tracé de la courbe sans le dessiner avec l'affectation `[xs,ys]=stairs(...);`, puis le dessiner ultérieurement avec `plot(xs,ys,linespec);`

Ex:

```
x1=0:0.6:4*pi; y1=x1.*cos(x1);
stairs(x1,y1,'r');
hold('on');
x2=0:0.2:4*pi; y2=5*cos(2*x2);
stairs(x2,y2,'g');
grid('on');
```

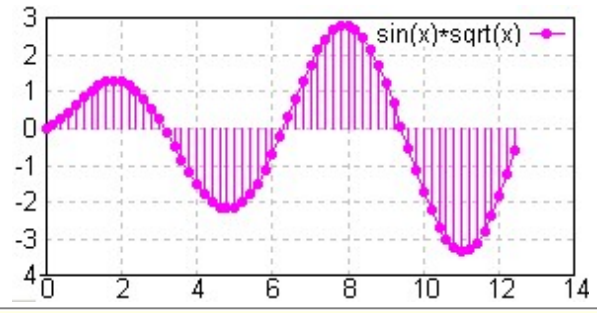


`stem( {x,} y {, linespec } )`

Graphique 2D en bâtonnets :
Graphe la courbe définie par les vecteurs (ligne ou colonne) *x* et *y* en affichant une ligne de rappel verticale (bâtonnet, pointe) sur tous les points de la courbe. Si l'on ne fournit pas de vecteur *x*, la fonction utilise en X les indices de *y* (donc les valeurs 1 à length(*y*)).

Ex:

```
x=0:0.2:4*pi; y=sin(x).*sqrt(x);
stem(x,y,'mo-');
grid('on');
```

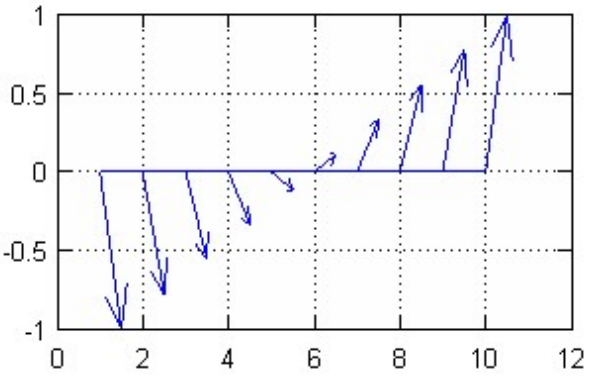


`feather( {dx,} dy)`

Graphique 2D de champ de vecteurs en "plumes" :
Dessine un champ de vecteurs dont les origines sont uniformément réparties sur l'axe X (en (1,0), (2,0), (3,0), ...) et dont les dimensions/orientations sont définies par les valeurs *dx* et *dy*

Ex:

```
dy=linspace(-1,1,10) ;
dx=0.5*ones(1,length(dy)) ;
   % vecteur ne contenant que des val. 0.5
feather(dx,dy)
grid('on')
axis([0 12 -1 1])
```

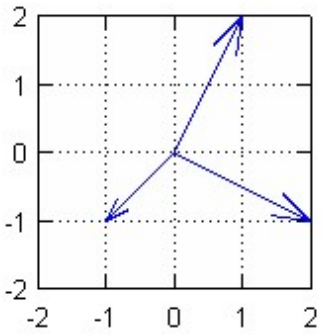


`compass( {dx,} dy)`

Graphique 2D de champ de vecteurs de type "boussole" :
Dessine un champ de vecteurs dont les origines sont toutes en (0,0) et dont les dimensions/orientations sont définies par les valeurs *dx* et *dy*

Ex:

```
compass([1 2 -1],[2 -1 -1])
axis([-2 2 -2 2])
grid('on')
```



a) `errorbar(x, y, error {,format} )`
b) `errorbar(x, y, lower, upper {,format} )`

Graphique 2D avec barres d'erreur :
a) Graphe la courbe définie par les vecteurs de coordonnées *x* et *y* (de type ligne ou colonne, mais qui doivent avoir le même nombre d'éléments) et ajoute, à cheval sur cette courbe et en chaque point de celle-ci, des barres d'erreur verticales symétriques dont la longueur totale sera le double de la valeur absolue des valeurs définies par le vecteur *error* (qui doit avoir le même nombre d'éléments que *x* et *y*).
b) Dans ce cas, les barres d'erreur seront **asymétriques**, allant de :

- M *y*-abs(*lower*) à *y*+abs(*upper*)
- O *y*-*lower* à *y*+*upper* (donc Octave utilise le signe des valeurs contenus dans ces vecteurs !)

Attention : le paramètre *format* a une signification différente selon que l'on utilise MATLAB ou Octave :

- M il correspond simplement au paramètre `linespec` (spécification de couleur, type de trait, symbole...) comme dans la fonction `plot`
- O la fonction `errorbar` de Octave offre davantage de possibilités que celle de MATLAB : ce paramètre *format* **doit commencer** par l'un des codes ci-dessous définissant le type de barre(s) ou box d'erreur à dessiner :
  - `~` : barres d'erreur verticales (comme sous MATLAB)
  - `>` : barres d'erreur **horizontales**
  - `~>` : barres d'erreur en X et en Y (il faut alors fournir **4 paramètres** `lowerX, upperX, lowerY, upperY` !)
  - `#~>` : dessine des "**boxes**" d'erreur

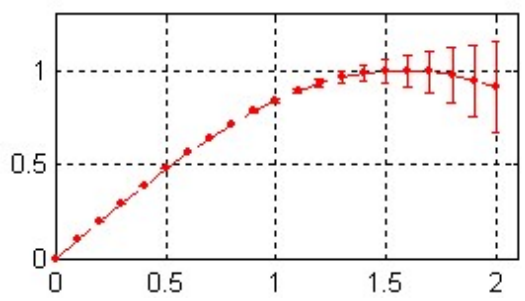
  puis se poursuit par le `linespec` habituel, le tout entre apostrophes

Voir en outre les fonctions suivantes, spécifiques à Octave : O `semilogxerr` , O `semilogyerr` , O `loglogerr`

Ex 1:

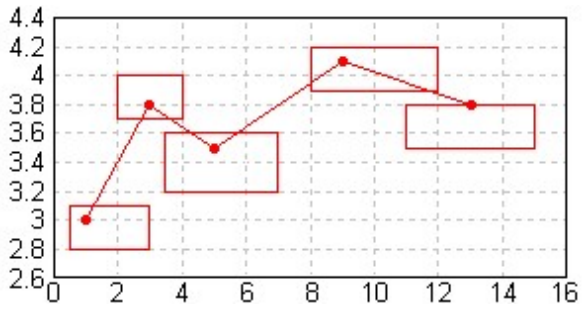
```
x=0:0.1:2; y=sin(x);
y_approx = x - (x.^3/6); % approximation fct sinus
error = y_approx - y;
errorbar(x,y,error,'r--o');
grid('on');
```

Remarque : on illustre ci-dessus la différence entre la fonction sinus et son approximation par un polynôme



Ex 2: (graphique ci-contre réalisé avec Octave)

```
x=[1 3 5 9 13];
y=[3 3.8 3.5 4.1 3.8];
lowerX=[0.5 1 1.5 1 2];
upperX=[2 1 2 3 2];
lowerY=[0.2 0.1 0.3 0.2 0.3];
upperY=[0.1 0.2 0.1 0.1 0];

errorbar(x,y, ...
   lowerX,upperX,lowerY,upperY,'#~>r');
hold('on');
plot(x,y,'r-o');
legend('off');
grid('on');
```



`scatter(` *x, y* **`{,`** *size* **`{,`** *color* **`} } {,`** *symbol* **`} {,'filled'} )`**

Graphique 2D de symboles :
Dessin du semis de points défini par les vecteurs de coordonnées *x* et *y* (de type ligne ou colonne, mais qui doivent avoir le même nombre d'éléments)

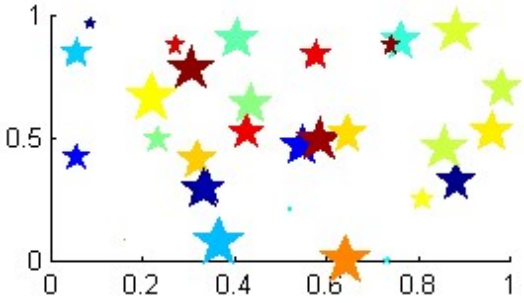
- *size* permet de spécifier la **surface** des symboles : ce peut être soit une valeur scalaire (=> tous les symboles auront la surface spécifiée), soit un vecteur de même dimension que *x* et *y* (=> indique alors taille de chaque symbole) ; concernant l'unité de ce paramètre :
  M surface du "carré englobant" du symbole en [pixels^2] : ex: 100 => symbole de surface 100 pixels^2 donc de coté 10 x 10 [pixels]
  O largeur et hauteur du "carré englobant" du symbole en [pixels]
- *color* permet de spécifier la couleur des symboles : ce peut être :
  • soit **une** couleur, appliquée uniformément à tous les symboles, exprimée sous forme de chaîne selon la syntaxe décrite plus haut (p.ex. `'r'` pour rouge)
  • soit un vecteur (de la même taille que *x* et *y*) qui s'appliquera linéairement à la colormap
  • ou une matrice n x 3 de couleurs exprimées en composantes RGB
- *symbol* permet de spécifier le type de symbole (par défaut: cercle) selon les possibilités décrites plus haut, c'est-à-dire `'o'`, `'*'`, `'+'`, `'x'`, `'^'`, `'<'`, `'>'`, `'v'`, `'s'`, `'d'`, `'p'`, `'h'`
- le paramètre-chaîne `'filled'` provoquera le remplissage des symboles

Remarque : en jouant avec l'attribut *color* et en choisissant une table de couleur appropriée, cette fonction permet de grapher des données 3D x/y/color

Ex: (graphique ci-contre réalisé avec MATLAB ou Octave/FLTK)

```
if ~ exist('OCTAVE_VERSION')
  facteur=50*50 ; % MATLAB
else
  facteur=50 ;    % Octave
end

scatter(rand(30,1),rand(30,1), ...
        facteur*rand(30,1),rand(30,1),'p','filled');
```

Remarque : nous graphons donc ici 30 paires de nombres x/y aléatoires compris entre 0 et 1 ; de même, nous définissons la couleur et la taille des symboles de façon aléatoire



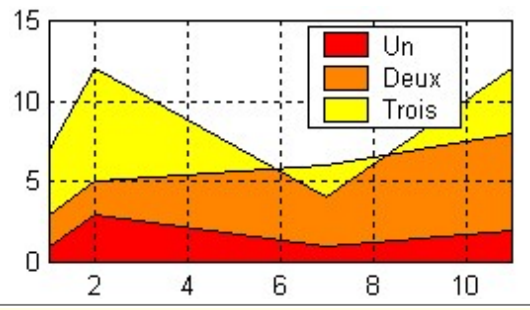
`area( {` *x,* **`}`** *ymat* **`)`**

Graphique 2D de type surface :
Graphe de façon empilée (cumulée) les différentes courbes définies par les colonnes de la matrice *ymat*, et colorie les surfaces entre ces courbes. Le nombre d'éléments du vecteur *x* (ligne ou colonne) doit être identique au nombre de lignes de *ymat*. Si l'on ne spécifie pas *x*, les valeurs sont graphées en X selon les indices de ligne de *ymat*.
Remarque : si on ne veut pas "empiler" les surfaces, on utilisera plutôt la fonction `fill`

☒ O sous Octave 3.4.2/FLTK, l'usage de la fonction `colormap` est sans effet

Ex: (graphique ci-contre réalisé avec MATLAB)

```
x=[1 2 7 11];
ymat=[1 2 4 ; 3 2 7 ; 1 5 -2 ; 2 6 4];
area(x,ymat);
colormap(autumn); % changement palette couleurs
grid('on');
set(gca,'Layer','top'); % quadrillage 1er plan
legend('Un','Deux','Trois')
```



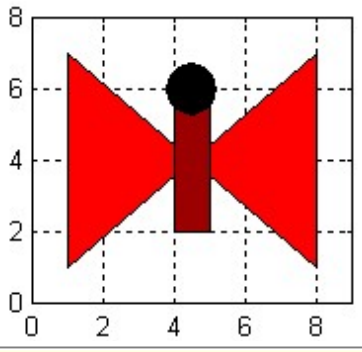
a) **`fill(`** *x, y, couleur* **`)`**
b) **`fill(`** *xA, yA, couleurA* **`{,`** *xB, yB, couleurB ...* **`} )`**
c) **`patch(`** *x, y, couleur* **`)`**

Dessin 2D de surface(s) remplie(s) :
a) Dessine et rempli de la *couleur* spécifiée le polygone défini par les vecteurs de coordonnées *x* et *y*. Le polygone bouclera automatiquement sur le premier point, donc il n'y a pas besoin de définir un dernier couple de coordonnées xn/yn identique à x1/y1.
b) Il est possible de dessiner plusieurs polygones (A, B...) d'un coup en une seule instruction en passant en paramètre à cette fonction plusieurs triplets *x*,*y*,*couleur*.
c) Primitive de bas niveau de tracé de surfaces remplies, cette fonction est analogue à `fill` sauf qu'elle accumule (tout comme la primitive de dessin de ligne `line`) son tracé dans la figure courante sans qu'il soit nécessaire de faire au préalable un `hold('on')`

On spécifie la *couleur* par l'un des codes de couleur définis plus haut (p.ex. pour rouge: `'r'` ou `[1.0 0 0]`)

Ex: (graphique ci-contre réalisé avec MATLAB ou Octave)

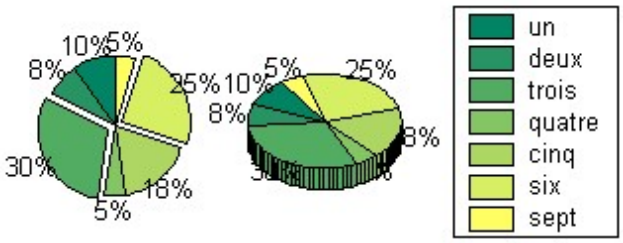
```
a=linspace(0,2*pi,20);
  x= 4.5 + 0.7*cos(a); % contour disque noir de
  y= 6.0 + 0.7*sin(a); % rayon 0.7, centre 4.5/6.0
 fill([1 8 8 1],[1 7 1 7],'r', ...
      [4 5 5 4],[2 2 6 6],[0.6 0 0], ...
      x,y,'k');
axis('equal');
axis([0 9 0 8]);
grid('on');
```



a) `pie(val {,explode} {,labels} )`
b) `pie3(val {,explode} {,labels} )`
Graphique de type camembert :
a) Dessine un camembert 2D sur la base du vecteur `val`, chaque valeur se rapportant à une tranche de gâteau. Le vecteur logique `explode` (de même taille que *val* et composé de 0 ou de 1) permet de spécifier (avec 1) quelles tranches de gâteau doivent être "détachées"
Le vecteur cellulaire `labels` (de même taille que *val* et composé de chaînes de caractères) permet de spécifier le texte à afficher à coté de chaque tranche en lieu et place des pourcentages
b) Réalise un camembert en épaisseur (3D)

Ex: (graphiques ci-contre réalisés avec MATLAB ou Octave)

```
val=[20 15 60 10 35 50 10];

subplot(1,2,1);
pie(val, [0 0 1 0 0 1 0]);
colormap(summer);  % changement palette couleur

subplot(1,2,2);
pie3(val);
legend('un','deux','trois','quatre', ...
       'cinq','six','sept', ...
       'location','east');
```



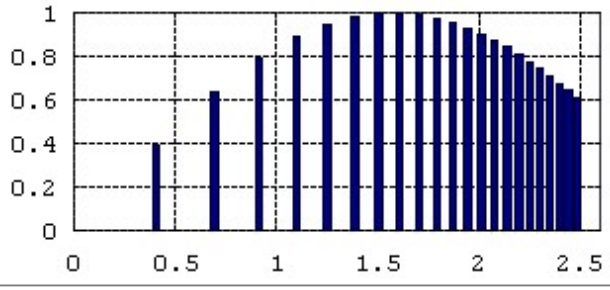
a) `bar( {x,} y )`
b) `bar( {x,} mat {,larg} {,'style'} )`
c) `barh( {y,} mat {,larg} {,'style'} )`
Graphique 2D en barres :
a) Dessine les barres verticales définies par les vecteurs *x* (position de la barre sur l'axe horizontal) et *y* (hauteur de la barre). Si le vecteur *x* n'est pas fourni, les barres sont uniformément réparties en X selon les indices du vecteur *y* (donc positionnées de 1 à n).
b) Sous cette forme, on peut fournir une matrice *mat* dans laquelle chaque ligne définira un groupe de barres qui seront dessinées :
- côte-à-côte si le paramètre *style* n'est pas spécifié ou que sa valeur est `'grouped'`
- de façon empilée si la valeur de ce paramètre est `'stacked'`

Avec le paramètre *larg*, on spécifie le rapport "largeur des barres / distance entre barres" dans le cadre du groupe ; la valeur par défaut est 0.8 ; si celle-ci dépasse 1, les barres se chevaucheront. Le nombre d'éléments du vecteur *x* doit être égal au nombre de lignes de la matrice *mat*.
c) Identique à la forme b), sauf que les barres sont dessinées horizontalement et positionnées sur l'axe vertical selon les valeurs du vecteur *y*

Remarque : on peut aussi calculer les tracés sans les dessiner avec l'affectation `[xb,yb]=bar(...);`, puis les dessiner ultérieurement avec `plot(xb,yb,linespec);`

Ex 1:

```
x=log(1:0.5:12);
y=sin(x);
bar(x,y);
axis([0 2.6 0 1]);
grid('on');
```



Ex 2:

Premier graphique ci-contre :

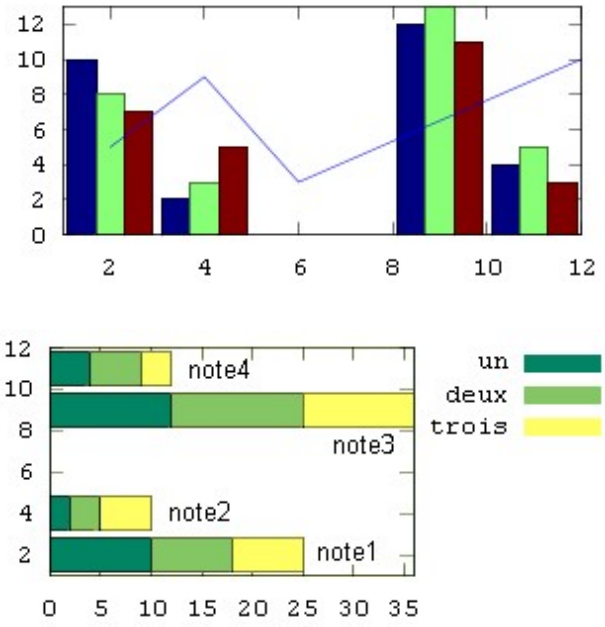
```
x=[2 4 9 11];
mat=[10 8 7 ; 2 3 5 ; 12 13 11 ; 4 5 3];
bar(x,mat,0.9,'grouped');
hold('on');
plot([2 4 6 12],[5 9 3 10]);
```

Second graphique ci-contre :

```
barh(x,mat,0.8,'stacked');
legend('un','deux','trois',-1)
colormap(summer)
```

On a ensuite **annoté** le second graphique, en placement interactivement des chaînes de texte définies dans un tableau avec le code ci-dessous :

```
annot={'note1','note2','note3','note4'};
for n=1:length(annot)
   gtext(annot{n});
end
```



a) [*nval* {*xout*}] = **`hist(y {,n} )`**
b) [*nval* {*xout*}] = **`hist(y, x )`**

Histogramme 2D de distribution de valeurs, ou calcul de cette distribution :
a) Détermine la **répartition** des valeurs contenues dans le vecteur *y* (ligne ou colonne) selon *n* catégories (par défaut 10) de même 'largeur' (catégories appelées boîtes, bins, ou containers), puis dessine cette répartition sous forme de graphique 2D en barres où l'axe X reflète la plage des valeurs de *y*, et l'axe Y le nombre d'éléments de *y* dans chacune des catégories.
IMPORTANT: Si l'on affecte cette fonction à **`[nval {xout}]`**, le graphique n'est **pas** effectué, mais la fonction retourne le vecteur-ligne *nval* contenant nombre de valeurs trouvées dans chaque boîte, et le vecteur-ligne *xout* contenant les valeurs médianes de chaque boîtes. On pourrait ensuite effectuer le graphique à l'aide de ces valeurs tout simplement avec la fonction **`bar(xout,nval)`**.
b) Dans ce cas, le vecteur *x* spécifie les valeurs du 'centre' des boîtes (qui n'auront ainsi plus nécessairement la même largeur !) dans lesquelles les valeurs de *y* seront distribuées, et l'on aura autant de boîtes qu'il y a d'éléments dans le vecteur *x*.

Voir aussi la fonction **`[nval {vindex}]=histc(y,limits)`** (qui ne dessine pas) permettant de déterminer la distribution des valeurs de *y* dans des catégories dont les 'bordures' (et non pas le centre) sont précisément définies par le vecteur *limits*.

Remarque : sous MATLAB, *y* peut aussi être une **matrice** de valeurs ! Si cette matrice comporte k colonnes, la fonction **`hist`** effectue k fois le travail en examinant les valeurs de la matrice *y* colonne après colonne. Le graphique contiendra alors *n* groupes de *k* barres. De même, la variable *nval* retournée sera alors une matrice de *n* lignes et k colonnes, mais *xout* restera un vecteur de *n* valeurs (mais, dans ce cas, en colonne).
Remarque : il existe sous Octave une variante de cette fonction nommée **`hist2d`** (dans le package "plot")

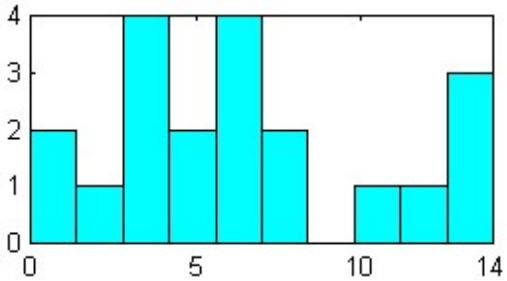
Voir (plus bas) la fonction **`rose`** qui réalise aussi des histogrammes de distribution mais dans un système de coordonnées polaire.

Ex:

```
y=[4 8 5 2 6 8 0 6 13 14 10 7 4 3 12 13 6 3 5 1];
```

1) Si l'on ne spécifie pas n => n=10 catégories, et comme les valeur *y* vont de 0 à 14, les catégories auront une largeur de (14-0)/10 = 1.4, et leurs 'centres' *xout* seront respectivement : 0.7, 2.1, 3.5, 4.9, etc... jusqu'à 13.3

```
[nval xout]=hist(y)
% => nval=[2 1 4 2 4 2 0 1 1 3]
%    xout=[0.7 2.1 3.5 4.9 6.3 7.7 9.1
%          10.5 11.9 13.3]
hist(y);   % => 1er graphique ci-contre
```



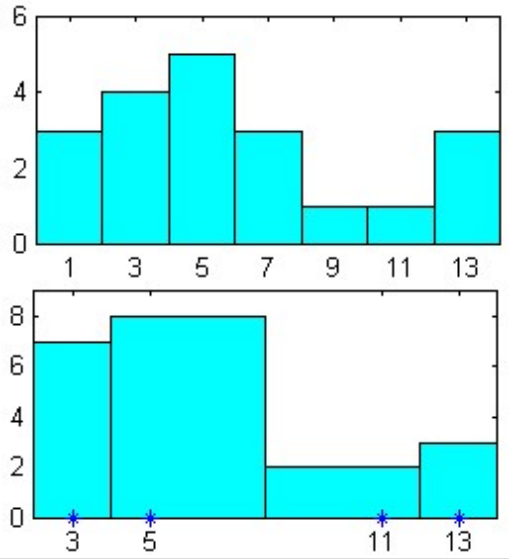
```
set(gca,'XTick',xout) % annote axe X sous barres
```

2) Spécifions n=7 catégories => elles auront une largeur de (14-0)/7 = 2, et leurs 'centres' *xout* seront respectivement : 1, 3, 5, 7, 9, 11 et 13

```
[nval xout]=hist(y,7)
% => nval=[3 4 5 3 1 1 3]
%    xout=[1 3 5 7 9 11 13]
hist(y,7);   % => 2e graphique ci-contre
set(gca,'XTick',xout) % annote axe X sous barres
```

3) Spécifions un vecteur centres=[3 5 11 13] définissant les centres de 4 boîtes

```
[nval xout]=hist(y,centres)
% => nval=[7 8 2 3]
%    xout=[3 5 11 13] % identique à centres
hist(y,centres)  % => 3e graphique ci-contre
axis([2 14 0 9]);
set(gca,'XTick',centres) % annote axe X sous barres
```



a) **`plotmatrix(`***`m1, m2 {,linespec}`***`)`**
b) **`plotmatrix(`***`m {,linespec}`***`)`**

Matrice de graphiques en semis de points :
a) En comparant les **colonnes** de la matrice *m1* (de dimension P lignes x M colonnes) avec celles de *m2* (de dimension P lignes x N colonnes), affiche une matrice de N (verticalement) x M (horizontalement) graphiques en semis de points
b) Cette forme est équivalente à **`plotmatrix(`***`m, m {,linespec}`***`)`**, c'est à dire que l'on effectue toutes les comparaisons possibles, deux à deux, des colonnes de la matrice *m* et qu'on affiche une matrice de comportant autant de lignes et colonnes qu'ily a a de colonnes dans *m*. En outre dans ce cas les graphiques se trouvant sur la diagonale (qui représenteraient des semis de points pas très intéressants, car distribués selon une ligne diagonale) sont remplacés par des graphiques en histogrammes 2D (fréquence de distribution) correspondant à la fonction **`hist(m(:,`***`i`***`))`**

Sous Octave 3.2.0 à 3.4.2, cette fonction est buguée si N est différent de M (exemple 1 ci-dessous)

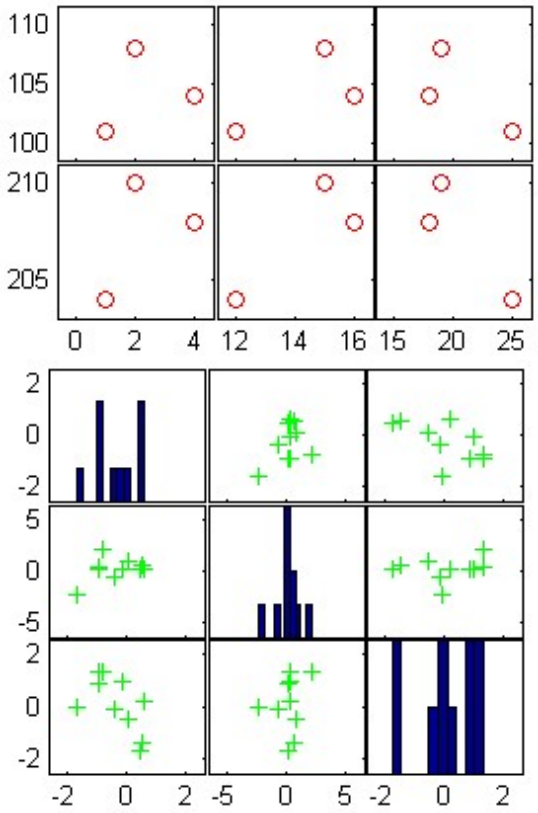
Ex:

1) La fonction ci-dessous produit le premier graphique ci-contre

```
plotmatrix([1 12 25; 2 15 19; 4 16 18], ...
           [101 204; 108 210; 104 208], 'ro')
```

2) La fonction ci-dessous produit le second graphique ci-contre

```
plotmatrix( randn(10,3), 'g+')
```



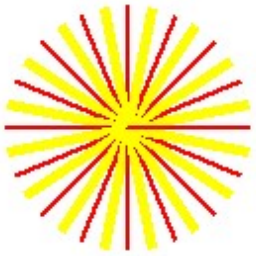
`line(`*`x, y {,z} {,'property', value }`***`)`**

Primitive de tracé de lignes 2D/3D :
Cette fonction est une primitive de tracé de **lignes** 2D/3D de bas niveau proche de **`plot`** et **`plot3`**. Elle s'en distingue cependant par le fait qu'elle permet d'accumuler, dans un graphique, des tracés sans qu'il soit nécessaire de mettre **`hold`** à **`on`** !

Remarque : la primitive de tracé de **surfaces remplies** de bas niveau est **`patch`**

Ex:

```
hold('off'); clf;
for k=0:32
  angle=k*2*pi/32;
  x=cos(angle); y=sin(angle);
  if mod(k,2)==0
    coul='red'; epais=2;
  else
    coul='yellow'; epais=4;
  end
   line([0 x], [0 y], ...
        'Color', coul, 'LineWidth', epais);
end
axis('off'); axis('square');
```



`polar(angle, rayon {,linespec} )`

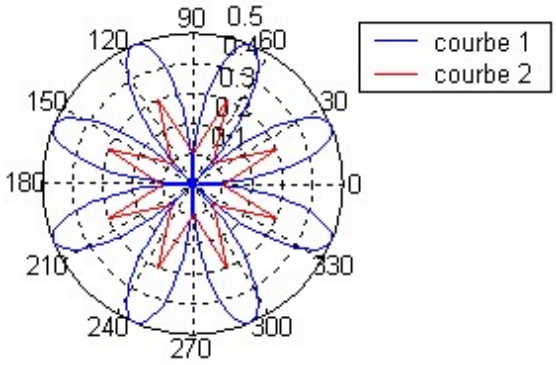
Graphique 2D de lignes et/ou semis de points en coordonnées polaires :
Reçoit en paramètre les coordonnées polaires d'une courbe (ou d'un semis de points) sous forme de 2 vecteurs *angle* (en radian) et *rayon* (vecteurs ligne ou colonne, mais de même taille), dessine cette courbe sur une grille polaire.
On peut tracer plusieurs courbes en utilisant `hold('on')`, ou en passant à cette fonction des matrices *angle* et *rayon* (qui doivent être de même dimension), la i-ème courbe étant construite sur la base des valeurs de la i-ème colonne de *angle* et de *rayon*.

Sour Octave 3.0 à 3.4.2, il n'est pas possible d'afficher le quadrillage polaire ; on effacera en outre le cadre avec `axis('off')`

Voir aussi la fonction `ezpolar` qui permet de tracer, dans un système polaire, une fonction définie par une expression.
Voir en outre les fonctions `cart2pol` et `pol2cart` de conversion de coordonnées carthésiennes en coordonnées polaires et vice-versa.

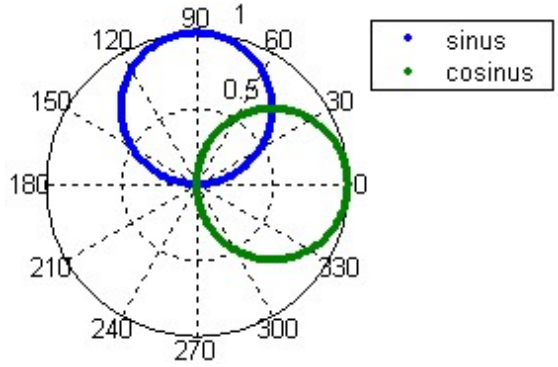
Ex 1: (graphique ci-contre réalisé avec MATLAB)

```
angle1=0:0.02:2*pi;
rayon1=sin(2*angle1).*cos(2*angle1);
polar(angle1, rayon1, 'b');

angle2=0:pi/8:2*pi;
rayon2=[repmat([0.1 0.3],1,8), 0.1];
hold('on');
polar(angle2, rayon2, 'r');

legend('courbe 1','courbe 2');
```



Ex 2: (graphique ci-contre réalisé avec MATLAB)

```
angle=0:0.02:2*pi;
polar([angle' angle'], ...
      [sin(angle') cos(angle')],'.');
legend('sinus','cosinus');
```



`rose(val {,n} )`

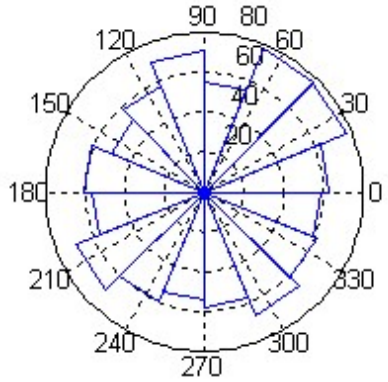
Histogramme polaire de distribution de valeurs (ou histogramme angulaire) :
Cette fonction est analogue à la fonction `hist` vue plus haut, sauf qu'elle travaille dans un système polaire angle/rayon. Les valeurs définies dans le vecteur *val*, qui doivent ici être **comprises entre 0 et 2*pi**, sont réparties dans *n* catégories (par défaut 20 si *n* n'est pas spécifié) et dessinées sous forme de tranche de gâteau dans un diagramme polaire où l'angle désigne la plage des valeurs, et le rayon indique le nombre de valeurs se trouvant dans chaque catégorie.

Sour Octave 3.0 à 3.4.2, il n'est pas possible d'afficher le quadrillage polaire; on effacera en outre le cadre avec `axis('off')`

Ex: (graphique ci-contre réalisé avec MATLAB)

```
rose(2*pi*rand(1,1000),16);
```

Explications : on établit ici un vecteur de 1000 nombres aléatoires compris entre 0 et 2*pi, puis on calcule et dessine leur répartition en 16 catégories (1ère catégorie pour les valeurs allant de 0 à 2*pi/16, etc...).



Autres fonctions graphiques 2D non décrites dans ce support de cours :

- `rectangle` : dessin de rectangles 2D (avec angles arrondis)