LE GRAPHISME AVEC MuPAD

J. ROUSSEL – http://www.femto-physique.fr/analyse_numerique/

Année 2013-2014

Résumé

Ce tutoriel présente les outils graphiques de MuPAD. Dans un premier temps, nous passons en revue les commandes permettant de saisir les notions essentielles. Nous abordons ensuite, via une Foire Aux Questions quelques astuces ou concepts plus subtils en matière de graphisme et d'animation. Ce guide synthétique accompagnera avantageusement l'Aide de MuPAD qui est une source d'apprentissage plus exhaustive.

1 Notions essentielles

Nous présentons ici quelques commandes graphiques en précisant à chaque fois leur contexte, leur syntaxe et en donnant des exemples.

1.1 Graphes de fonction

1.1.1 Fonction d'une variable

Soit la fonction réelle $f : x \to f(x)$. Le graphe de f est l'ensemble des points $M(x, y)$ du plan cartésien tel que $y = f(x)$. Soit la fonction réelle $f : x \to f(x)$. Le graphe de f est l'ensemble des points $M(x, y)$ du plan cartésien tel que $y = f(x)$. Sous MuPAD, la manière la plus simple de tracer le graphe d'une fonction est d'utiliser la commande `plotfunc2d` dont la syntaxe est :

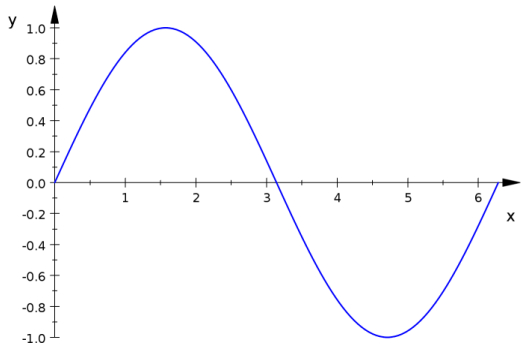
```
[ plotfunc2d(f(x), x = xmin .. xmax,Options);
```

Une autre méthode - qui a un intérêt quand on veut superposer plusieurs objets graphiques (voir section-1.5) - consiste à **définir** l'objet graphique à l'aide de la commande `plot::Function2d` (attention à la majuscule !) puis à le **tracer à l'écran** grâce à la commande `plot`. La syntaxe est la suivante :

```
[ fig:=plot::Function2d(f(x), x=xmin .. xmax)://définition
[ plot(fig); //sortie à l'écran
```

Exemple 1

```
[ f:= x->sin(x):
[ plotfunc2d(f,x=0..2*PI);
```



1.1.2 Fonction de deux variables

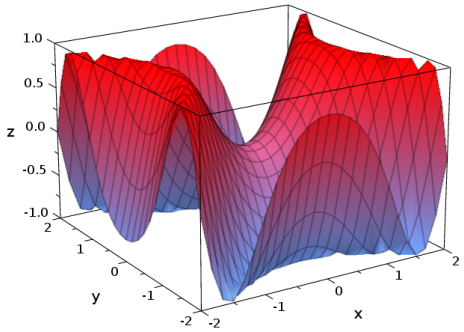
Soit la fonction réelle $f : (x, y) \to f(x, y)$. Le graphe de f est l'ensemble des points $M(x, y, z)$ de l'espace cartésien tel que $z = f(x, y)$. Là encore, il y a deux commandes qui donneront le même résultat :

`plotfunc3d` et `plot::Function3d`.

La syntaxe est identique à celle des commandes `plotfunc2d` et `plot::Function2d`.

Exemple 2

```
[ f:=(x,y)->sin(x^2+y^2):
[ fig:=plotFunction3d(f,x=-2..2,y=-2..2):
[ plot(fig);
```



1.2 Courbes et surfaces implicites

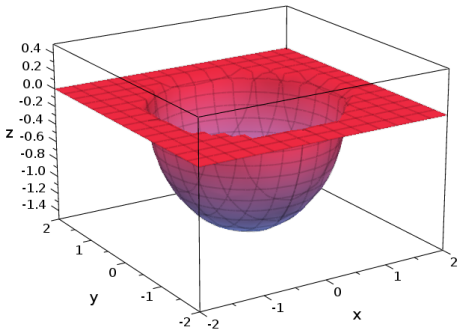
Si l'on cherche l'ensemble des points $M(x, y)$ tel que $f(x, y) = 0$ (on obtient alors une courbe de niveau de f), on utilisera la commande `plot::Implicit2d`. Dans le cas où l'on cherche l'ensemble des points $M(x, y, z)$ tel que $f(x, y, z) = 0$ (là, il s'agit d'une surface de niveau de f), on utilisera la commande `plot::Implicit3d`. La syntaxe est la suivante :

```
[ fig:= plot::Implicit2d(f,x=xmin..xmax,y=ymin..ymax):

[ plot(fig);
```

Exemple 3

```
[ f:=(x,y,z)->min(x^2+y^2+z^2-2,-z):
[ fig:=plot::Implicit3d(f, x=-2..2, y=-2..2, z=-1.5..0.5):
[ plot(fig);
```



1.3 Courbes paramétriques

La commande `plot::Curve2d` permet de tracer la courbe paramétrique d'équation $[x(t), y(t)]$ où t est le paramètre variant entre t_{min} et t_{max}. Comme vous l'aurez deviné, la commande `plot::Curve3d` permet, quant à elle, le tracé de la courbe paramétrique d'équation $[x(t), y(t), z(t)]$.

La syntaxe est la suivante :

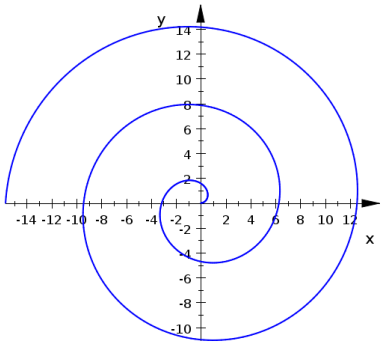
```
[ fig:= plot::Curve2d([x(t), y(t)],t=tmin..tmax,Mesh=N)):

[ plot (fig);
```

avec N le nombre de points calculés.

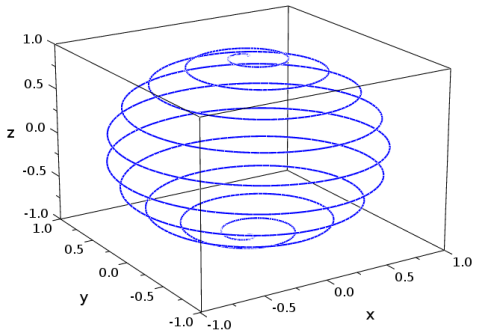
Exemple 4
La spirale d'Archimède a pour équation paramétrique $[x(t) = t\cos t,\, y(t) = t\sin t]$ ce qui donne :

```
[ fig:=plot::Curve2d([t*cos(t),t*sin(t)], t=0..35,Mesh=500):
[ plot(fig);
```



Exemple 5
Traçons une hélice s'enroulant sur une sphère de rayon 1.

```
[ fig:= plot::Curve3d([sin(theta)*cos(20*theta),sin(theta)*sin(20*theta),cos(theta)],
 theta=0..PI,Mesh=1000):
[ plot(fig);
```



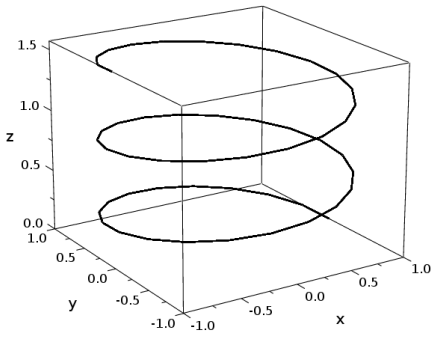
Si l'on connaît l'équation de la courbe dans d'autres systèmes de coordonnées on peut utiliser des **commandes spécifiques :**

- `plot::Polar` pour le système de coordonnées polaires $[r(t), \phi(t)]$;
- `plot::Cylindrical` pour le système de coordonnées cylindriques $[r, \phi, z]$;
- `plot::Spherical` pour le système de coordonnées spéhriques $[r, \phi, \theta]$ (ϕ est l'angle polaire et θ la colatitude).

Exemple 6

Représentation d'une hélice d'équation : $\begin{cases} r(t) = 1 \\ \phi(t) = t \\ z(t) = 0,1t \end{cases}$

```
[ fig:=plot::Cylindrical([1,t,0.1*t],t=0..5*PI,z=0..2*PI, Mesh=[50,50])):
[ plot(fig);
```



1.4 Formes géométriques

Des formes géométriques simples peuvent être dessinées à l'aide de commandes spécifiques dont voici une liste non exhaustive :

`plot::Arc2d(r,[x0,y0],`θ_1`..`$\theta_2`)` : trace un arc de rayon r de centre [x0,y0] et d'angle compris entre θ_1 et θ_2.

`plot::Arrow2d ([x1,y1],[x2,y2])` : trace une flèche joignant le point [x1, y1] au point [x2, y2].

`plot::Arrow3d([x1,y1,z1],[x2,y2,z2])` : même chose dans l'espace.

`plot::Circle2d(r,[x0,y0])` : trace un cercle de centre [x0,y0] et de rayon r.

`plot::Circle3d(r,[x0,y0,z0])` : idem mais dans l'espace.

`plot::Line2d([x1,y1],[x2,y2])` : trace une ligne entre [x1, y1] et [x2, y2].

`plot::Line3d([x1,y1,z1],[x2,y2,z2])` : trace une ligne entre [x1, y1, z1] et [x2, y2, z2].

`plot::Point2d(x,y)` : Trace un point de coordonnées (x,y).

`plot::Point3d(x,y,z)` : Trace un point de coordonnées (x,y,z).

`plot::PointList2d(liste-coord)` : Si liste-coord est une liste de coordonnées du type ([x1,y1],..,[xn,yn]) alors cette commande trace l'ensemble de ces points.

`plot::PointList3d (liste-coord) :` Idem. Ici, il faut que la liste soit de la forme ([x1,y1,z1],..,[xn,yn,zn]).

`plot::Sphere(r,[xc,yc,zc])` : Trace une sphère de rayon r et de centre C ayant pour coordonnées [xc,yc,zc].

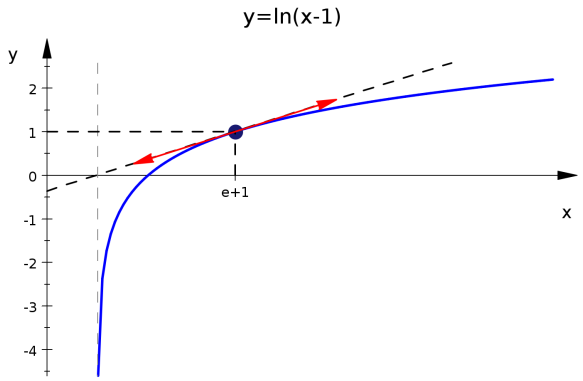
`plot::Turtle(commandes,options)` : dessine des formes géométriques en déplaçant un crayon obéissant à certaines commandes. Les commandes sont :

- Left(a), Right(a) : le crayon tourne d'un angle a.
- Forward(d) : le crayon trace un trait de longueur d à partir de sa position initiale.
- Up et Down : pour lever et descendre le crayon

Exemple 7

//Définition de la fontion f et de son graphe F en enlevant les graduations et en plaçant uniquement une marque pour $x = e + 1$.

```
[ f := x ->ln(x-1):
[ F:=plot::Function2d(f,x=0..10, XTicksAt=[E+1=''e+1''],XTicksNumber=None,LineWidth=0.5):
//Définition d'un point sur la courbe de 3mm de diamètre.
[ pt:=plot::Point2d(E+1,f(E+1), PointSize=3.0*unit::mm):
//Définition de lignes pointillées pour marquer les coordonnées du point et pour la tangente à la courbe en ce point.
[ L1:=plot::Line2d([0,1],[E+1,1],Color=RGB::Black,LineStyle=Dashed):
[ L2:=plot::Line2d([0,-1/E],[8,7/E],Color=RGB::Black,LineStyle=Dashed):
[ L3:=plot::Line2d([E+1,0],[E+1,1],Color=RGB::Black,LineStyle=Dashed):
//Définition d'une liste contenant de flèches rouges et tangentes au même point.
[ fleches:=plot::Arrow2d([E+1,1],[E+3, 1+2/E],Color=RGB::Red),
 plot::Arrow2d([E+1,1],[E-1, 1-2/E],Color=RGB::Red):
//Tracé du graphe, du point, des lignes et des flèches. La scène contient un titre.
[ plot(F,pt,L1,L2,L3,fleches,Header=''y=ln(x-1)'');
```



1.5 Objets graphiques

Sous MuPAD les figures font parties d'une arborescence à 3 niveaux :

1. **le canevas**, représente le cadre dans lequel on trouve la ou les scènes graphiques ;
2. **la scène** graphique est le cadre dans lequel on trouve la ou les figures ;
3. **la figure** est obtenue à l'aide d'une commande du type `plot::commande()`.

Si `fig1` et `fig2` sont deux figures, on peut alors les rassembler dans une même scène en faisant

```
[ plot(fig1,fig2);
```

Exemple 8

On cherche à tracer le graphe de la fonction réelle $f : x \to x \sin x$ et sa tangente en $x_0 = 1,2$ ainsi qu'un point de coordonnées $(x_0, f(x_0))$.

//Définition de la fonction f et de deux paramètres.

```
[ f:=x->x*sin(x): x0:=1.2: dx:=1:
```

//Définition du graphe de f.

```
[ fig1:=plot::Function2d(f(x),x=0..2*PI):
```

//Définition d'un point d'abscisse x_0 situé sur la courbe.

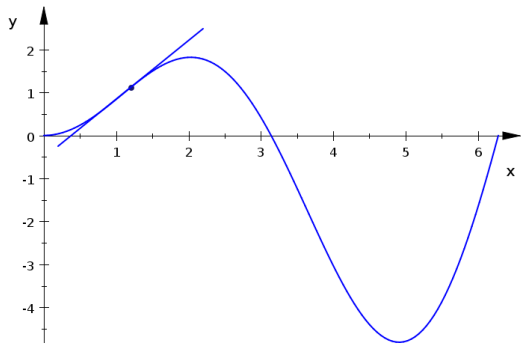
```
[ fig2:=plot::Point2d(x0,f(x0)):
```

//Définition d'un segment tangent à la courbe.

```
[ fig3:=plot::Line2d([x0-dx,f(x0)-f'(x0)*dx],[ x0+dx,f(x0)+f'(x0)*dx]):
```

//Tracé de la courbe, du point et du segment.

```
[ plot(fig1,fig2,fig3);
```



Nous venons de voir comment superposer plusieurs figures dans une même scène. Parfois, on préfère disposer des scènes graphiques côte-à-côte. Dans ce cas il faut définir plusieurs scènes à l'aide de `plot::Scene2d` puis les disposer en canevas à l'aide de la commande `plot::Canvas`.

Exemple 9

//Définition de 6 scènes graphiques à l'aide d'une boucle for. Chaque scène contient une courbe de LISSAJOUS.

```
[ for i from 1 to 6 do
[ curve[i]:=plot::Scene2d(plot::Curve2d([cos(theta+0.25*PI),cos(i*theta)],theta=0..2*PI)):
[ end_for:
```

//Définition du canevas graphique.

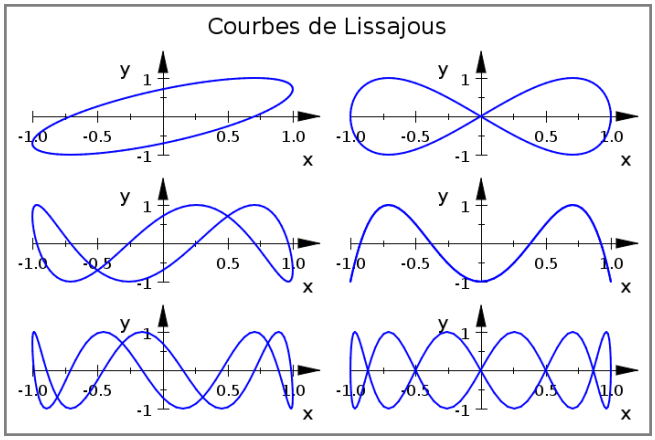
```
[ C:=(plot::Canvas(curve[i],$ i=1..6)):
```

// Réglages de quelques options du canevas : titre, bordure,police.

```
[ C::Header:=''Courbes de Lissajous'':
[ C::BorderWidth:=0.5*unit::mm:
[ C::HeaderFont:=[''Times New Roman'',Bold, 18]:
```

//Tracé du canevas.

```
[ plot(C);
```



1.6 Quelques options utiles

Chaque objet peut recevoir un certain nombre d'attributs sous la forme d'options mis en argument dans la commande. Par exemple, si l'on veut une courbe noire, on peut écrire :

```
[ fig1:=plot::Function2d(f(x),x=0..2*PI,Color=RGB::Black):
```

Il n'est pas question ici de passer en revue toutes les options et nous vous invitons à utiliser la boîte de contrôle (« Object Browser ») qui s'affiche à droite lorsque vous cliquez sur l'objet graphique. En navigant dans cette boîte vous aurez un aperçu des options de chaque objet graphique que vous pouvez modifier en temps réel.

Cependant, voici quelques options très utiles :

Légende et titre On peut donner un titre à une scène ou à un canevas graphique à l'aide de de la commande `Header="Titre"`. Chaque courbe peut être légendée grâce à `Legend="texte"`. Si l'on ne veut pas afficher de légende, il faut ajouter à la scène graphique, l'option `LegendVisible="FALSE"`.

Graduations Les axes sont automatiquement gradués. On peut jouer sur l'espacement des graduations (prenons l'exemple de l'axe Ox) grâce à `XTicksBetween`, sur le nombre de graduations grâce à `XTicksNumber` et sur la visibilité des graduation grâce à `XTicksVisible`.

Grille La boîte graphique n'est pas quadrillée par défaut. On peut ajouter un quadrillage (suivant X par exemple) avec l'option `XGridVisible`. Le style (continu, discontinu, pointillé) est défini grâce à `XGridLineStyle`. Les mêmes options existent pour la « sous-grille » (`XSubgridVisible`, `XSubgridVisible` et `XGridLineStyle`).

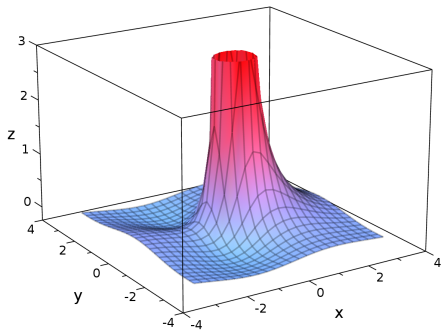
ViewingBox l'option `ViewingBox` permet d'imposer les dimensions du système de coordonnées. Attention, à ne pas confondre avec les intervalles $x_{min}..x_{max}$ que l'on retrouve par exemple dans `Plot::Function2d`. La syntaxe est :

`ViewingBox=[xmin..xmax,ymin..ymax,zmin..zmax]`.

Exemple 10

la figure ci-contre correspond au code suivant :

```
//Définition d'une fonction et de son graphe (ici une surface).
[ f:=x->(sin(x)+cos(y))/(x^2+y^2):
[ fig:=plot::Function3d(f,x=-PI..PI,y=-PI..PI):
//Tracé du graphe. Notez que la figure ne remplit pas toute la scène.
[ plot(fig,ViewingBox=[-4..4,-4..4,Automatic..3);
```



Scaling l'option `Scaling` permet de définir le rapport d'échelle. Avec `Scaling=Constrained`, les cercles apparaissent circulaires et non elliptiques. Avec `Scaling = Unconstrained`, les échelles des différents axes ne sont pas nécessairement identiques ce qui fait qu'un cercle peut paraître elliptique.

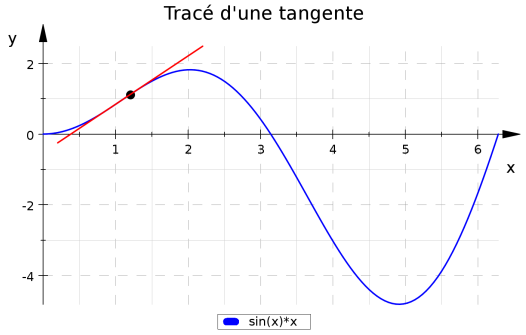
AdaptativeMesh l'option `AdaptativeMesh` contrôle la précision du tracé grâce à une grille de calcul adaptative ; la grille est alors plus resserrée là ou la fonction subit des variations importantes (utile pour les fonctions discontinues par exemple). Cette option attend une valeur entière positive en suivant la syntaxe `AdaptiveMesh = n`. La profondeur `n` doit rester petit (1, 2 ou 3).

Ces options sont à rentrer dans les commandes sous la forme `Option=valeur`. Cependant, il est possible de modifier l'option de chaque objet graphique sans retaper la commande. On peut en effet utiliser une syntaxe qui s'inspire de la Programmation Orientée Objet :

```
objet::attribut:=valeur de l'attribut.
```

Exemple 11
Reprenons l'exemple 8 de la section précédente et ajoutons des options.

```
[ g::Header:=''Tracé d'une tangente'':
[ g::Legend:=''sin(x)*x'':
[ p::Color := RGB::Black:
[ p::PointSize := 2.0*unit::mm:
[ t::Color := RGB::Red:
[ plot(g,p,t,GridVisible=TRUE,SubgridVisible=TRUE,TicksNumber=Low,GridLineStyle=Dashed):
```



1.7 La couleur sous Mupad

La couleur est définie à l'aide des attributs `Color`, `PointColor`, `LineColor` ou `FillColor` suivant le type de commande utilisée (voir l'aide).

Dans `MuPAD`, la couleur est exprimée dans la base RGB ou RGBa.

RGB : Acronyme de Red-Green-Blue pour Rouge-Vert-Bleu. D'après le principe de la Trichromie, toute couleur se décompose en trois composantes RGB. Le blanc contient autant de rouge, de vert et de bleu tandis que le jaune contient autant de rouge et de vert mais pas de bleu. La valeur d'une couleur s'écrit :

```
RGB::[r,g,b]
```

où les composantes r, g, b sont comprises entre 0 et 1. Si l'on veut qu'une figure soit jaune et d'intensité maximale on écrira :

```
[ fig::Color:=RGB::[1,1,0]
```

NB : Certaines couleurs sont prédéfinies. Par exemple le jaune se note aussi `RGB::Yellow`.

RGBa : On peut aussi définir une transparence pour chaque figure puisque l'on peut être amené à les superposer. C'est pourquoi on définit un paramètre d'opacité **a** compris entre 0 et

1 (a car la transparence est assurée à l'aide d'une couche dite « couche alpha »). Pour a=0, la figure est complètement transparente c'est-à-dire invisible ; pour a=1 elle est opaque et pour $0 < a < 1$, la figure est semi-transparente.
Par exemple si l'on veut qu'une figure soit jaune avec une transparence de 50%, on écrira :

`[ fig::Color:=RGB::[1,1,0,0.5]` ou `[ fig::Color:=RGB::Yellow.[0.5]`

`FillColorFunction`

Cette option permet de paramétrer la couleur de chaque pixel en fonction des variables (x,y,z etc...).

Exemple 12
Traitons l'exemple du tracé de la figure de diffraction par une pupille carrée. Commençons par définir la fonction donnant la répartition de l'intensité lumineuse sur un écran (L'intensité correspond en réalité à $f(x,y)^2$ mais dans ce cas on ne voit pas les taches secondaires de diffraction).

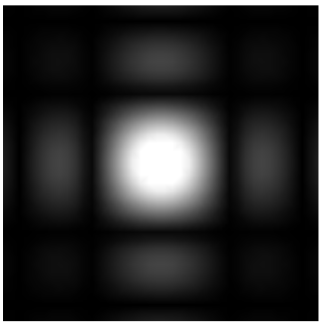
```
[ f:=(x,y)->abs(sin(x)/x*(sin(y)/y))
```

//On trace un plan z=1 que l'on regarde d'un point de l'axe z. Chaque pixel est gris d'intensité proportionnel à f(x,y).

```
[ fig:=plot::Function3d(1,x=-7..7,y=-7..7,CameraDirection=[0,0,50],
 FillColorFunction=[f(x,y),f(x,y),f(x,y)],Mesh=[40,40]):
```

//On enlève le maillage, les axes et on trace.

```
[ fig::XLinesVisible:=FALSE :  fig::YLinesVisible:=FALSE :
[ fig::Axes:=None;
[ plot(fig);
```



---

2 Notions avancées

On aborde ici des notions plus avancées par le biais d'une foire aux questions.

2.1 Comment écrire du texte sur un graphique ?

La commande `plot::Text2d(''texte'',[x,y])` permet de placer un texte au point de coordonnées [x,y]. `plot::Text3d(''texte'',[x,y,z])` fait la même chose dans l'espace.

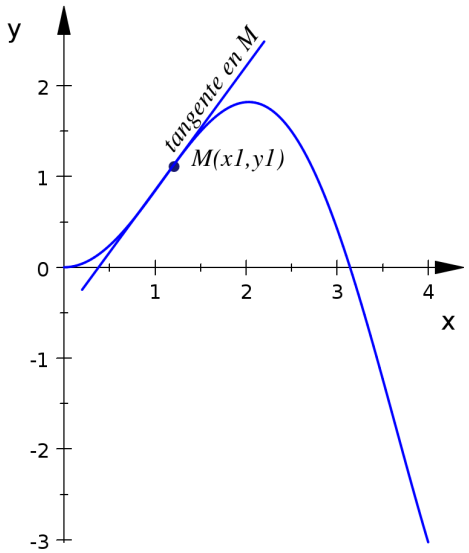
La police la taille et le style sont précisés grâce à l'attribut `TextFont` sous la forme

`TextFont=[police,taille,style]`.

Il est possible d'effectuer une rotation du texte grâce à l'attribut `TextRotation=angle`.

Exemple 13

```
//Définition d'une courbe, d'un point et d'un segment tangent à la courbe en ce point.
[ f:=x -> x*sin(x):
[ x0:=1.2: dx := 1:
[ g:=plot::Function2d(f(x), x = 0..4):
[ p:=plot::Point2d(x0,f(x0)):
[ t:=plot::Line2d([x0-dx,f(x0)-f'(x0)*dx],[x0+dx,f(x0)+f'(x0)*dx]):
//Définition des textes et tracé.
[ M0:=plot::Text2d(''M(x1,y1)'',[x0+0.2,f(x0)],TextFont=[''Times New Roman'', 10,Italic]):
[ T1:=plot::Text2d(''tangente en M'',[x0,f(x0+0.1)],TextRotation=arctan(f'(x0)),
 TextFont=[''Times New Roman'',10,Italic]):
[ plot(g,p,t,M0,T1,Scaling=Constrained);
```



2.2 Comment tracer rapidement, le graphe d'une famille de fonctions ?

Imaginons que l'on veuille tracer -sur la même scène graphique- certaines courbes d'une famille de courbes définies par $\{f_n(x)\,;\, n = a..b\}$. Si l'on veut tracer f_a, f_{a+2}, ..., f_{b-2}, f_b, il est alors pratique d'utiliser la syntaxe

```
[ fcts:=(f.n:=...)$n=a..b step 2
```

Exemple 14
On traite l'exemple de la fonction

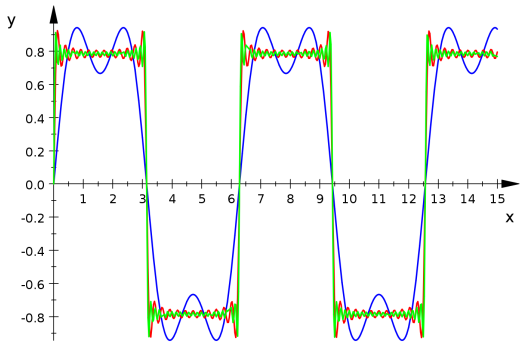
$$f_n\,:\, x \to \sum_{k=0}^{n} \frac{\sin((2k+1)x)}{2k+1}$$

qui a la propriété de converger vers la fonction créneau lorsque $n \to \infty$ (série de Fourier). On cherche à tracer f_1, f_{11}, f_{21}.
//définition de la famille de fonctions

```
[ famille:=(f.n:=sum((1/(2*k+1))*sin((2*k+1)*x),k=0..n)) $ n=1..21 step 10:
```

//tracé

```
[ plotfunc2d(famille,x=0..15,LegendVisible=FALSE);
```



2.3 Comment représenter les solutions d'équations différentielles ?

La commande `plot::Ode2d` permet de représenter graphiquement les solutions d'une équation différentielle de la forme

$$\frac{\mathrm{d}}{\mathrm{d}t}Y = F(t, Y)$$

avec Y un **nombre réel** ou un **vecteur**. Rappelons qu'une équation différentielle du second ordre du type $\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = f(t, x, \frac{\mathrm{d}x}{\mathrm{d}t})$ peut se mettre sous la forme ci-dessus si l'on pose

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} x \\ \frac{\mathrm{d}x}{\mathrm{d}t} \end{bmatrix} \text{ et } F = \begin{bmatrix} Y_2 \\ f(t, Y_1, Y_2) \end{bmatrix}$$

Pour une équation différentielle d'ordre n, F représente un vecteur fonction de t et des $Y_n = \frac{d^n x}{dt^n}$, et Y0 est le vecteur Y pour $t = t_0$. La commande suivante permet de tracer (par défaut) les valeurs des Y_n en différents instants t_0, $t_0 + dt$, ..., t_1. Les points sont reliés par une courbe spline (par défaut).

```
plot(plot::Ode2d(F, [Automatic,t0,t1,dt], Y0));
```

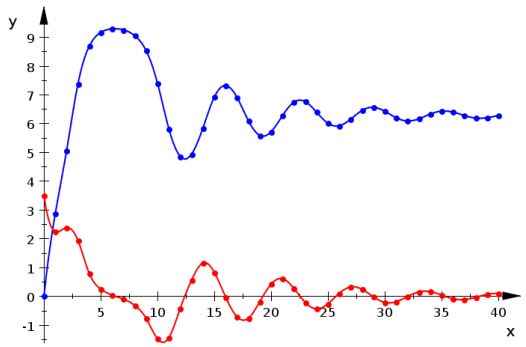
Exemple 16

Cherchons à représenter l'évolution d'un pendule simple amorti (pulsation propre $\omega_0 = 1\,\mathrm{rad.s^{-1}}$ et facteur de qualité $Q = 5$) dont l'angle obéit à l'équation différentielle

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2}(t) = -\sin\theta(t) - \frac{1}{5}\frac{d\theta}{dt}(t)$$

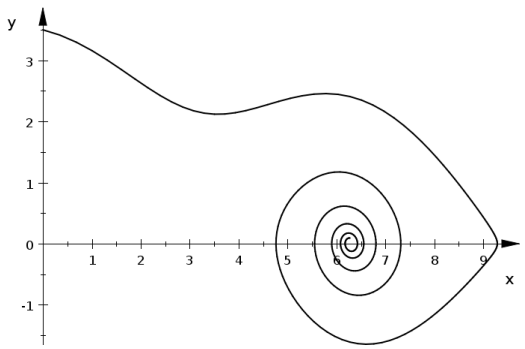
avec comme condition initiale $\theta_0 = 0\,\mathrm{rad}$ et $\frac{d\theta}{dt}|_0 = 3,5\,\mathrm{rad.s^{-1}}$.

```
//définition de F
[ F := proc(t, Y)  begin [Y[2],-sin(Y[1])-0.2*Y[2]] end_proc:
//conditions initiales
[ Y0 := [0,3.5]:
//tracé de θ(t) et θ̇(t)
[ plot(plot::Ode2d(F, [Automatic,0,40,1],Y0));
```



Si l'on veut représenter les solutions dans un autre espace que $[t, Y_n]$, on peut utiliser des projecteurs que l'on définit comme suit : $G : (t, Y) \to [x(t, Y), y(t, Y)]$. L'appel de ce projecteur permettra d'obtenir les points solutions dans l'espace $[x, y]$.

Exemple 17
Dans notre exemple du pendule simple, on peut par exemple chercher à obtenir la trajectoire de phase dans l'espace des phases $[\theta, \dot{\theta}]$. Dans ce cas, on écrira :
//définition du projecteur
```
[ G1 := (t, Y) -> [Y[1], Y[2]]:
```
//tracé
```
plot(plot::Ode2d(F,[Automatic,0,40,0.1],Y0,[G1,Style=Lines,Color=RGB::Black]));
```



`StepSize` : l'option `StepSize=h` fixe le pas utilisé par la méthode numérique. Attention, cette notion est sans rapport avec le dt !

2.4 Comment tracer une carte de champ vectoriel ?

Considérons un champ vectoriel $\overrightarrow{A}(x,y) = \begin{pmatrix} A_x(x,y) \\ A_y(x,y) \end{pmatrix}$ dont on cherche à tracer la carte de champ c'est-à-dire, un ensemble (plus ou moins dense) de vecteurs $\overrightarrow{A}(M)$ en différents points $M(x,y)$. La commande dédiée aux cartes de champ vectoriel 2D s'appelle `plot::VectorField2d`. La syntaxe est la suivante :

```
plot(plot::VectorField2d([Ax,Ay],x=x1..x2,y=y1..y2,Mesh=[N,N]));
```

ce qui permet d'obtenir un réseau de N^2 vecteurs régulièrement espacés. L'option `ArrowLength` détermine comment la longueur des flèches varie avec la norme du vecteur. Il y a trois mode possibles : `Fixed`, `Logarithmic` ou `Proportional`.

Exemple 18
Nous souhaitons représenter la carte de champ du vecteur

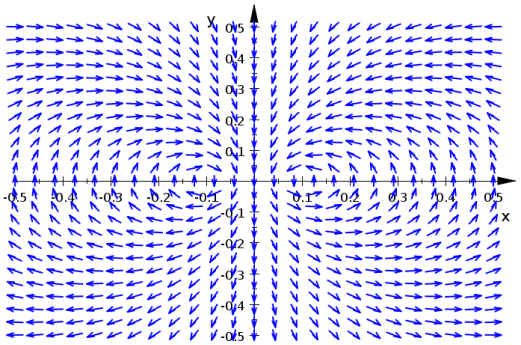
$$\overrightarrow{B} = \overrightarrow{a}(x - 0.1, y) - \overrightarrow{a}(x + 0.1, y)$$

avec

$$\overrightarrow{a}(x, y) = \begin{pmatrix} -\frac{y}{(x^2+y^2)} \\ \frac{x}{(x^2+y^2)} \end{pmatrix}$$

Le code est le suivant :

```
[ ax:=(x,y)->-y/(x^2+y^2) : ay:=(x,y)->x/(x^2+y^2) :
[ field:=plot::VectorField2d([ax(x-0.1,y)-ax(x+0.1,y),ay(x-0.1,y)-ay(x+0.1,y)],
 x=-0.5..0.5, y=-0.5..0.5,Mesh=[25,25],ArrowLength=Fixed): plot(field);
```



2.5 Comment faire des animations ?

Chaque commande graphique attend un certain nombre de plages de valeurs sous la forme `x=xmin..xmax`. Par exemple la commande `plot::Function2d(f(x), x = xmin..xmax)` attend une plage de valeurs. Si vous ajoutez une nouvelle plage de valeur de la forme `a=amin..amax`, `MuPAD` interprétera `a` comme un *paramètre d'animation* : `MuPAD` produira (par défaut) 50 images correspondantes au 50 valeurs de `a` régulièrement (par défaut) espacées dans l'intervalle `amin..amax`.

Exemple 18
(Pour voir l'animation rendez vous sur le site Web : `perso.ensc-rennes.fr/jimmy.roussel/` ▷ Outils ▷ Faire une animation)

```
// L'objet c est notre première une animation : ici, tmax est un paramètre. on obtient le tracé
progressif de la courbe de Lissajous.
[ x:= t -> sin(3*t) : y:= t -> cos(5*t) :
[ c := plot::Curve2d([x(t), y(t)], t=0..tmax, tmax=0..2*PI):
//L'objet p est une seconde animation : en effet, [x(t),y(t)] repère un point dont la position
dépend du paramètre t. il s'agit d'un point qui parcours une courbe de Lissajous.
[ p:= plot::Point2d([x(t), y(t)], t=0..2*PI) :
//Tracé des deux animations en même temps.
[ plot(c, p);
```

2.6 Y a t-il possibilité d'exporter les graphiques et animations ?

Exportation des graphiques Il est effectivement possible d'exporter les graphiques. Pour cela, après avoir cliqué sur un objet graphique, allez dans le Menu ▷`File`▷`Export Graphics...`. Une boîte de dialogue vous propose différents formats. Préférez les formats vectoriels (EPS,SVG...) aux formats bitmaps (PNG,BMP,JPEG).

Exportation des animations Les animations peuvent également être exportées. Les formats GIF Animé, JVX, etc. sont proposés.

Option pour exporter Au lieu d'afficher un graphique à l'écran, on peut lui demander de l'exporter directement dans un fichier en utilisant l'option `OutputFile`. La syntaxe est la suivante :

```
plot(figure, OutputFile=''mafigure.jpg''):
```

Table des matières