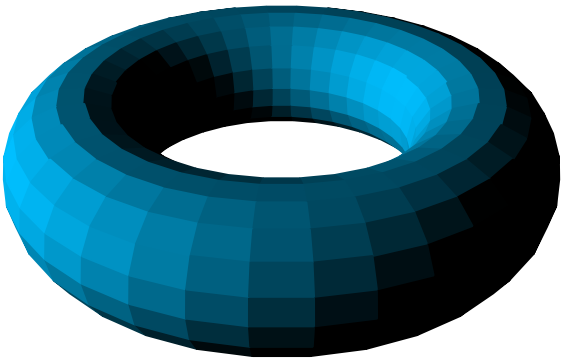
Dessiner avec postscript



jps2ps : guide de l'utilisateur

par Jean-Paul Vignault
Groupe des Utilisateurs de Linux Poitevins (GULP)
(**jpv@melusine.eu.org**)
19 Janvier 2008

Table des matières

III - Commandes de tracés supplémentaires

IV - Compléments

V - Tracés en 3d

VI - Nœuds, arbres

VII - Contributions – Bibliographie

Annexe I : résumé des opérateurs Postscript

Annexe II : Index des commandes jps

1. Introduction

Tout le monde a entendu parler du format postscript, devenu peu à peu un standard dans le monde de l'imprimerie et de l'édition. Il est en général moins connu que c'est également un vrai langage de programmation doté de toutes les structures classiques (boucles, variables typées, tests, appels récursifs, etc. . .) ainsi que de primitives graphiques puissantes.

L'idée directrice du travail présenté ici a été d'écrire, en postscript, un certains nombre de commandes enrichissant le langage pour permettre de façon plus aisée la description de figures mathématiques simples. L'ensemble de ces commandes et macros-commandes constitue donc ce que l'on peut appeller un *format*[*] au-dessus de postscript, format que je désignerai sous le nom de *format jps.*

Au final nous obtenons un petit logiciel de dessin mathématique produisant des fichiers postscripts *humainement lisibles* (ce qui garantit une certaine pérennité puisque ces fichiers restent facilement modifiables à l'aide d'un simple éditeur de texte) et tel que l'utilisateur manipule directement le langage postscript (à travers le format proposé).

Les fichiers produits peuvent être facilement réutilisés dans la plupart des traitements de texte standards, et pour ceux qui ont la chance d'utiliser (LA)TEX,une option permet d'obtenir toutes les indications nécessaires pour une réutilisation simple avec PSTricks (du moins pour les cas usuels).

Lors de l'écriture de ce logiciel, le langage PSTricks s'est rapidement imposé comme modèle. C'est pourquoi, en règle générale, j'ai essayé de reprendre la syntaxe proposée dans les diverses extensions de ce langage[**]

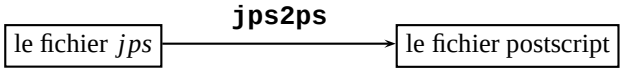
Dans certains cas, le code PSTricks est directement utilisable par jps. C'est ainsi que le code de *pst-grad.pro* de W. R. Lee ou celui de *pst-math.pro* de Christophe Jorssen ont été intégrés au format.

Le langage Metapost sert lui aussi de référence depuis le début de ce travail. Là encore, lorsque c'était possible, je me suis largement inspiré de sa syntaxe.

Ce manuel ne comporte pas encore de bibliographie faute de temps. Je m'en excuse par avance auprès de tous les auteurs que je dois citer.

2. Utilisation de `jps2ps`

Pour obtenir un dessin, on commence par le décrire dans un fichier séparé avec une syntaxe spécifique (le fichier *source jps*), puis on fait lire ce fichier au script qui produit l'image désirée.



Le langage utilisé pour le fichier *jps* est Postscript lui-même, enrichi d'un certain nombre de macros (le format jps). L'essentiel du travail de **`jps2ps`** consiste à regarder quelles sont les macros utilisées afin d'insérer leurs définitions dans le fichier final, avant d'insérer le fichier jps lui-même qui consistue la seule partie génératrice de dessin.

La syntaxe générale est

`jps2ps` [*options*] *source*

avec

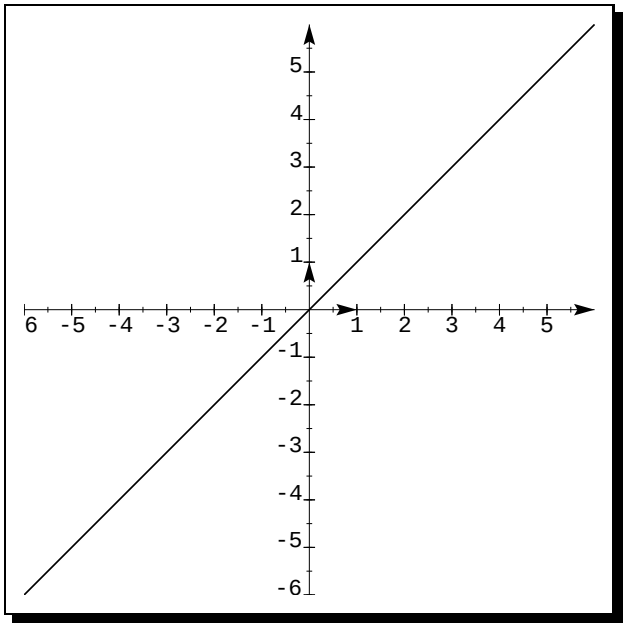
source	est le fichier source. Si celui-ci est absent, le script lira son entrée standard, et si le nom de fichier ne comporte pas d'extension, le script rajoutera lui-même le suffixe *.jps* et cherchera le fichier *source.jps*
`-a`	indique que la sortie doit se faire dans le fichier *source.ps*
`-v`	affiche le numéro de version
`-i` *file*	charge le fichier *file* avant de lire le fichier source. Cette option permet notamment le chargement d'un ou plusieurs fichiers de macros utilisateurs.
`-o` *file*	indique que la sortie doit se faire dans le fichier *file*
`-s` *source*	indique que le fichier source est *source*
`-pst`	affiche les indications utiles pour utiliser le fichier postscript produit dans l'environnement PSTricks pour TEX.
`-lpst`	affiche les indications utiles pour utiliser le fichier postscript produit dans l'environnement PSTricks pour LaTEX.

Par exemple, si le fichier *jps* se nomme **`essai.jps`**, la commande **`jps2ps.pl -a essai.jps`** produira le fichier postscript et le sauvegardera dans le fichier **`essai.ps`**.

Si les options **`-a`** et **`-o`** ne sont pas utilisées, la sortie se fera sur la sortie standard.

Ainsi, voici un fichier *jps* de 3 lignes accompagné de l'image produite :

[*] de la même façon que **`plain`** ou **`latex`** sont des formats au-dessus de TEX.
[**] voir **`http://melusine.eu.org/syracuse/pstricks/`**



source jps

```
tracerepere
marks
{} courbe
```

Il n'est pas prévu d'interface graphique. Pour utiliser ce script, il vous faudra un éditeur pour créer vos fichiers *jps*, ainsi qu'un visualisateur de fichiers postscript.

Il existe cependant une interface HTML pour l'utilisation en réseau, ce qui permet de piloter le script à travers des formulaires. Efficace en réseau local, il est également installé sur l'Internet, même si les temps de transmissions rendent son utilisation difficile (voir **`http://melusine.eu.org/syracuse/bbgraf/`**).

Le but de ce document est de décrire le format *jps*. Tout ceci n'est encore qu'à un stade très expérimental. Merci de votre indulgence.

I - Quelques notions de Postscript

Vous trouverez ici quelques notions concernant le langage PostScript. Néammoins, je n'ai aucune compétence particulière et les notions présentées ici ne le sont que de façon très rudimentaire et incomplète, voire partiellement inexacte.

Le lecteur intéressé se reportera à un ouvrage dédié.

Pour ma part, mes connaissances viennent du *Manuel de référence du langage PostScript*, deuxième édition, *Adobe Systems Incorporated*, Trad. Denys Bondeville, mai 1992, chez Addison-Wesley (ISBN 2-87908-009-6). C'est ce manuel qui a très largement inspiré la rédaction de ce chapitre.

1. Insérer des commentaires dans le fichier

Sur une ligne de fichier donnée, le caractère **%** est ignoré, ainsi que toute la fin de la ligne. Cette propriété de Postscript permet d'insérer des commentaire dans les fichiers *jps*, afin d'améliorer leur lisibilité et leur éventuelle modification ultérieure.

2. Gestion de la pile

2.1 - Principe

Le fonctionnement de Postscript est basé sur l'utilisation de *piles* de type *Last In First Out* (LIFO). L'une d'entre elles, la pile des opérandes, est particulièrement importante : lorsque le langage exécute un opérateur qui a besoin d'arguments, celui-ci va les chercher dans la pile d'opérandes, en commençant par le dernier entré. De la même façon, lorsqu'une procédure produit un résultat, elle dépose celui-ci sur la pile d'opérandes.

Par exemple, lorsque Postscript rencontre la séquence d'instructions : **`2 3 add`**, celui-ci dépose d'abord le nombre 2 sur la pile, puis le nombre 3, puis exécute l'opérateur d'addition **`add`** avec les arguments 2 et 3, et dépose enfin le résultat 5 sur la pile. Si la pile était vide avant le début de la séquence, elle ne contiendra que le nombre 5 après l'exécution de celle-ci.

2.2 - Quelques opérateurs de gestion de pile

Il existe divers opérateurs pour manipuler les objets dans la pile d'opérandes. En voici quelques uns :

- **`pop`** : enlève l'élément au sommet de la pile
- **`dup`** : duplique l'objet au sommet de la pile
- **`exch`** : échange la place des 2 éléments au sommet de la pile
- **`copy`** : duplique des portions de la pile dopérandes
- **`roll`** : traite une portion de la pile comme une file circulaire
- **`clear`** : efface le contenu de la pile

3. Quelques objets Postscripts

3.1 - Les nombres

Les nombres dans Postscript comportent des entiers positifs ou négatifs, comme **`123`**, **`-76`** ou **`0`**, des réels, comme **`-.01`**, **`7.5`** ou **`12.6E-5`**, et des nombres basés comme **`8#1777`**, **`16#FFFE`** ou **`2#1010`**.

3.2 - Les chaînes de caractères

Deux conventions pour représenter les chaînes :

- comme un texte encadré de parenthèses **`(`** et **`)`**
- sous forme hexadécimale encadrée de **`<`** et **`>`**.

Voici quelques exemples, extraits du manuel de référence Postscript, de chaînes valides :

```
(ceci est une chaine)
(Les chaines peuvent contenir un caractère de nouvelle ligne
comme ça)
(les chaines peuvent contenir des caractères spéciaux *-&^}% et des parenthèses
si elles vont par paires () (et ainsi de suite))
()
```

Le cararactère **** (backslash) sert à *échapper* un caractère particulier. On a ainsi les correspondances suivantes :

`\n` line-feed (LF ou nouvelle ligne)

\r	Carriage Return (CR)
\t	tabulation horizontale
\b	backspace
\f	form-feed
****	backslash
\(	parenthèse ouvrante
\)	parenthèse fermante
ddd	code de caractère *ddd* (octal)

3.3 - Les noms

Tous les caractères, en dehors des délimiteurs et des espaces peut-être un nom, y compris les caractères ordinairement considérés comme étant de ponctuation. Par exemple, les noms **abc**, **Offset**, **$$**, **@pattern**, **36-15** sont valides. Il faut néammoins faire attention si le nom commence par un chiffre : **23A** est un nom valide, mais **23E1** est un nombre réel et **23#1** est un nombre basé.

Le caractère **/** (slash) précède un nom littéral. Il ne fait pas partie du nom, mais est un préfixe indiquant que le nom qui suit est un littéral.

3.4 - Les booléens

Le langage PostScript fournit les objets booléens *vrai* et *faux*. Les noms **true** et **false** sont respectivement associés à ces objets.

3.5 - Les tableaux

Les caractères **[** et **]** spécifient la construction d'un tableau. Le fragment de programme **[123 /abc (xyz)]** donne comme résultat la construction d'un objet tableau contenant l'objet entier 123, le nom littéral *abc*, et l'objet chaîne xyz. Les données entre **[** et **]** sont exécutées l'une après l'autre.

3.6 - Les procédures

Les caractères spéciaux **{** et **}** délimitent un *tableau exécutable*, autrement appelé *procédure*.

L'interpréteur n'exécute pas immédiatement une procédure mais la traite comme une donnée et la place dans la pile des opérandes. C'est uniquement lorsqu'une procédure est explicitement évoquée qu'elle est exécutée.

Un exemple d'utilisation courant : **/A {2 3} def** associe au nom *A* la procédure **{2 3}**. L'appel de *A* provoquera ensuite l'exécution de la procédure qui déposera alors les nombres 2 et 3 sur la pile.

3.7 - Création d'objets

On crée un objet en l'associant à un nom avec la commande **def**. Par exemple, les commandes suivantes :

```
/chaine (ceci est une chaine) def
/tableau [1 2 3] def
/A {2 3} def
/moyenne {add 2 div def} def
```

créent 4 objets. Les 2 premiers sont des constantes et les 2 derniers sont des exécutables. À la suite de ces associations, lorsque l'interpréteur Postscript rencontre les commandes **chaine** ou **tableau**, il dépose sur la pile la chaîne de caractères ou le tableau respectivement ; lorsqu'il rencontre les commandes **A** ou **moyenne**, il exécute les procédures associées. La procédure *A* dépose les nombres 2 puis 3 sur la pile, et la procédure *moyenne* prend 2 nombres sur la pile, les additionne, puis divise cette somme en 2 et dépose le résultat sur la pile.

II - Les bases du format jps

Dans ce chapitre, on décrit l'ensemble des macros disponibles dans le format *jps*. Les descriptions se font souvent par l'exemple, et on se reportera aux annexes pour une description plus rigoureuse.

1. Différences par rapport aux versions précédentes

Dans la version 0.014 : apparition de l'objet chemin paramétré et des flèches courbes « à la Metapost ».

Dans la version 0.013 : possibilité d'appliquer des transformations aux nœuds d'arbres, qui peuvent contenir des objets de l'environnement picture. La gestion des tangentes se fait maintenant en valeur approchée, avec gestion des demi-tangentes. Bien que cela reste possible, on ne transmet plus le nom de la fonction mais un exécutable.

La version 0.011 voit les premières commandes 3d concoctées par Philippe Saadé.

La version 0.07*b* voit apparaitre les commandes **`bubblesort`**, **`moyenne`**, **`mediane`**, **`variance`**, **`covariance`**, **`ecarttype`**, **`correlation`**, ainsi que de nouvelles options sur la ligne de commande (**`-o`**, **`-v`**, **`-s`**).

Depuis la version 0.07*a*, la commande **`Poisson`** change de syntaxe.

Depuis la version 0.07, le script supporte l'option **`-i`** permettant d'insérer ses propres fichiers de macros. Les commandes **`#tex#`**, **`#inc#`** et **`#rpn#`** font leur apparition dans le format *jps*.

Depuis la version 0.06*d*, le fonctionnement interne change pour les labels TEX. La *baseline* est désormais mieux reconnue, mais il faut maintenant éviter les **`$$`** pour les formules simples.

Depuis la version 0.06*b*, les commandes liées à la fonte **`Courier`** perdent un **`r`**. Les commandes **`setCourrier`**, **`setCourrierItalic`**, **`setCourrierBold`**, et **`setCourrierBoldItalic`** sont maintenant obsolètes et respectivement remplacées par **`setCourier`**, **`setCourierItalic`**, etc. . .

Depuis la version 0.06*a*, la commande **`tripointarcarrow`** est renommée en **`tripointarc`**.

Depuis la version 0.05*g*, le paramètre *curvelinewidth* est obsolète. On n'utilise désormais plus que la paramètre postscript *linewidth*.

Depuis la version 0.05*e*, les seules instructions analysées par le script concernant les *xrange* et *yrange* sont **`setxrange`** et **`setyrange`**.

Depuis la version 0.05, les commandes **`courbe`**, **`courbeparam`**, **`hachcourbe`**, **`fillcourbe`** et dérivées changent de syntaxe : la fonction passée en argument l'est via un exécutable plutôt qu'une chaîne de caractères. Idem pour les commandes **`plot`** et **`relief`**.

2. Paramètres et arguments

Pour les différentes commandes présentées, on désignera par *argument* un objet transmis à l'opérateur ou la procédure, et par *paramètre* ou *variable* une variable globale.

• Par exemple, la commande d'addition *add* prend deux nombres en argument, et la commande de numérotation *marks* ne prend aucun argument mais utilise plusieurs paramètres comme *xmin*, *xmax*, etc. . . .

Pour montrer les arguments, leur type, et le résultat sur la pile, on notera

nombre$_1$ *nombre*$_2$ **`add`** *nombre*$_3$ $\longrightarrow$ additionne *nombre*$_1$ et *nombre*$_2$

$-$ **`marks`** $-$ $\longrightarrow$ numérote les graduations sur les axes *Ox* et *Oy*

On voit ainsi que *add* prend deux nombres en argument et en dépose un sur la pile, alors que *marks* n'a aucune influence sur la pile.

• Quand aux paramètres du format *jps*, il peuvent être *simples* (comme *hangle*) ou *complexes* (comme *arrowscale*). Un paramètre simple est défini comme une constante Postscript, alors qu'un paramètre complexe est défini comme une procédure Postscript. Dans la plupart les cas, on peut modifier ce paramètre soit directement en Postscript, soit en utilisant la commande *set* associée.

Par exemple, pour affecter le paramètre simple *hangle* à 30 (angulation en degrés pour les hachures), on peut taper

`/hangle 30 def` ou **`30 sethangle`**

et pour affecter le paramètre complexe *arrowscale* à $(2, 2)$, on peut taper

`/arrowscale {2 2} def` ou **`2 2 setarrowscale`**.

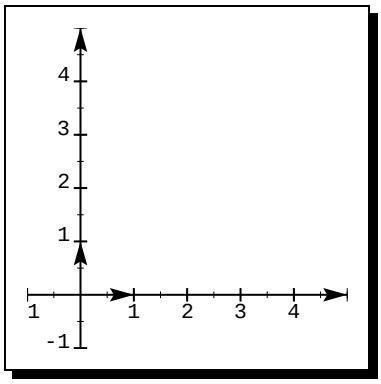
3. Géométrie de l'image produite

3.1 - La fenêtre de dessin (espace utilisateur)

Pour se repérer dans la fenêtre de dessin, on y place un repère cartésien, que je désignerai sous le nom de *repère jps*. Ce repère est défini par quatre paramètres principaux : *xmin*, *xmax*, *ymin* et *ymax*, fixés respectivement à $-5, 5, -5, 5$

par défaut, paramètres que l'on peut modifier en utilisant les opérateurs **`setxrange`** et **`setyrange`**. Si le format de l'image n'est pas complètement déterminé (autrement dit si le paramètre *format*, et au moins un des paramètres *width* et *height* n'ont pas été fixés par l'utilisateur), et s'il n'y a pas d'indications supplémentaires, le script choisira un format d'image tel que le repère *jps* soit orthonormé.

Par exemple, voici un fichier *jps*, et la figure générée :



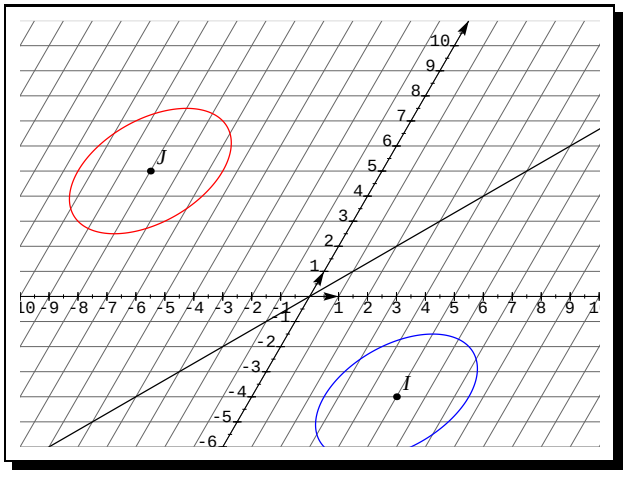
source jps

```
20 setxunit
-1 5 setxrange
-1 5 setyrange
tracerepere
marks
```

On peut utiliser plusieurs fois les commandes **`setxrange`** ou **`setyrange`** dans un même fichier, mais il faut savoir que la première occurence de chacune d'entre elle sera interprétée par le script **`jps2ps`**. Pour des raisons techniques (voir paragraphe suivant), il est nécessaire que ces premières occurences soient chacune seule sur une ligne, avec un argument strictement numérique. Ces contraintes sont levées dès la deuxième occurence, qui ne sera interprétée que par postscript.

Pour avoir un repère non orthonormé, la commande **`setxyrapport`** permet d'indiquer le rapport entre les unités sur *Ox* et les unités sur *Oy*. Ainsi, la commande **`2 setxyrapport`** indique que l'unité sur l'axe *Ox* est le double de celle sur *Oy*.

Pour avoir un repère non orthogonal, la commande **`setangle_repere`** permet de spécifier l'angle en degrés que fait la demi-droite $(O,\vec{\imath})$ avec la demi droite $(O,\vec{\jmath})$ (où $(O,\vec{\imath},\vec{\jmath})$ est la base canonique associée au repère *jps*). Là encore, cette commande doit être interprétée par le script **`jps2ps`** et elle subit les mêmes contraintes que les commandes **`setwidth`**, etc. . .



source jps

```
%% les 3 lignes qui suivent
%% sont interpretees par jps2ps
%% ==> 1 commande par ligne, et des
%% arguments strictement numeriques
60 setangle_repere
-10 10 setxrange
-6 11 setyrange

%% la ligne suivante n'est
%% interpretee que par postscript
%% ==> tout est autorise
-10 6 sub 10 5 add setxrange
masque
/I {5 -4} def
/J {-8 5} def
quadrillage
marks
tracerepere
{} courbe
[I J] points
bleu I 2.5 cercle
rouge J 2.5 cercle

noir setTimesItalic
   (I) I urtext
   (J) J urtext
```

En résumé :

> x_1 x_2 **`setxrange`** $-\longrightarrow$ Affecte respectivement les valeurs x_1 et x_2 à *xmin* et *xmax* (amplitude horizontale de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

y_1 y_2 **`setyrange`** $-\longrightarrow$ Affecte respectivement les valeurs y_1 et y_2 à *ymin* et *ymax* (amplitude verticale de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

α **`setxyrapport`** $-\longrightarrow$ Instruction destinée au script *jps2ps*. Elle indique que le rapport entre l'unité sur *Ox* et l'unité sur *Oy* est α. Attention, α doit être lisible en clair par le script

α **`setangle_repere`** $-\longrightarrow$ Instruction destinée au script *jps2ps*. Elle indique que l'angle entre les axes *Ox* et *Oy* est de α degrés. Attention, α doit être lisible en clair par le script

Et on a les variables :

- *xmin* : borne inférieure sur l'axe *Ox*. **valeur par défaut :** -5
- *xmax* : borne supérieure sur l'axe *Ox*. **valeur par défaut :** 5
- *ymin* : borne inférieure sur l'axe *Oy*. **valeur par défaut :** -5
- *ymax* : borne supérieure sur l'axe *Oy*. **valeur par défaut :** 5

3.2 - Taille et format de l'image (la Bounding Box)

Pour une image donnée, l'interpréteur postscript possède son propre système de coordonnées (dans un repère orthonormal). Celui-ci est entièrement déterminé par ce que l'on appelle la *Bounding Box*, qui est une boîte contenant complètement le dessin, ce qui permet sa manipulation par des logiciels externes. Cette boîte est définie par la donnée de 4 nombres positifs correspondant respectivement aux coordonnées du coin inférieur gauche et du coin supérieur droit. Ces coordonnées sont exprimées en points postscript, ce qui nous donne la taille et le format de l'image.

L'unité postscript fait un $1/72$ de pouce, et un pouce fait $25,4$ mm. Moralité : il faut $10 \times 72/25,4 \approx 28,346\,457$ points postscript pour faire 10 mm.)*

La Bounding Box est toujours calculée par le script **`jps2ps`**, alors que la plupart des autres calculs sont effectués par l'interpréteur postscript. Une des conséquences en est la syntaxe particulièrement rigide des quelques commandes concernant le format de l'image produite (**`setxrange`**, **`setyrange`**, **`setwidth`**, **`setheight`**, **`setxyrapport`**, **`setformat`**, **`setrotate`**, **`setborder`** et **`setangle_repere`**). En effet, ces commandes sont lues puis traduites par le script afin d'en extraire les informations utiles, et mon niveau actuel concernant les expressions régulières en Perl ne me permet pas de faire des prouesses. En particulier, chacune de ces commandes doit se trouver seule sur une ligne, et les arguments de ces commandes doivent être strictement numériques (pas de résultat de calcul ou de variable). Pour toutes ces commandes, merci de vous conformer aux exemples donnés dans ce document ou sur le site **`http://melusine.eu.org/syracuse/bbgraf/`**.

Le format *jps* prévoit deux façons de définir le format de l'image produite : une méthode globale où l'utilisateur définit explicitement les dimensions voulues, et une méthode locale (souvent plus pratique) où l'utilisateur définit les échelles sur les axes et l'amplitude sur chacun des axes.

Par défaut c'est la méthode globale qui est employée, de telle façon que la plus grande des dimensions largeur/hauteur soit de 250 points postcript.

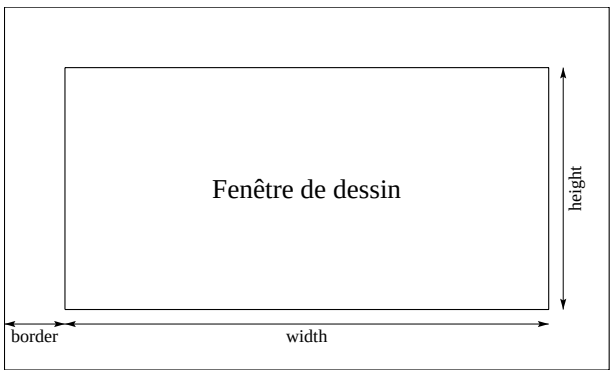
3.2.1 - Définir les unités

Les commandes **`setxunit`** et **`setyunit`** servent à indiquer explicitement l'échelle sur chacun des axes. Ainsi, la commande **`20 setxunit`** va affecter 20 points postcript par unité sur l'axe *Ox*.

Dès que l'une des échelles est fixée, l'autre est calculée en tenant comptre du paramètre *xyrapport* (égal à 1 par défaut).

La taille de l'image est alors déterminée par les amplitudes *xrange* et *yrange*.

3.2.2 - Imposer les dimensions externes



* Merci à Michel Goudinot pour ces précisions

Le format de l'image est déterminé par 3 paramètres : *width* et *height* qui correspondent à la fenêtre de dessin, et *border* qui indique la taille (en points postscript) de la bordure blanche à insérer autour de la fenêtre de dessin. La variable *border* est initialisée à $7,5$ par défaut.

Les commandes **`setwidth`**, **`setheight`** et **`setborder`** permettent à l'utilisateur d'imposer la valeur de tel ou tel paramètre. Ainsi la séquence **`400 setwidth 400 setheight`** (sur 2 lignes distinctes. . .) va imposer une fenêtre de dessin aussi large que haute, de côté 400 points postscript. Si l'utilisateur ne fixe pas les valeurs de *width* et *height*, et si les échelles sur *Ox* et *Oy* ne sont pas imposées, alors le script calculera les valeurs des variables *width* et *height* en imposant que la plus petite des deux soit égale 250, et en tenant compte des contraintes sur la fenêtre de dessin (amplitudes sur *Ox* et *Oy* et rapport des échelles).

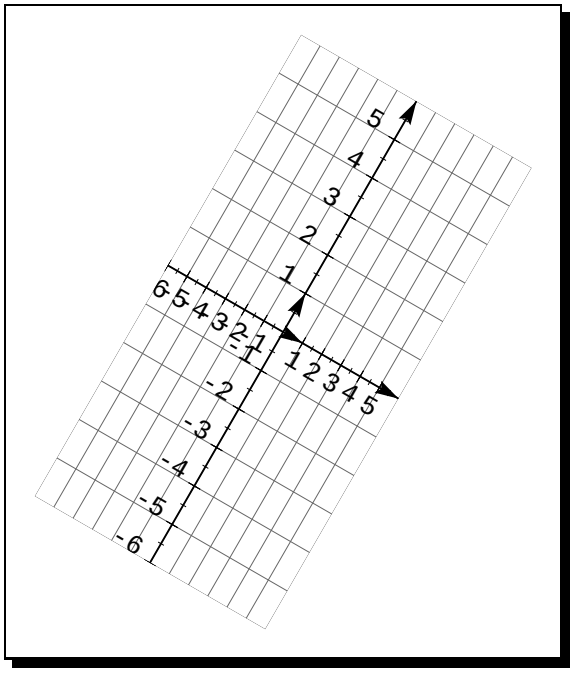
Pour aider à la gestion des variables *height* et *width*, on dispose également de la commande **`setformat`**, qui permet d'imposer un rapport entre les dimensions horizontales et verticales (hors bordure) de l'image produite. Ainsi, la commande **`2 setformat`** permettra d'obtenir une image dont le rapport $width/height$ sera égal à 2.

Enfin la commande **`setrotate`** permet de faire effectuer une rotation à l'image produite. Elle prend un argument numérique indiquant l'angle en degré de cette rotation, dont le centre est le coin inférieur gauche de l'image produite. Si on utilise cette option, la Bounding Box est recalculée pour contenir la totalité de l'image « rotationnée ».

Voyons maintenant les ordres de priorité. Une fois récupéré les informations dans le source *jps*, l'algorithme est le suivant :

- si *width* et *height* sont définis, c'est fini.
- si on a *format* et *width*, on calcule *height* et c'est fini.
- si on a *format* et *height*, on calcule *width* et c'est fini.
- si on a seulement *format*, le script choisit *height* et *width* en imposant qu'ils soient tous deux supérieurs ou égaux à 250, et c'est fini.
- sinon on calcule les données manquantes en utilisant les valeurs fournies pour le repère *jps* et les valeurs par défaut (voir le paragraphe sur la fenêtre de dessin).

Une fois cela accompli, on limite *width* et *height* à 1 000 pour éviter des images trop grandes lors de l'utilisation via le web (la convertion au format jpeg alourdit sérieusement le fichier), puis on effectue éventuellement la rotation spécifiée par **`setrotate`**.



source jps

```
-30 setrotate
200 setheight
.5 setxyrapport
quadrillage
.7 setlinewidth
tracerepere
marks
```

En résumé :

u **`setxunit`** $-\longrightarrow$ Spécifie, en nombre de points postcripts par unité, l'échelle sur l'axe *Ox*. La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

v **`setyunit`** $-\longrightarrow$ Spécifie, en nombre de points postcripts par unité, l'échelle sur l'axe *Oy*. La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

b **`setborder`** $-\longrightarrow$ Indique, en points postripts, a taille de la bordure entourant l'image. Cette instruction est interprétée par le script *jps*2*ps* pour déterminer la BoundingBox.

L **`setwidth`** $-\longrightarrow$ Affecte la valeurs *L* à la variable *width* (taille horizontale, en points postscript, de la

fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

ℓ **setheight** $-\!\!\longrightarrow$ Affecte la valeurs ℓ à la variable *height* (taille verticale, en points poscripts, de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

k **setformat** $-\!\!\longrightarrow$ Instruction destinée au script *jps*2*ps*. Elle indique que *k* est le rapport entre largeur et hauteur de l'image. Attention, *k* doit être lisible en clair par le script

α **setrotate** $-\!\!\longrightarrow$ Instruction destinée au script *jps*2*ps*. Elle indique que l'image finale devra subir une rotation d'angle α (en degrés). Le calcul de la BoundingBox tient compte de cette rotation. Attention, α doit être lisible en clair par le script

3.3 - Les trois repères utilisés

Finalement, trois repères distincts sont utilisés :

- Le premier est le repère Postscript lié à la Bounding Box. Il s'agit d'un repère orthonormé dont l'origine est située au coin inférieur gauche de l'image (dans le cas d'une image produite par **jps2ps**). Je désignerai ce dernier sous le nom de *repère BB*. Lorsque l'on visualise une image et que l'on y lit des coordonnées à la souris, ces coordonnées sont dans ce repère BB.
- Le second est le repère *jps*. C'est le repère utilisateur du format. Il n'est pas forcément orthogonal, et les échelles sur ses axes ne sont pas forcément linéaires.
- Le dernier est un repère intermédiaire entre les précédents : il s'agit du repère BB dont l'origine a été translatée pour correspondre à l'origine du repère *jps*. Il s'agit donc d'un repère orthormé et les vecteurs unitaires sont les mêmes que pour le repère BB. Ce repère est en général utilisé pour nombres de calculs intermédiaires et je le désignerai sous la dénomination de *repère postscript.*

Quelques commandes permettent de passer d'un repère à l'autre :

$X\ Y$ **jtoppoint** $x\ y \longrightarrow$ Reçoit les coordonnées (X, Y) dans le repère *jps* et renvoie les coordonnées (x, y) dans le repère postscript

$x\ y$ **ptojpoint** $X\ Y \longrightarrow$ Reçoit les coordonnées (x, y) dans le repère postscript et renvoie les coordonnées (X, Y) dans le repère *jps*

$x\ y$ **rptojpoint** $X\ Y \longrightarrow$ Reçoit les coordonnées (x, y) dans le repère BB (*real postscript*) et renvoie les coordonnées (X, Y) dans le repère *jps*

Ces commandes servent souvent pour additionner des dimensions hétérogènes. Par exemple, la taille des fontes est connue en points postscript et lorsque l'on souhaite déplacer une légende, ce sont souvent les coordonnées du vecteur de translation dans le repère jps qui nous viennent à l'esprit. D'où les **ptojpoint** et **jtoppoint**.

Quant-à la commande **rptojpoint** elle sert à déterminer dans le repère *jps* les coordonnées d'un point lu à la souris.

4. Tracé du repère

La commande **tracerepere** génère un tracé complet du repère *jps* (les 2 axes, les graduations, sous-graduations et marques, les unités et les flèches au bout des axes). On peut également affiner et ne demander qu'une partie de ce tracé.

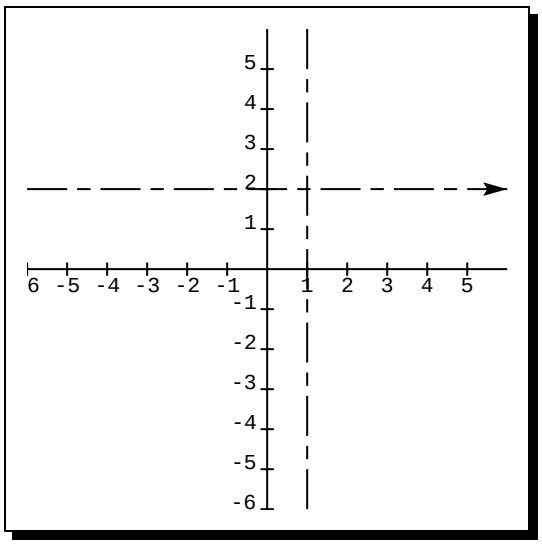
À noter que par défaut, le repère est orthonormé. Pour avoir un repère seulement orthogonal, merci de vous reporter au paragraphe « Gestion de la Bounding Box ».

4.1 - Les axes

On dispose des commandes suivantes, qui font ce que l'on pense : **traceOx**, **traceOy**, **traceaxes**. À noter que l'origine du tracé des axes n'est pas forcément le point $O(0, 0)$. On peut la changer avec la commande **setorigine**.

Pour avoir les flèches au bout des axes, on dispose des commandes **axeOxarrow**, **axeOyarrow** et **axesarrow**.

Par exemple, voici un fichier jps, et la figure générée :



source jps

```
15 setxunit
traceaxes
marks
mixte
1 2 setorigine
traceaxes
axeOxarrow
```

4.2 - Les graduations, sous-graduations, et numérotations

Pour tracer des graduations, on dispose des commandes suivantes : **`ticks`**, **`xticks`**, **`yticks`**, **`xtick`** et **`ytick`**. Les trois premières n'ont pas d'argument et tracent les graduations auxquelles on pense. Les deux dernières prennent un argument numérique, et placent une seule graduation. Par exemple, la commande **`pi xtick`** place une graduation sur l'axe *Ox* au point d'abscisse π.

Pour tracer des sous-graduations, on dispose des commandes analogues suivantes : **`subticks`**, **`xsubticks`**, **`ysubticks`**, **`xsubtick`** et **`ysubtick`**.

Pour marquer les valeurs, on dispose des commandes **`marks`**, **`xmarks`**, **`ymarks`**, **`xmark`** et **`ymark`**. Les 3 premières ne demandent pas d'argument, les 2 dernières veulent un nombre. Pour l'affichage, les commandes **`xmark`**, **`ymark`** et dérivées utilisent le contenu des macros **`xmarkstyle`** et **`ymarkstyle`**, dont les définitions par défaut respectives sont **`dctext`** et **`(-1 0) bltext`**.

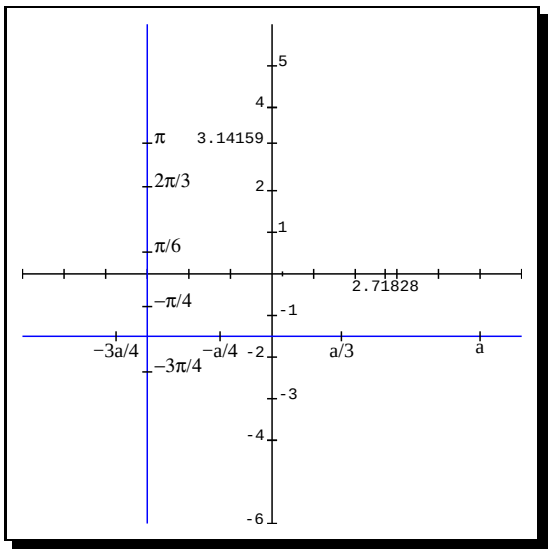
Les 9 commandes sans argument que nous venons de voir utilisent des paramètres contenant les valeurs d'incrémentation (*xmkstep*, *ymkstep*, *xtkstep*, *ytkstep*, *xsubtkstep*, *ysubtkstep*,). Par défaut celles-ci sont fixées à 1 pour les marques et graduations, et à $0,5$ pour les sous-graduations. On peut changer ces valeurs avec les commandes **`setxmkstep`**, **`setymkstep`**, **`setxtkstep`**, **`setytkstep`**, **`setxsubtkstep`** et **`setysubtkstep`** qui prennent chacune un argument numérique.

On dispose également des commandes **`setmkstep`**, **`settkstep`** et **`setsubtkstep`** qui prennent deux arguments numériques et qui permettent d'affecter les paramètres sur les deux axes à la fois. Par exemple, la séquence **`3 2 setmkstep`** affectera la valeur 3 à *xmkstep* el la valeur 2 à *ymkstep*.

La taille de fonte utilisée est la taille courante.

Les commandes **`coeff_xticks`**, **`coeff_yticks`**, **`coeff_xmarks`**, **`coeff_ymarks`**, **`trig_xmarks`**, **`trig_ymarks`**, **`trig_marks`** permettent de graduer ou numéroter les axes en fractions de π ou d'un coefficient donné.

Ainsi, voilà un fichier jps, et la figure générée :



source jps

```
traceaxes
4 setytkstep
2 setymkstep
ticks
ymarks
pi ymark
e xmark
.25 xsubtick
/ymarkstyle {brtext} def
[-3 -1 1 5] {ymark} apply

-3 -1.5 setorigine
bleu traceaxes noir
[(1/6) (-1/4) (2/3) (-3/4) (1)] trig_ymarks
setTimes
[(-1/4) (1/3) (-3/4) (1)] (a) 5 coeff_xmarks
```

4.3 - Les unités

Pour tracer des flèches indiquant les unités, on dispose de la commande **`unites`**.

4.4 - En résumé

– **`tracerepere`** – ⟶ trace les axes *Ox* et *Oy*, des flèches au bout des axes, ainsi que des flèches unités

– **`traceOx`** – ⟶ trace l'axe *Ox*

– **`traceOy`** – ⟶ trace l'axe *Oy*

– **`traceaxes`** – ⟶ trace les axes *Ox* et *Oy*

– **`axeOxarrow`** – ⟶ trace la flèche au bout de l'axe *Ox*

– **`axeOyarrow`** – ⟶ trace la flèche au bout de l'axe *Oy*

– **`axesarrow`** – ⟶ trace les flèches au bout des axes *Ox* et *Oy*

– **`unites`** – ⟶ trace les flèches unités sur chacun des axes

x y **`setorigine`** – ⟶ affecte les coordonnées (x, y) au point origine du repère *jps*

x **`xtick`** – ⟶ trace un tiret au point d'abscisse *x* de l'axe *Ox*

– **`xticks`** – ⟶ trace tous les tirets de l'axe *Ox*

y **`ytick`** – ⟶ trace un tiret au point d'ordonnée *y* de l'axe *Oy*

– **`yticks`** – ⟶ trace tous les tirets de l'axe *Oy*

– **`ticks`** – ⟶ trace tous les tirets des axes *Ox* et *Oy*

x **`xsubtick`** – ⟶ trace un sous-tiret au point d'abscisse *x* de l'axe *Ox*

– **`xsubticks`** – ⟶ trace tous les sous-tirets de l'axe *Ox*

y **`ysubtick`** – ⟶ trace un sous-tiret au point d'ordonnée *y* de l'axe *Oy*

– **`ysubticks`** – ⟶ trace tous les sous-tirets de l'axe *Oy*

– **`subticks`** – ⟶ trace tous les sous-tirets des axes *Ox* et *Oy*

x **`xmark`** – ⟶ inscrit la numérotation au point d'abscisse *x* de l'axe *Ox*

– **`xmarks`** – ⟶ inscrit toute la numérotation de l'axe *Ox*

y **`ymark`** – ⟶ inscrit la numérotation au point d'ordonnée *y* de l'axe *Oy*

– **`ymarks`** – ⟶ inscrit toute la numérotation de l'axe *Oy*

– **`marks`** – ⟶ inscrit toute la numérotation des axes *Ox* et *Oy*

string A **`xmarkstyle`** – ⟶ Procédure utilisée par les commandes **`xmark`** et dérivées pour inscrire la chaîne *string* au point *A*

string A **`ymarkstyle`** – ⟶ Procédure utilisée par les commandes **`ymark`** et dérivées pour inscrire la chaîne *string* au point *A*

array coeff **`coeff_xticks`** – ⟶ Trace des tirets sur l'axe *Ox* en fractions de *coeff* en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

array coeff **coeff_yticks** – ⟶ Trace des tirets sur l'axe *Oy* en fractions de *coeff* en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

array string coeff **coeff_xmarks** – ⟶ Numérote l'axe *Ox* en fractions de *coeff* en utilisant *string* pour l'affichage et les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

array string coeff **coeff_ymarks** – ⟶ Numérote l'axe *Ox* en fractions de *coeff* en utilisant *string* pour l'affichage et les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

array **trig_xmarks** – ⟶ Numérote l'axe *Ox* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

array **trig_ymarks** – ⟶ Numérote l'axe *Oy* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

array **trig_marks** – ⟶ Numérote les axes *Ox* et *Oy* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, [(1)(−3/2)(2/3)] est un tableau valide

t **setxmkstep** – ⟶ définit le pas pour la numérotation sur l'axe *Ox*

t **setymkstep** – ⟶ définit le pas pour la numérotation sur l'axe *Oy*

t_1 t_2 **setmkstep** – ⟶ définit respectivement les pas t_1 et t_2 pour la numérotation sur les axes *Ox* et *Oy*

t **setxtkstep** – ⟶ définit le pas pour les tirets sur l'axe *Ox*

t **setytkstep** – ⟶ définit le pas pour les tirets sur l'axe *Oy*

t_1 t_2 **settkstep** – ⟶ définit respectivement les pas t_1 et t_2 pour les tirets sur les axes *Ox* et *Oy*

t **setxsubtkstep** – ⟶ définit le pas pour les sous-tirets sur l'axe *Ox*

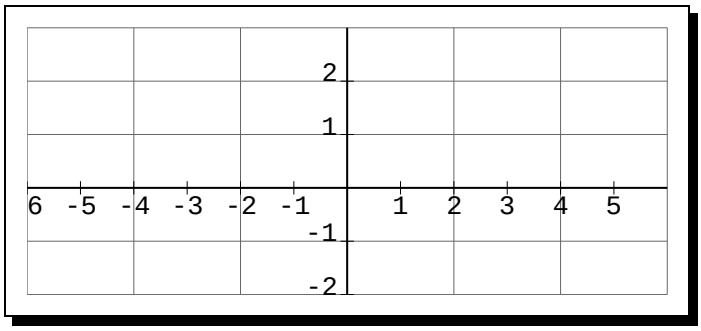
t **setysubtkstep** – ⟶ définit le pas pour les sous-tirets sur l'axe *Oy*

t_1 t_2 **setsubtkstep** – ⟶ définit respectivement les pas t_1 et t_2 pour les sous-tirets sur les axes *Ox* et *Oy*

5. Quadrillage

La commande **quadrillage** permet de générer un quadrillage en fond d'écran. Cette commande ne prend pas d'argument, mais utilise les 3 paramètres *xstquadrillage*, *ystquadrillage* et *quadrillagegray* définissant respectivement les incrémentations horizontales et verticales et le niveau de gris utilisé (de 0 à 1). On peut modifier ces paramètres avec les commandes à un argument **setxstquadrillage**, **setystquadrillage** et **setquadrillagegray**. On peut également modifier les 2 paramètres d'incrémentation en une seule fois avec la commande à 2 arguments **setquadrillagestep**.

Ainsi, voilà un fichier jps, et la figure générée :



source jps

```
40 setxunit
-2 3 setyrange
2 1 setquadrillagestep
quadrillage
traceaxes
marks
```

Pour des quadrillages (un peu) plus complexes, on dispose de la commande **Quadrillage** qui dispose de plusieurs syntaxes :

– **quadrillage** – ⟶ Trace un quadrillage simple. Les paramètres sont *xstquadrillage*, *ystquadrillage*, *quadrillagegray* et *quadrillagewd*

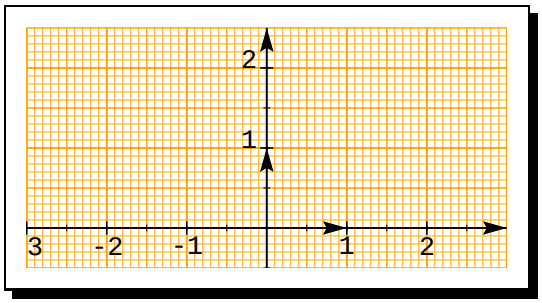
[xs_0 ys_0 xs_1 ys_1 xs_2 ys_2] { *color* } **Quadrillage** – ⟶ Trace un triple quadrillage, de couleur *color*, avec les pas sur *x* et *y* définis par les xs_i et ys_i. L'argument {*color*} est optionnel, de même que les couples (xs_2, ys_2) et (xs_1, ys_1). Les épaisseurs de trait sont relevées dans le tableau *quadrillagewidth*

xs_0 ys_0 { *color* } **Quadrillage** – ⟶ Trace un quadrillage simple, de couleur *color*, avec les pas sur *x* et *y* définis par le couple (xs_0, ys_0). L'argument {*color*} est optionnel. Avec cette syntaxe, l'épaisseur du trait est l'épaisseur courante.

Pour changer les épaisseurs des traits de quadrillage, il suffit de réaffecter le tableau *quadrillagewidth*. L'instruction

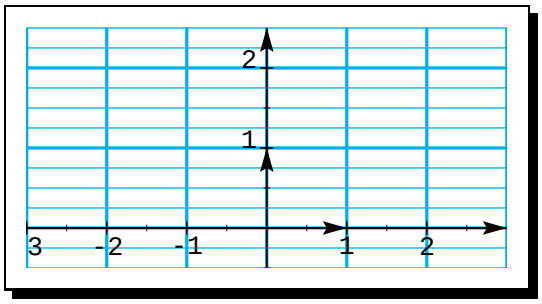
utilisée par défaut est **`/quadrillagewidth [.7 .4 .2] def`**.

Les réglages sont à faire au coup par coup, ils dépendent beaucoup du support final et des transformations intermédiaires (image papier noir et blanc, couleur, format jpeg, format pdf, etc. . .)



source jps

```
30 setxunit
-3 3 setxrange
-.5 2.5 setyrange

%% la commande utilisee avec le maximum
%% d'arguments :
%%    - dans le tableau les pas pour
%%       * les traits epais en x, y
%%       * les traits moyens en x, y
%%       * les traits fin en x, y
%%    - puis la couleur entre accolades

[1 1 .5 .5 .1 .1] {orange} Quadrillage

%% pour voir le repere maintenant,
%% mieux vaut epaissir le trait
.7 setlinewidth
tracerepere
marks
```



source jps

```
30 setxunit
-3 3 setxrange
-.5 2.5 setyrange
%% on change les epaisseurs par defaut
%% (on ne met que 2 dimensions car on
%% a juste un double quadrillage)
/quadrillagewidth [1.2 .5] def

%% la commande utilisee avec :
%%    - dans le tableau les pas pour
%%       * les traits epais en x, y
%%       * les traits moyens en x, y
%%    - puis la couleur entre accolades

[1 1 1 .25] {cyan} Quadrillage

%% pour voir le repere maintenant,
%% mieux vaut epaissir le trait
.7 setlinewidth
tracerepere
marks
```

6. Les objets graphiques élémentaires

6.1 - Les points

Un objet *point* est défini par la donnée de deux nombres, représentant ses coordonnées dans le repère jps. Par exemple **`1 1`** représentera le point de coordonnées $(1, 1)$.

La commande **`point`** permet de dessiner un point donné par ses coordonnées cartésiennes. Ainsi la commande **`-2 3 point`** va dessiner le point de coordonnées $(-2, 3)$. Pour dessiner plusieurs points, on utilise la commande **`points`** dont la syntaxe est **`[`**$A_1\ A_2\ \ldots\ A_n$**`] points`** où les A_i sont des couples de nombres réels définissant des points.

Pour placer un point avec la projection en pointillés sur les axes, on utilise la commande **`dashpoint`**. Là encore, on dispose de la commande **`dashpoints`** si on veut faire des tracés multiples.

À noter que **`O`** désigne le point prédéfini $O(0, 0)$.

On peut nommer un point en définissant une procédure Postscript. Ainsi, la séquence **`/A {4 2} def`** définit A comme étant le point de coordonnées $(4, 2)$. Le format *jps* définit également la commande **`defpoint`** qui remplit un rôle

analogue. Par exemple la séquence **`4 2 /A defpoint`** définit également A comme étant le point de coordonnées $(4, 2)$. Cette dernière commande permet de récupérer les coordonnées d'un point sur la pile, ce qui est particulièrement intéressant lorsque le point A est le résultat d'un calcul complexe, qui est réalisé à chaque appel avec la première méthode.

En résumé :

x y name **`defpoint`** $-\!\!\longrightarrow$ affecte le nom *name* au couple de nombres (x, y)

point **`point`** $-\!\!\longrightarrow$ dessine le point spécifié

point **`dashpoint`** $-\!\!\longrightarrow$ dessine le point spécifié avec projection sur les axes en pointillé

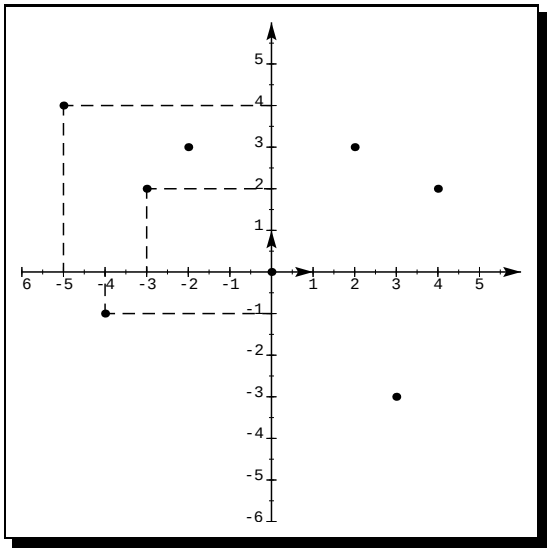
`[` $point_1 \ldots point_n$ **`]`** **`points`** $-\!\!\longrightarrow$ dessine les points spécifiés

`[` $point_1 \ldots point_n$ **`]`** **`dashpoints`** $-\!\!\longrightarrow$ dessine les points spécifiés avec projection sur les axes en pointillé

$-$ **`O`** $O_x\ O_y \longrightarrow$ dépose sur la pile les coordonnées de l'origine du repère *jps*

Il existe de nombreuses autres façons de dessiner des points ; celle-ci seront détaillées dans un paragraphe ultérieur.

Pour le moment, voilà un fichier jps, et la figure générée :



source jps

```
tracerepere
marks
3 -3 point
-3 2 dashpoint
/A {4 2} def
A point

/B {-4 -1} def
/C {-5 4} def
/D {-2 3} def
2 3 /E defpoint

[B C] dashpoints
[D E O] points
```

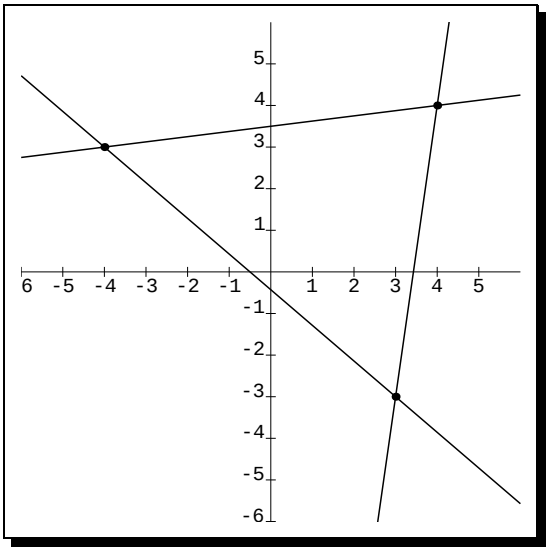
6.2 - Les droites

Un objet *droite* est défini par la donnée de deux de ses points. Par exemple **`1 1 2 2`** représente la droite passant les points de coordonnées $(1, 1)$ et $(2, 2)$.

La commande **`droite`** permet le dessin d'une droite donnée.

Comme pour les points, on peut nommer une droite en définissant une procédure Postscript. Par exemple **`/d1 {1 1 2 2} def`** définit d_1 comme étant la droite passant par les points de coordonnées respectives $(1, 1)$ et $(2, 2)$. Comme pour les points, on dispose de la commande **`defdroite`** qui permet de nommer une droite déposée au sommet de la pile.

Un exemple :



source jps

```
traceaxes
marks
/A {4 4} def
/B {3 -3} def
/C {-4 3} def
%% definition d'une droite
/d1 {A C} def
B C /d2 defdroite

[A B C] points
A B droite
d1 droite
d2 droite
```

En résumé :

d **`droite`** $-\!\!\longrightarrow$ trace la droite *d*

d name **`defdroite`** $-\!\!\longrightarrow$ associe la droite *d* au nom *name*

6.3 - Les cercles

Un objet *cercle* est défini par la donnée de son centre et de son rayon. Par exemple **`1 1 2`** représente le cercle de centre (1, 1) et de rayon 2.

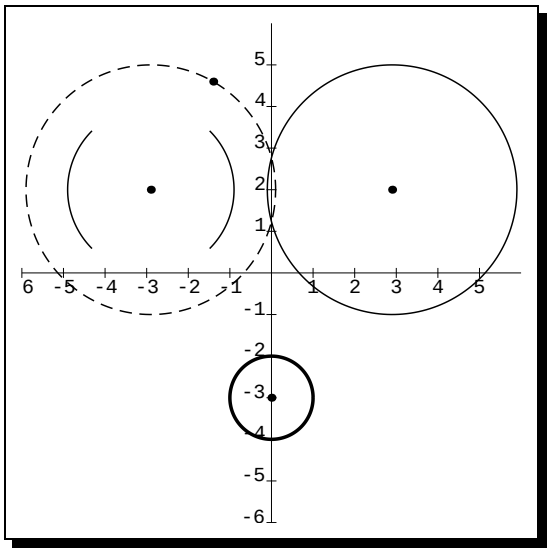
La commande **`cercle`** permet le dessin d'un cercle donné, et la commande **`Cercle`** permet de n'en tracer qu'un arc spécifié par 2 angles.

La commande **`cpoint`** permet de spécifier un point particulier d'un cercle donné.

Comme pour les points et les droites, on peut nommer un cercles en définissant une procédure Postscript. Par exemple si *A* est un point déjà défini, **`/C1 {A 2} def`** définit C_1 comme étant le cercle de centre *A* et de rayon 2, et les commandes **`C1 cercle`** et **`A 2 cercle`** ont toute deux pour effet de le tracer.

On peut également nommer un cercle déposé au sommet de la pile en utilisant la commande **`defcercle`**.

Par exemple :



source jps

```
traceaxes
marks

/A {2.9 2} def
/B {-2.9 2} def
B 3 /C1 defcercle

[A B 0 -3] points

A 3 cercle
-45 45 B 2 Cercle
135 225 B 2 Cercle
pointilles
C1 cercle
60 C1 cpoint point
1.5 setlinewidth
continu
0 -3 1 cercle
```

En résumé :

cerc **`cercle`** $-\!\!\longrightarrow$ trace le cercle spécifié

α β *cerc* **`Cercle`** $-\!\!\longrightarrow$ trace la portion de cercle spécifiée

α *cerc* **`cpoint`** *M* $\longrightarrow$ dépose sur la pile les coordonnées du point *M* du cercle *cerc* correspondant à l'angle α

cerc name **defcercle** $-\!\!\longrightarrow$ associe le cercle *cerc* au nom *name*

6.4 - Les lignes brisées et polygônes

Le format *jps* fournit l'objet *polygone* qui est un tableau de points. Par exemple, **[]** est le polygône vide, et **[0 0 1 1]** est le polygône ayant pour sommets les points de coordonnées $(0, 0)$ et $(1, 1)$.

On dispose de 2 commandes de tracé : **ligne** et **polygone** qui ont toutes deux une syntaxe du type

$$\textbf{[}\ A_1\ A_2\ \ldots\ A_n\ \textbf{]}\ \textbf{ligne} \qquad \text{et} \qquad \textbf{[}\ A_1\ A_2\ \ldots\ A_n\ \textbf{]}\ \textbf{polygone}$$

où les A_i sont des points. La différence entre ces deux commandes est que **polygone** ferme le contour avant de faire le tracé.

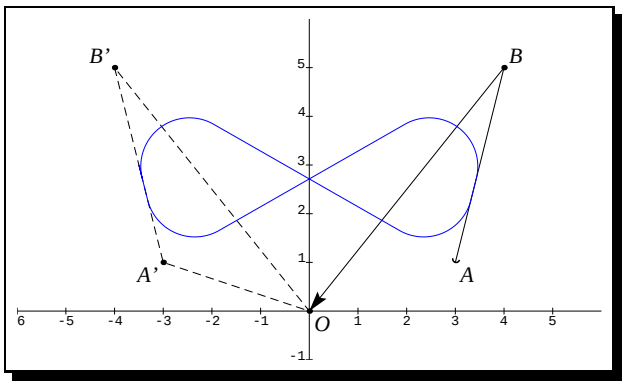
Pour ces deux commandes, on dipose du paramètre *linearc* (à 0 par défaut) qui indique au format le rayon (dans le repère jps) de l'arc de cercle reliant deux segments de droites. (Remarque : pour que cet arc de cercle paraisse circulaire, le repère jps doit être orthonormé.)

Pour un simple tracé entre deux points, le format propose la commande **line** disposant d'une syntaxe (un peu) plus légère :

$$A\ B\ \textbf{line}$$

Les commandes **ligne** et **line** admettent en plus une option sous la forme d'une chaîne de caractères pour la gestion des fins de ligne (voir le paragraphe sur les paramètres jps).

Un exemple :



source jps

```
-1 6 setyrange
traceaxes
marks

/A {3 1} def   /A' {-3 1} def
/B {4 5} def   /B' {-4 5} def

[B O A' B'] points
/lisere_arrow false def
/arrowscale {2 dup} def
[A B O] (\(->) ligne
pointilles
[A' B' O] polygone

continu bleu
/linearc 1 def
[A' B A B'] polygone

noir
16 setfontsize
setTimesItalic
(A) A drtext     (A') A' dltext
(B) B urtext     (O) O drtext
(B') B' ultext
```

En résumé, on a les commandes :

array **polygone** $-\!\!\longrightarrow$ trace le polygône défini par le tableau de points *array*

array string **ligne** $-\!\!\longrightarrow$ trace la ligne définie par le tableau de points *array*. Les extrémités de la ligne sont décrites par la chaîne optionnelle *string*

A B string **line** $-\!\!\longrightarrow$ trace le segment de droite $[AB]$. Les extrémités du segment sont décrites par la chaîne optionnelle *string*

et la variable :

- *linearc* : indique au format le rayon (dans le repère jps) de l'arc de cercle reliant deux segments de droites pour les commandes **ligne**, **polygone**, ainsi que les **frame** et dérivés. . **valeur par défaut :** 0

6.5 - Les ellipses

Pour décrire une ellipse, on dispose d'un objet *ellipse*, constitué d'un tableau du type **[***I a b* α**]** où *A* est le centre de l'ellipse, *a* une mesure de son demi-axe « horizontal », *b* une mesure de son demi-axe « vertical », et α une mesure de l'angle que fait le premier axe de l'ellipse avec l'axe *Ox*.

On dispose de 2 commandes pour le tracé d'un objet de type *ellipse*. La commande **`ellipse`** permet le dessin d'une ellipse donnée, et la commande **`Ellipse`** permet de n'en tracer qu'un arc spécifié par 2 angles.

Les commandes **`epoint`** et **`Epoint`** permettent de spécifier un point particulier d'une ellipse donnée.

Syntaxe :

ell **`ellipse`** $-\!\!\longrightarrow$ trace l'ellipse spécifiée

t ell **`epoint`** $A \longrightarrow A$ est le point de l'ellipse *ell* de paramètre t (avec le paramétrage $(a\cos t, b\sin t)$)

α *ell* **`Epoint`** $A \longrightarrow A$ est le point de l'ellipse *ell* tel que $(\vec{u}, \overrightarrow{\Omega A}) = \alpha$, où Ω désigne le centre de l'ellipse, et $\vec{u}$ le vecteur de base de l'axe *Ox*.

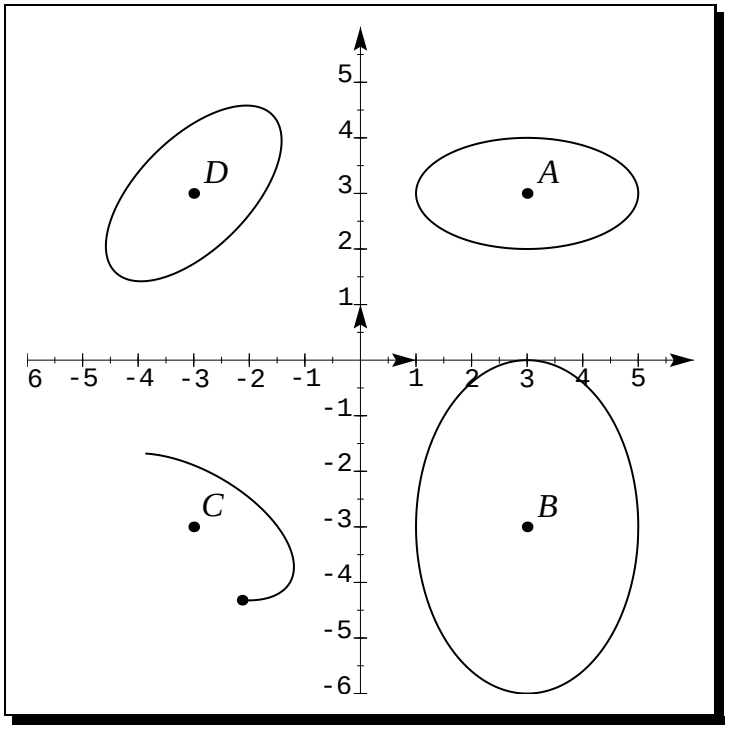
$\alpha\ \beta$ *ell* **`Ellipse`** $-\!\!\longrightarrow$ trace la portion de l'ellipse entre les points de paramètres respectifs α et β.

Dans certains cas, on a envie d'une syntaxe plus légère. Pour ce faire, les 2 commandes **`ellipse`** et **`Ellipse`** (ainsi que leurs versions étoilées) diposent d'une autre syntaxe.

I a b **`ellipse`** $-\!\!\longrightarrow$ trace l'ellipse de centre *I* et de demi-axes *a* et *b*. L'angle que fait le premier axe avec l'horizontale est déterminé par le paramètre *ellipseangle*

$\alpha\ \beta$ *I a b* **`Ellipse`** $-\!\!\longrightarrow$ trace la portion spécifiée de l'ellipse de centre *I* et de demi-axes *a* et *b*. L'angle que fait le premier axe avec l'horizontale est déterminé par le paramètre *ellipseangle*

Ces 2 commandes utilisent le paramètre *ellipseangle*, à 0 par défaut, déterminant l'angle de rotation de l'ellipse autour de son centre. Ce paramètre est modifiable directement ou avec la commande **`setellipseangle`**.



source jps

```
tracerepere
marks

/A {3 3} def   /C {-3 -3} def
/B {3 -3} def  /D {-3 3} def
/ell [C 2 1 -30] def

[A 2 1 0] ellipse
B 2 3 ellipse
-45 135 ell Ellipse
-45 ell epoint point

/ellipseangle 45 def
D 2 1 ellipse

[A B C D] points

setTimesItalic
   (A) A urtext   (C) C urtext
   (B) B urtext   (D) D urtext
```

6.6 - Les frames

Le format *jps* prévoit 4 types de représentations pour les rectangles : *frame*, *rframe*, *cframe* et *mframe*. Le type *frame* est constitué d'un tableau du type [*A B* α] où *A* et *B* sont respectivement les coins inférieurs droit et coins supérieurs gauches, alors que le nombre α désigne l'angle que fait le côté inférieur avec l'horizontale. Les 3 autres types sont constitués d'un tableau du type [*A L* ℓ α] où *L* et ℓ désignent respectivement les dimensions inférieures et latérales, *A* désigne le point de référence, et le nombre α désigne l'angle que fait le côté inférieur avec l'horizontale.

Ces quatres types de rectangles sont respectivement dessinés par les commandes **`frame`**, **`rframe`**, **`cframe`** et **`mframe`**, ces commandes supportant en plus une chaîne de caractère optionnelle, constitué des caractères **`b`**, **`u`**, **`l`** et **`r`**, indiquant si l'on veut tracer tout ou partie seulement des côtés du rectangle.

Comme pour les ellipses, on peut utiliser les commandes de tracé avec une syntaxe plus légère ne précisant pas les angles. Ce qui donne :

A B **`frame`** $-\!\!\longrightarrow$ trace le rectangle dont les points *A* et *B* sont repectivement les coins inférieur droit et supérieur gauche

A L ℓ **`rframe`** $-\!\!\longrightarrow$ trace le rectangle dont le point *A* est le coin inférieur droit, de dimension horizontale *L* et de dimension verticale ℓ

A L ℓ **cframe** $-\longrightarrow$ trace le rectangle dont le point *A* est le centre, de dimension horizontale *L* et de dimension verticale ℓ

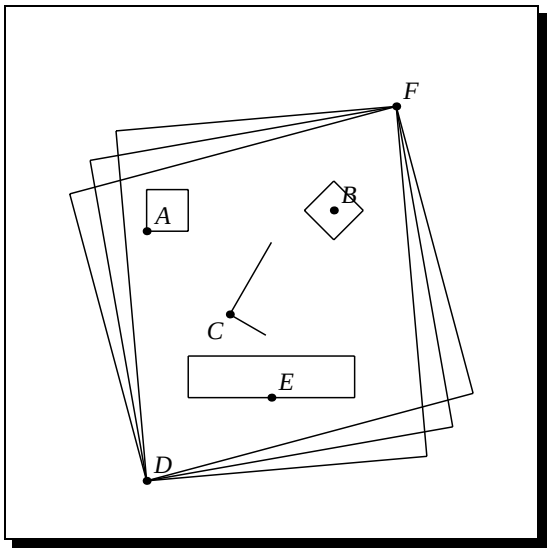
A L ℓ **mframe** $-\longrightarrow$ trace le rectangle dont le point *A* est le milieu du côté inférieur, de dimension horizontale *L* et de dimension verticale ℓ

Ces quatre dernières commandes utilisent le paramètre *frameangle*, à 0 par défaut, déterminant l'angle de rotation du rectangle autour de son point de référence. Ce paramètre est modifiable directement ou avec la commande **setframeangle**.

Comme pour les lignes et polygones, on dispose du paramètre *linearc*, égal à 0 par défaut.

Si on le désire, et si le paramètre *linearc* est nul, on peut ne dessiner que certains côtés du rectangle. Il suffit pour cela de passer en argument une chaîne contenant les caractères **b**, **u**, **l** ou **r** (pour *up*, *bottom*, *left* et *right*).

Un exemple :



source jps

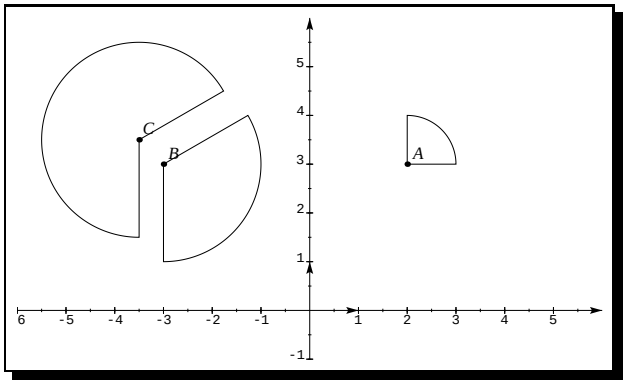
```
/A {-3 1} def     /D {-3 -5} def
/B {1.5 1.5} def  /E {0 -3} def
/C {-1 -1} def    /F {3 4} def

A 1 1 rframe
E 4 1 mframe
/frameangle 45 def
B 1 1 cframe
%[D F 5] frame
%[D F 10] frame
%[D F 15] frame
[C 1 2 -30] (lb) rframe

[A B C D E F] points

setTimesItalic
   (A) A urtext   (D) D urtext
   (B) B urtext   (E) E urtext
   (C) C dltext   (F) F urtext
```

6.7 - Les wedge

La commande **wedge** permet de tracer une portion « camembert ».

Syntaxe :

$\alpha\ \beta\ A\ r$ **wedge** $-\longrightarrow$ trace la portion de camembert de centre *A*, de rayon *r*, délimité par les angles α et β



source jps

```
-1 6 setyrange
tracerepere
marks

/A {2 3} def
/B {-3 3} def
/C {-3.5 3.5} def

0 90 A 1 wedge
-90 30 B 2 wedge
270 30 C 2 wedge

[A B C] points

setTimesItalic
   (A) A urtext
   (B) B urtext
   (C) C urtext
```

6.8 - Courbes de Bézier

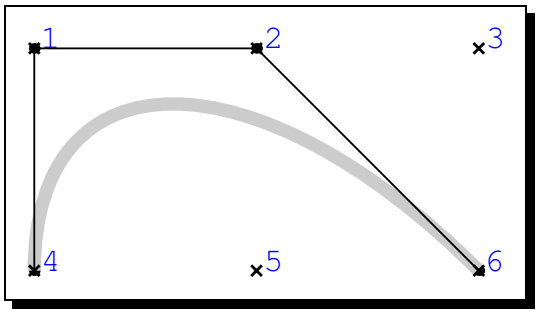
La commande **bezier_curve** permet de tracer une courbe de Bézier décrite par un tableau de points. Par exemple,

la commande **[A B C D] bezier_curve** va tracer la courbe de Bézier passant par les points A et D et de points de contrôle B et C (les droites (AB) et (CD) seront alors tangentes à la courbe produite).

Tout tableau de n points avec $n-1$ multiple de 3 peut être considéré comme décrivant une courbe de Bézier.

Le format contient une bascule indiquant si les points de contrôle doivent être dessinés (faux par défaut). La commande **withcontrolpoints** sélectionne le tracé des points de contrôle, et **withoutcontrolpoints** le désélectionne.

On dispose également des commandes **bezier_curve*** et **bezier_curve_** qui réagissent de la façon habituelle.



source jps

```
autocrop
-1 3 setxrange
-1 2 setyrange
/A1 {0 1} def       /A4 {0 0} def
/A2 {1 1} def     /A5 {1 0} def
/A3 {2 1} def     /A6 {2 0} def
withcontrolpoints
.8 setgray
5 setlinewidth

%% exemple 2 p 19 du Metafont Book
[A4 A1 A2 A6] bezier_curve
noir
[A1 A2 A3 A4 A5 A6 ] {times2} plot
.7 setlinewidth
[A4 A1 A2 A6] ligne
bleu setCourier
(1) A1 brtext      (4) A4 brtext
(2) A2 brtext      (5) A5 brtext
(3) A3 brtext      (6) A6 brtext
```

7. Les courbes mathématiques

7.1 - Opérateurs et fonctions mathématiques

Les fonctions et constantes mathématiques reconnues sont :

$a\ b$ **add** $c \longrightarrow c = a + c$

$a\ b$ **sub** $c \longrightarrow c = a - c$

$a\ b$ **mul** $c \longrightarrow c = a \times b$

$a\ b$ **div** $c \longrightarrow c = a/b$

$a\ b$ **idiv** $q \longrightarrow q$ est le quotient de la division euclidienne de a par b

$a\ b$ **mod** $r \longrightarrow r$ est reste de la division euclidienne de a par b

a **sin** $c \longrightarrow c = \sin a$ (a en degré)

a **cos** $c \longrightarrow c = \cos a$ (a en degré)

a **tan** $c \longrightarrow c = \tan a$ (a en degré)

a **cotan** $c \longrightarrow c = \text{cotan}\, a$ (a en degré)

a **arccos** $c \longrightarrow c = \text{Arccos}\, a$ (en degrés)

a **arcsin** $c \longrightarrow c = \text{Arcsin}\, a$ (en degrés)

a **arctan** $c \longrightarrow c = \text{Arctan}\, a$ (en degrés)

a **Sin** $c \longrightarrow c = \sin a$ (a en radian)

a **Cos** $c \longrightarrow c = \cos a$ (a en radian)

a **Tan** $c \longrightarrow c = \tan a$ (a en radian)

a **coTan** $c \longrightarrow c = \text{cotan}\, a$ (a en radian)

a **Arccos** $c \longrightarrow c = \text{Arccos}\, a$ (en radians)

a **Arcsin** $c \longrightarrow c = \text{Arcsin}\, a$ (en radians)

a **Arctan** $c \longrightarrow c = \text{Arctan}\, a$ (en radians)

a **sinh** $c \longrightarrow c = \text{sh}\, a$

a **cosh** $c \longrightarrow c = \text{ch}\, a$

a **tanh** $c \longrightarrow c = \text{th}\, a$

a **cotanh** $c \longrightarrow c = \coth a$

a **argsinh** $c \longrightarrow c = \text{Arg sh}\, a$

a **argcosh** $c \longrightarrow c = \text{Arg ch}\, a$

a **argtanh** $c \longrightarrow c = \text{Arg th}\, a$

a **Exp** $c \longrightarrow c = \exp(a) = e^a$

a **ln** $c \longrightarrow c = \ln a$, logarithme népérien de a

a **log** $c \longrightarrow c = \log a$, logarithme décimal de a

a **sqrt** $c \longrightarrow c = \sqrt{a}$

$a\ n$ **exp** $c \longrightarrow c = a^n$

a **abs** $c \longrightarrow c = |a|$

a **neg** $c \longrightarrow c = -a$

$a\ b$ **max** $c \longrightarrow c$ est le plus grand des deux nombres a et b

$a\ b$ **min** $c \longrightarrow c$ est le plus petit des deux nombres a et b

– **pi** $3,141\,59 \longrightarrow$ le nombre π

– **e** $2,718 \longrightarrow$ le nombre e

num_1 **ceiling** $num_2 \longrightarrow$ plafond de num_1

num_1 **floor** $num_2 \longrightarrow$ plancher de num_1

num_1 **round** $num_2 \longrightarrow$ arrondit num_1 à l'entier le plus proche

num_1 **truncate** $num_2 \longrightarrow$ enlève la partie fractionnaire de num_1

– **rand** $int \longrightarrow$ génère un entier au hasard

n **factorielle** $b \longrightarrow b = a!$

$n\ p$ **Anp** $a \longrightarrow a = A_n^p = n \times (n-1) \times \cdots \times (n-p+1)$

$n\ p$ **Cnp** $c \longrightarrow c = C_n^p = A_n^p / p!$

$k\ n\ p$ **binomiale** $a \longrightarrow a = C_n^k p^k (1-p)^{n-k}$

$x\ \lambda$ **Poisson** $y \longrightarrow y$ est l'image de x par la loi de Poisson de paramètre λ

$x\ m\ \sigma$ **Gauss** $y \longrightarrow y$ est l'image de x par la loi normale de paramètres m et σ. Soit $y = e^{-\left((x-m)/\sigma\right)^2/2}/(\sigma\sqrt{2\pi})$

x **fresnelS** $y \longrightarrow y$ est l'image de x par la fonction de Fresnel $x \mapsto \displaystyle\int_0^x \sin\left(\frac{\pi t^2}{2}\right) dt$

x **fresnelC** $y \longrightarrow y$ est l'image de x par la fonction de Fresnel $x \mapsto \displaystyle\int_0^x \cos\left(\frac{\pi t^2}{2}\right) dt$

x **Gammaln** $y \longrightarrow y$ est l'image de x par la fonction $\ln \circ \Gamma$

x **Bessel_y0** $y \longrightarrow y$ est l'image de x par la fonction Y_0 de Bessel

x **Bessel_y1** $y \longrightarrow y$ est l'image de x par la fonction Y_1 de Bessel

x **Bessel_j0** $y \longrightarrow y$ est l'image de x par la fonction J_0 de Bessel

x **Bessel_j1** $y \longrightarrow y$ est l'image de x par la fonction J_1 de Bessel

7.2 - Définir une fonction

Pour définir une fonction, le plus simple (?) consiste à utiliser le langage Postscript et la manipulation de la pile. On a tout de même rajouté la commande **setxvar** qui permet d'assigner à la variable **x** l'élement en haut de la pile.

Par exemple, voici quelques définitions de fonctions, avec leur équivalent mathématique :

/f {} def	$\longrightarrow f(x) = x$
/f {setxvar x} def	$\longrightarrow f(x) = x$
/f {Sin} def	$\longrightarrow f(x) = \sin x$
/f {setxvar x Sin} def	$\longrightarrow f(x) = \sin x$
/f {2 add ln} def	$\longrightarrow f(x) = \ln(x+2)$
/f {setxvar x 2 add ln} def	$\longrightarrow f(x) = \ln(x+2)$
/g {setxvar x 2 add x neg 3 sub mul} def	$\longrightarrow g(x) = (x+2)(-x-3)$
/g {setxvar x 2 exp 3 x mul add 2 sub} def	$\longrightarrow g(x) = x^2 + 3x - 2$

Néammoins, pour les réticents, la commande **#rpn#** (placée dès le premier caractère de la ligne) permet de sous-traiter l'écriture en notation polonaise inverse de l'expression de la fonction en envoyant toute la fin de la ligne au script **exp2rpn** de Jean-Michel Sarlat. Ainsi, voici une façon de définir une fonction à partir de son écriture en notation

cartésienne :

```
/g {setxvar
#rpn# x^2 + 3*x - 2 * ln (x)
} def
```
$\longrightarrow g(x) = x^2 + 3x - 2\ln x$

Ce script fonctionne également sur les fonctions à plusieurs variables. On peut donc utiliser les fonctions **gauss**, **Cnp**, **Poisson**, etc. . . sur une ligne *#rpn#*.

Remarque : On peut également utiliser une ligne *#rpn#* pour une définition de constante : tous les caractères alphabétiques, ainsi que *pi* sont considérés comme des constantes. La définition suivante est donc valable :

```
/maconstante {
#rpn# 3 * sqrt (pi) + Cnp (n, 3*i)
} def
```
$\longrightarrow maconstante = 3\sqrt{\pi} + C_n^{3i}$

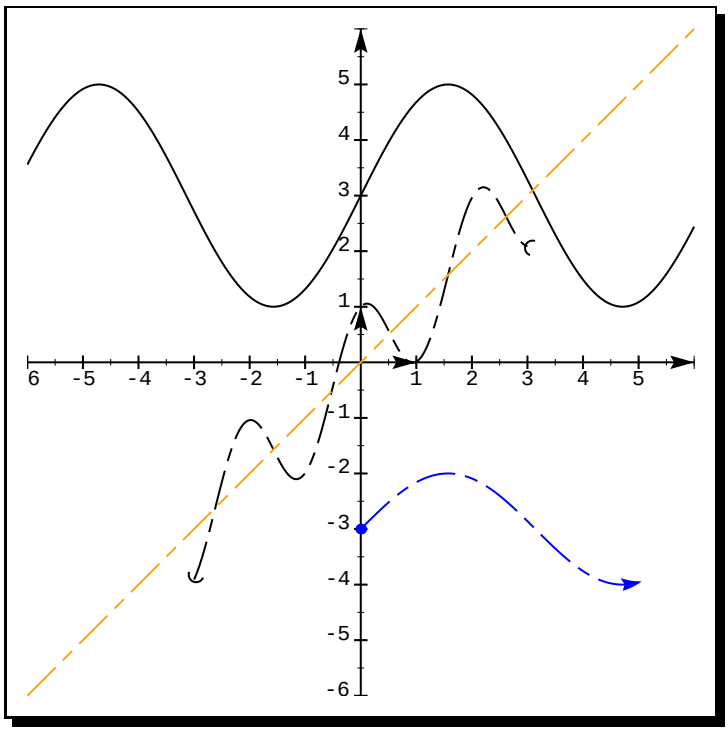
7.3 - Courbes en coordonnées cartésiennes

Pour le tracé d'une fonction numérique, on utilise la commande **courbe** pour un tracé sur toute la largeur de la fenêtre de dessin, et la commande **Courbe** pour un tracé partiel. Les problèmes d'existence ne sont pas gérés par le format et sont laissés à l'utilisateur. Celui-ci ne devra donc pas etre étonné si le tracé de la fonction $x \mapsto \ln x$ ou $x \mapsto 1/x$ sur $[-5; 5]$ provoque des erreurs postscript. . .

La commande **courbe** prend en argument un exécutable, qui correspond à la fonction dont on veut le tracé. Elle effectue le tracé de la courbe représentative de cette fonction sur l'intervalle $[xmin, xmax]$. La commande **Courbe** prend en plus deux nombres a et b en argument, et trace la courbe de la fonction donnée sur l'intervalle $[a, b]$.

Par exemple, le fichier suivant génère le tracé de 2 graphes de fonctions nommées, et de 2 graphes de fonctions non nommées :

- le graphe, sur tout l'intervalle considéré, de la fonction f définie par $f(x) = 3 + 2\sin x$,
- le tracé, sur l'intervalle $[1, 5]$, de la courbe de la fonction g définie par $g(x) = x + \cos(3x)$,
- le tracé, sur l'intervalle $[xmin, xmax]$, de la courbe de la fonction $t \mapsto t$,
- et enfin le tracé, sur l'intervalle $[0; 5]$, du graphe de la fonction $t \mapsto \sin t - 3$.



source jps

```
tracerepere
marks

%% la fonction f definie par
%% f (x) = 3 + 2 Sin (x)
/f {setxvar
   x Sin 2 mul 3 add
} def

%% la fonction g definie par
%% g (x) = x + Cos (3x)
/g {setxvar
#rpn#   Cos (3*x) + x
} def

{f} courbe
mixte
-3 3 {g} (\(-\() Courbe

%% on peut donner directement
%% le corps de la fonction :
%% l'identite (rien a faire, on
%% laisse le nombre sur la pile)
orange
{} courbe
%% la fonction x --> Sin (x) - 3
bleu
0 5 {Sin 3 sub} (*->) Courbe
```

Pour tracer une courbe, on calcule un certain nombre de points également répartis sur l'intervalle de dessin, puis on réunit ces points deux à deux par des segments de droite. Le nombre de ces points est stocké dans la variable globale *resolution*, fixée à 200 par défaut. La commande **setresolution** permet de changer cette valeur.

En résumé :

$\{f\}$ **courbe** $-\longrightarrow$ trace la courbe représentative de la fonction f sur l'intervalle $[xmin; xmax]$

proc **courbe** – ⟶ trace la courbe représentative sur l'intervalle [*xmin*; *xmax*] de la fonction définie par l'exécutable *proc*

a b { *f* } **Courbe** – ⟶ trace la courbe représentative de la fonction f sur l'intervalle [*a*; *b*]

a b proc **Courbe** – ⟶ trace la courbe représentative sur l'intervalle [*A*; *B*] de la fonction définie par l'exécutable *proc*

n **setresolution** – ⟶ affecte l'entier *n* à la variable *resolution* qui contrôle le nombre de points de calculs pour la représentation d'une courbe de fonction numérique

a **setxvar** – ⟶ affecte la valeur réelle *a* à la variable *x*

7.4 - Courbes paramétrées

Il est possible de dessiner des courbes décrites par un paramétrage du type

$$t \mapsto (X(t), Y(t)).$$

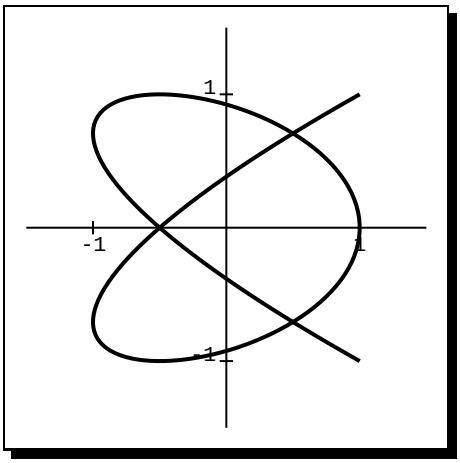
Pour cela, il faut définir deux fonctions, puis utiliser la commande **courbeparam** à laquelle on passe les fonctions sous forme de 2 exécutables.

On règle l'intervalle de dessin avec la commande **settrange**. Pour la définition des fonctions X et Y, on dispose des commandes **setxvar** et **settvar** qui assignent l'élément en haut de la pile respectivement à la variable *x* ou à la variable *t*.

Comme pour les fonctions numériques, la variable globale *resolution* est utilisée pour indiquer le nombre de points à calculer pour le dessin de la courbe.

Dans l'exemple ci-dessous, on génère le tracé de la courbe de la fonction f définie de $[-\pi; \pi]$ vers $\mathbb{R}^2$ par :

$$f(t) = (\cos(4t), \sin(3t))$$



source jps

```
-1.5 1.5 setxrange
-1.5 1.5 setyrange
50 setxunit
traceaxes
marks

1.5 setlinewidth

pi -2 div
pi 2 div settrange

/X {4 mul Cos} def
/Y {3 mul Sin} def

{X} {Y} courbeparam
```

En résumé :

{*X*} {*Y*} **courbeparam** – ⟶ trace la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle [*tmin*; *tmax*]

$proc_1$ $proc_2$ **courbeparam** – ⟶ trace sur l'intervalle [*tmin*; *tmax*] la courbe paramétrée définie par les exécutables $proc_1$ et $proc_2$

a b {*X*} {*Y*} **Courbeparam** – ⟶ trace la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle [*a*; *b*]

a b $proc_1$ $proc_2$ **Courbeparam** – ⟶ trace sur l'intervalle [*a*; *b*] la courbe paramétrée définie par les exécutables $proc_1$ et $proc_2$

a **settvar** – ⟶ affecte la valeur réelle *a* à la variable *t*

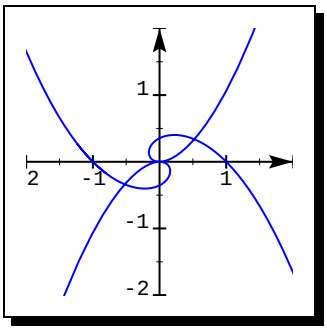
a b **settrange** – ⟶ affecte respectivement les valeurs réelles *a* et *b* aux variables *tmin* et *tmax*

7.5 - Courbes en coordonnées polaires

De manière analogue, on dispose des commandes **courbepolar** et **Courbepolar** qui permettent de tracer une courbe en coordonnées polaires :

{ρ} **courbepolar** – ⟶ trace la courbe paramétrée $\theta \mapsto (\rho(\theta)\cos\theta; \rho(\theta)\sin\theta)$ sur l'intervalle [*tmin*; *tmax*]

a b {ρ} **Courbepolar** – ⟶ trace la courbe paramétrée $\theta \mapsto (\rho(\theta)\cos\theta; \rho(\theta)\sin\theta)$ sur l'intervalle $[a; b]$



source jps

```
-2 2 setxrange
-2 2 setyrange
25 setxunit
/unites {} def
tracerepere
marks

-6.5 6.5 settrange
/R {settvar
#rpn# (Cos (t/2))/(1+Sin(t))
} def
bleu
{R} courbepolar
```

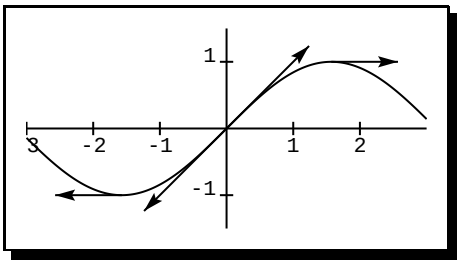
7.6 - Tangentes à une courbe de fonction numérique

La commande **tangente-** (resp. **tangente+**) permet le tracé de la demi-tangente à gauche (resp. à droite) pour la courbe d'une fonction numérique. La commande **tangente** trace les 2 demi-tangentes.

Toutes ces commandes reçoivent la fonction sous forme d'un exécutable, et déterminent les nombres dérivés en valeur approcheé. Néammoins, la commande **tangente** permet un calcul exact : si on lui transmet une chaine de caractère désignant le nom de la fonction, elle utilisera la définition de la fonction dérivée (dont le nom est obligatoirement celui de la fontion suffixée avec le caractère **'**) pour calculer le nombre dérivé.

Le seul paramètre utilisé est *taillletangente* qui donne la longueur, dans le repère, de la tangente tracée. On peut modifier ce paramètre directement ou avec la commande **settailletangente**.

Par exemple, le fichier suivant génère le dessin de la courbe représentative de la fonction f avec quelques tangentes. La fonction f étant définie par $f(x) = 1 + x^2/2$, sa fonction dérivée est $f'(x) = x$.



source jps

```
-3 3 setxrange
-1.5 1.5 setyrange
25 setxunit
traceaxes
marks

/f {Sin} def
%% trace de la courbe
{f} courbe
%% puis des tangentes en valeur approchee
2 settailletangente
pi 2 div {f} tangente+
pi -2 div {f} tangente-
%% la derniere tangente en valeur exacte
3.5 settailletangente
/f' {Cos} def
0 (f) tangente
```

En résumé :

x {*f*} **tangente-** – ⟶ trace la demi-tangente à gauche pour la courbe de la fonction f au point d'abscisse *x*.

x {*f*} **tangente+** – ⟶ trace la demi-tangente à droite pour la courbe de la fonction f au point d'abscisse *x*.

x {*f*} **tangente** – ⟶ trace les 2 demi-tangentes à gauche et à droite pour la courbe de la fonction f au point d'abscisse *x*.

x string **tangente** – ⟶ trace la tangente à la courbe de la fonction f au point d'abscisse *x*, où f est l'exécutable désigné par la chaîne de caractères *string*. Attention, le calcul utilise l'exécutable f' dont le nom est obtenu en adjoignant à la chaîne *string* le caractère **'**. L'exécutable f' doit donc être défini.

x **settailletangente** $-\longrightarrow$ affecte la valeur réelle a à la variable *tailletangente* qui gère la taille des tangentes tracées par **tangente**. La taille est exprimée en unités de l'axe *Ox*

- *tailletangente* : la taille, exprimée en unités de l'axe *Ox*, des tangentes tracées par **tangente**.. **valeur par défaut :** 1

7.7 - Tangentes à une courbe de fonction paramétrée

De façon analogue aux courbes de fonctions numériques, la commande **ptangente-** (resp. **ptangente+**) permet le tracé de la demi-tangente à gauche (resp. à droite) pour la courbe d'une fonction paramétrée. La commande **ptangente** trace les 2 demi-tangentes.

Comme précédemment, ces commandes déterminent les vecteurs dérivés en valeur approcheé, mais la commande **ptangente** permet un calcul exact : si on lui transmet deux chaines de caractères désignant les noms des fonctions coordonnées, elle utilisera les définitions des fonctions dérivées pour calculer le vecteur dérivé.

Le paramètre *tailletangente* est utilisé conformément à ce que l'on pense.

En résumé :

t $\{X\}$ $\{Y\}$ **ptangente-** $-\longrightarrow$ trace la demi-tangente à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre t.

t $\{X\}$ $\{Y\}$ **ptangente+** $-\longrightarrow$ trace la demi-tangente à droite pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre t.

t $\{X\}$ $\{Y\}$ **ptangente** $-\longrightarrow$ trace les demi-tangentes à droite et à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre t.

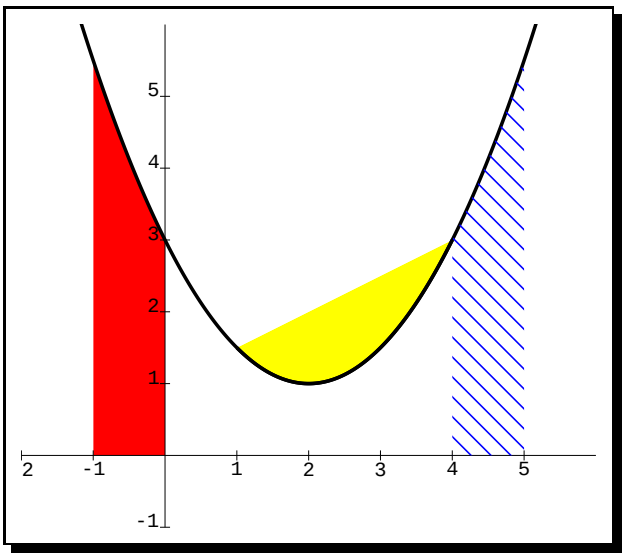
t (X) (Y) **ptangente** $-\longrightarrow$ trace les demi-tangentes à droite et à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre t en utilisant les fonctions X' et Y' pour le calcul du vecteur dérivé.

7.8 - Domaine limité par une courbe de fonction numérique

Ici, pas besoin de calculer une primitive de la fonction f. La commande **Hachcourbe** permet de hachurer le domaine plan limité par l'axe *Ox*, les droites verticales dont les abscisses sont passées en argument, et la courbe d'une fonction numérique passée en argument (sous forme d'un exécutable). La commande **hachcourbe** fait la même chose, mais avec les droites verticales correspondant généralement aux bornes de la fenêtre de dessin (variables *xmin* et *xmax*).

On dispose également des commandes **fillcourbe** et **Fillcourbe** qui procèdent de manière analogue mais qui remplissent le domaine plan selon les indications de *fillstyle*.

Restent les commandes **courbe*** et **Courbe*** qui remplissent, selon les indications de *fillstyle*, le domaine plan délimité par la courbe et ses points extrémaux reliés par un segment de droite.

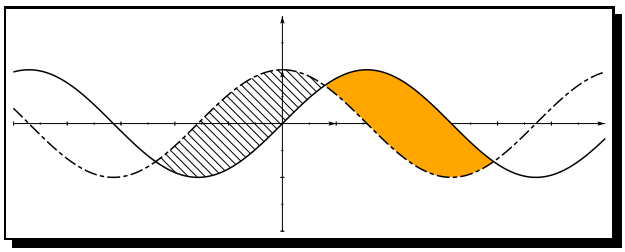


source jps

```
-1 6 setyrange
-2 6 setxrange
1.5 setlinewidth

/f {setxvar
#rpn# .5 * (x-2)^2 + 1
} def

/fillstyle {jaune fill} def
1 4 {f} Courbe*
/fillstyle {rouge fill} def
-1 0 {f} Fillcourbe

/hcolor {bleu} def
traceaxes
marks
{f} courbe
4 5 {f} Hachcourbe
```

On peut également hachurer un domaine plan limité par les courbes représentatives de 2 fonctions nommées. On a encore 2 commandes **hachcourbes** et **Hachcourbes**, la première avec 2 arguments (les noms des fonctions sous forme de 2 chaînes de caractères), et la deuxième avec quatre. Ces deux commandes ont également leurs analogues **fillcourbes** et **Fillcourbes**.

Par exemple, le fichier suivant génère le dessin de la courbe représentative des 2 fonctions f et g ainsi qu'un domaine hachuré et un domaine coloré. Les fonctions f et g sont respectivement définies par $f(x) = \sin x$ et $g(x) = \cos x$.



source jps

```
-5 6 setxrange
-2 2 setyrange
tracerepere

/a pi 4 div def

/f {Sin} def
/g {Cos} def

/fillstyle {orange fill} def

%% on remplit les domaines
-3 a mul a {f} {g} Hachcourbes
a a pi add {f} {g} Fillcourbes

%% puis on trace les courbes
1.5 setlinewidth
{f} courbe
mixte
{g} courbe
```

Pour les hachures, on dispose de 2 paramètres *hangle*, *hstep*, *hwidth* et *hcolor* qui définissent respectivement l'angle (en degré) que font les hachures avec l'horizontale ($-45°$ par défaut), l'écart entre 2 hachures (4 par défaut), l'épaisseur et la couleur du trait. On peut modifier ces paramètres directement ou avec les commandes **`sethangle`** et **`sethstep`**.

En résumé :

$\{f\}$ **`hachcourbe`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de f, et les droites verticales $x = xmin$ et $x = xmax$

proc **`hachcourbe`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = xmin$ et $x = xmax$

a b $\{f\}$ **`Hachcourbe`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de f, et les droites verticales $x = a$ et $x = b$

a b proc **`Hachcourbe`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = a$ et $x = b$

$\{f\}$ $\{g\}$ **`hachcourbes`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par les courbes représentatives des fonctions f et g, et les droites verticales $x = xmin$ et $x = xmax$

$proc_1$ $proc_2$ **`hachcourbes`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = xmin$ et $x = xmax$

a b $\{f\}$ $\{g\}$ **`Hachcourbes`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par les courbes représentatives des fonctions f et g, et les droites verticales $x = a$ et $x = b$

a b $proc_1$ $proc_2$ **`Hachcourbes`** $-\!\!\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = a$ et $x = b$

$\{f\}$ **`fillcourbe`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de f, et les droites verticales $x = xmin$ et $x = xmax$

proc **`fillcourbe`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = xmin$ et $x = xmax$

a b $\{f\}$ **`Fillcourbe`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de f, et les droites verticales $x = a$ et $x = b$

a b proc **`Fillcourbe`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = a$ et $x = b$

$\{f\}$ $\{g\}$ **`fillcourbes`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par les courbes représentatives des fonctions f et g, et les droites verticales $x = xmin$ et $x = xmax$

$proc_1$ $proc_2$ **`fillcourbes`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = xmin$ et $x = xmax$

a b $\{f\}$ $\{g\}$ **`Fillcourbes`** $-\!\!\longrightarrow$ remplit, suivant les indications de *fillstyle*, le domaine plan délimité

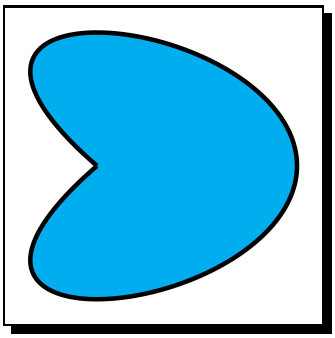
par les courbes représentatives des fonctions f et g, et les droites verticales $x = a$ et $x = b$

a b $proc_1$ $proc_2$ **`Fillcourbes`** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe Ox, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = a$ et $x = b$

7.9 - Domaine limité par une courbe paramétrée

On dispose ici des commandes **`courbeparam*`** et **`Courbeparam*`** qui fonctionnent comme leurs homologues pour les fonctions numériques. Attention, on peut parfois obtenir des résultats surprenants si choisit mal les valeurs du paramètre pour les points extrêmaux.

Un exemple où les valeurs extrêmales du paramètre sont $-\pi/3$ et $\pi/3$ pour la limitation du domaine plan.



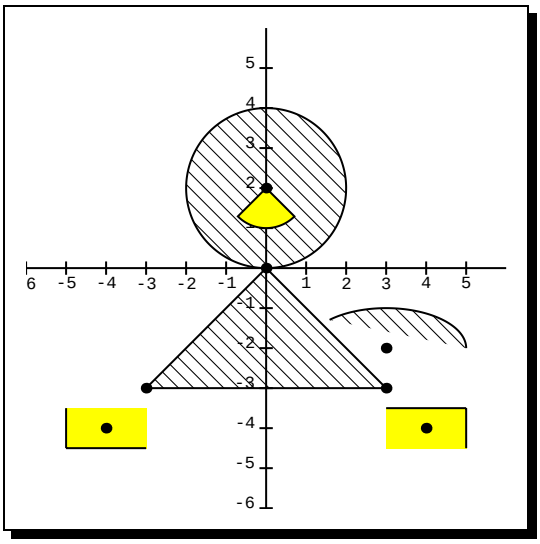
source jps

```
50 setxunit
autocrop

/X {4 mul Cos} def
/Y {3 mul Sin} def

/fillstyle {cyan fill} def
1.7 setlinewidth

pi -3 div
dup neg
{X} {Y} Courbeparam*
```

8. Les objets étoilés

La plupart des objets disposent d'une version *étoilée* qui permet de remplir un domaine plan défini par l'objet. Par exemple, la commande **`A 2 cercle*`** dessine le cercle de centre A et de rayon 2, puis remplit ce cercle suivant les indications de *fillstyle* (vide par défaut, voir le paragraphe sur les différents types de remplissage). De façon analogue, on dispose ainsi des commandes **`polygone*`**, **`frame*`**, **`rframe*`**, **`mframe*`**, **`cframe*`**, **`wedge*`**, **`courbe*`**, **`Courbe*`**, **`courbeparam*`**, **`Courbeparam*`**, **`cercle*`**, **`Cercle*`**, **`ABCercle*`**, **`ellipse*`**, **`Ellipse*`**.

Par exemple :



source jps

```
15 setxunit
8 setfontsize
/hstep 4 def
/hwidth .4 def
traceaxes
marks

/fillstyle {hachure} def
/A {0 2} def     /D {3 -2} def
/B {-3 -3} def  /E {-4 -4} def
/C {3 -3} def  /F {4 -4} def

A 2 cercle*
[B C O] polygone*
0 135 D 2 1 Ellipse*

/fillstyle {jaune fill} def
-45 -135 A 1 wedge*
E 2 1 (bl) cframe*
F 2 1 (ur) cframe*

[O A B C D E F] points
```

9. Gestion du texte

9.1 - Sélection d'une police

Pour le moment, le format *jps* reconnaît cinq familles de fontes différentes : Courier, Times, Symbol, Helvetica et Palatino. Chacune d'entre elle est disponible en standard, gras, italique, ou gras et italique.

Pour sélectionner une police, on dispose des commandes

setTimes	**setTimesItalic**	**setTimesBold**	**setTimesBoldItalic**
setCourier	**setCourierItalic**	**setCourierBold**	**setCourierBoldItalic**
setPalatino	**setPalatinoItalic**	**setPalatinoBold**	**setPalatinoBoldItalic**
setSymbol	**setSymbolItalic**	**setSymbolBold**	**setSymbolBoldItalic**
setHelvetica	**setHelveticaItalic**	**setHelveticaBold**	**setHelveticaBoldItalic**

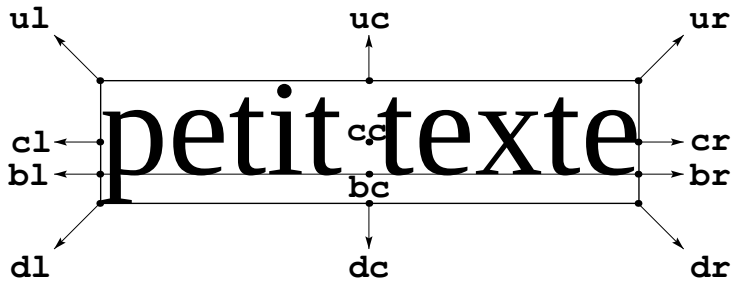
Pour utiliser d'autres fontes, il faut employer directement les ordres Postscript.

9.2 - Sélection de la taille d'une police

La taille de la police courante est stockée dans la variable *fontsize*. Elle est fixée par défaut à $12,5$, et on peut modifier cette valeur avec la commande **setfontsize**. Par exemple, l'instruction **8 setfontsize** sélectionne 8 pour la taille de la police courante.

9.3 - Affichage d'une chaîne de caractères

Pour une chaîne de caractères donnée, le format considère la boite contenant exactement cette chaîne, et reconnaît 12 points de référence distincts, représentés ci-dessous avec les directions des déplacements associés :



(Les lettres utilisées sont *u*, *c*, *b*, *d*, *l*, *r*, comme abbréviations respectives de *up*, *center*, *baseline*, *down*, *left*, *right*.)

Pour afficher une chaîne de caractères, il faut, après avoir sélectionné une police, spécifier la chaîne, le point de placement dans le repère jps, et la direction d'affichage par rapport au point de placement. Ainsi, la commande

```
(un petit texte) 1 1 cctext
```

va placer le point *cc* de la chaîne de caractères 'un petit texte' au point de coordonnées $(1, 1)$.

Lorsque la direction spécifiée est autre que *bc* ou *cc*, le format ajoute un déplacement avant l'affichage. Les paramètres utilisés pour ce déplacement sont **hadjust** et **vadjust** qui contiennent les valeurs en points postscript des déplacement horizontaux et verticaux respectivement. Par défaut, ces valeurs sont fixées à $3,75$ points postscript. Ainsi, la commande

```
(avec déplacement) 1 1 urtext
```

va placer le point *dl* de la chaîne de caractères 'avec déplacement' en haut en droite du point de coordonnées $(1, 1)$. Plus précisément, le point *dl* de la boite correspondant à cette chaîne va être placé exactement au point $(1, 1)$ (coordonnées dans le repère *jps*), avant de subir une translation de vecteur (*had just*, *vad just*) (dans le repère postscript).

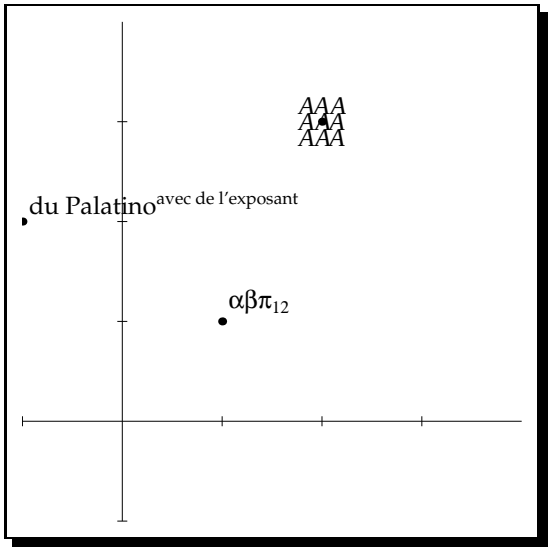
Les commandes disponibles sont **dltext**, **dctext**, **drtext**, **ultext**, **uctext**, **urtext**, **cltext**, **cctext**, **crtext**, **bltext**, **bctext**, **brtext**.

Les commandes **dtext**, **ltext**, **rtext**, **text** et **utext** sont obsolètes depuis la version $0.04c$.

9.4 - Indice et exposant

On dispose pour le moment de 2 commandes : **indice** et **exposant**. Elles prennent en argument une chaîne de caractères et procèdent aux étapes suivantes : sélection de la police courante avec une réduction de 70%, déplacement

vertical adapté du point courant, insertion de la chaîne de caractères, rétablissement de la police courante à sa taille initiale, puis repositionnement du point courant sur la ligne de base.



source jps

```
-1 4 setxrange
-1 4 setyrange
traceaxes
ticks
/A {2 3} def
/B {xmin 2} def
/C {1 1} def

[A B C] points

setTimesItalic
(A) A dctext    (A) A uctext   (A) A cctext
(A) A drtext    (A) A urtext   (A) A crtext
(A) A dltext    (A) A ultext   (A) A cltext

setPalatino
   (du Palatino) B urtext
   (avec de l'exposant) exposant

setSymbol
   (abp) C urtext
   (12) indice
```

9.5 - Encadrement

On peut encadrer ou encercler les chaînes de caractères affichées avec les commandes **`urtext`**, **`uctext`**, etc. . . . La commandes **`boxit`** (resp. **`circleit`**, **`ovalit`**, **`diaboxit`**) dessinera un rectangle (resp. un cercle, une ellipse, un losange) autour du prochain label affiché.

L'espace laissé entre le cadre et la boite contenant le texte est géré par les variables **`dx_boxit`** et **`dy_boxit`** (dimensions en points postscript) initialisées à 0 par défaut.

La commande **`boxit_all`** (resp. **`circleit_all`**, **`ovalit_all`**, **`diaboxit_all`**) provoquera l'encadrement de tous les labels suivants, et elle sera annulée par la commande **`boxit_none`** (resp. **`circleit_none`**, **`ovalit_none`**, **`diaboxit_none`**).

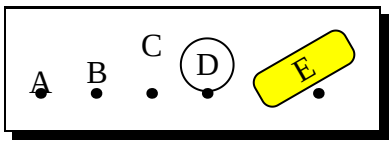
La commande *fillstyle* est utilisée pour tous ces encadrements.

9.6 - Options d'affichages

Pour les chaînes de caractères, on dispose de certaines options d'affichages : il est possible de préciser un déplacement ponctuel (en points postscript), un facteur d'agrandissement, et un angle de rotation. La syntaxe générale est

`(la chaine) A` (*dx dy*) [$scale_x$ $scale_y$] {α} **`urtext`**

où (*dx*, *dy*) est le déplacement en points postscript à rajouter au déplacement initial, ($scale_x$, $scale_y$) est le facteur d'agrandissement, et α l'angle de la rotation autour du point *A*. Il est également possible de donner la chaîne vide **`()`** pour le déplacement. Dans ce cas, le format annule tout déplacement; par exemple, la commande **`(texte) A () urtext`** va placer le point *dl* de boite encadrant la chaîne 'texte' *exactement* au point *A*.



source jps

```
autocrop
12 setfontsize

/A {-3 3} def
/B {-2 3} def
/C {-1 3} def
/D {0 3} def
/E {2 3} def

[A B C D E] points

noir
setTimes
   (A) A () uctext
   (B) B uctext
   (C) C (0 10) uctext
circleit
   (D) D uctext
boxit
/linearc .2 def
/fillstyle {jaune fill} def
/dx_boxit 16 def
   (E) E {30} uctext
```

9.7 - Labels TEX et LaTEX

La gestion du texte entièrement en Postscript est rapidement délicate, en particulier si l'on désire la gestion de toute la typographie mathématique.

Pour cela il est prévu un mécanisme permettant la sous-traitance, *via* les logiciels TEX et *dvips*. Cela a bien quelques inconvénients (le fichier postscript produit est plus gros et moins lisible), mais les textes produits par TEX sont tellement beaux. . .

Pour créer un label on dispose de deux méthodes, et chacune d'entre elle est déclinée en une version TEX et une version LaTEX.

• **1ère méthode :** on encapsule le code TEX (resp. LaTEX) dans 2 balises **<tex>** et **</tex>** (resp. **<latex>** et **</latex>**), chacune seule sur une ligne.

• **2ème méthode :** on commence la ligne par les caractères **#tex#** (resp. **#latex#**). Le reste de la ligne est alors considéré comme du code TEX (resp. LaTEX).

Une fois le code récupéré, celui-ci est compilé, soit par **tex** (format plain augmenté d'une couche pour avoir les lettres accentués et certaines fontes particulières(*) soit par **latex**(**).

Le résultat peut être alors réutilisé dans le format jps par les 16 commandes

urtexlabel	**uctexlabel**	**ubtexlabel**	**ultexlabel**
crtexlabel	**cctexlabel**	**cbtexlabel**	**cltexlabel**
brtexlabel	**bctexlabel**	**bbtexlabel**	**bltexlabel**
drtexlabel	**dctexlabel**	**dbtexlabel**	**dltexlabel**

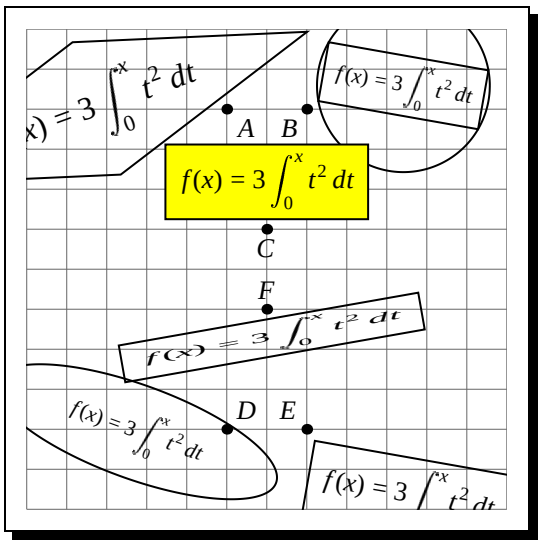
dont le fonctionnement est analogue à celui des commandes pour le placement du texte.

Prenons par exemple la syntaxe de **urtexlabel** :

A [*xscale yscale*] {*alpha*} **urtexlabel** $-\longrightarrow$ Se place en haut à droite du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle *alpha*. Le tableau d'échelle et l'argument {*alpha*} sont optionnels

Comme pour le texte, on dispose des options **boxit** et **circleit** pour les labels TEX.

(*) voir le fichier *jps.tex* dans le répertoire *jps2ps/*
(**) pour les packages utilisés, voir le fichier *labels.pl* dans le répertoire *jps2ps/*



source jps

```
15 setxunit
quadrillage
/A {-1 4} def   /D {-1 -4} def
/B {1 4} def    /E {1 -4} def
/C {0 1} def    /F {0 -1} def
[A B C D E F] points

%% definition du label
#tex# $$f (x) = 3 \int _0^x t^2 \, dt$$
ovalit
D [.8 .8] {-20} dltexlabel
diaboxit
A [1.2 dup] {20} ultexlabel
boxit_all
circleit
B [.8 dup] {-10} urtexlabel
E {-10} drtexlabel
F [1.5 .5] {10} dctexlabel
/fillstyle {jaune fill} def
C uctexlabel
boxit_none

setTimesItalic
(A) A drtext      (D) D urtext
(B) B dltext      (E) E ultext
(C) C dctext      (F) F uctext
```

10. Définir ses procédures

Pour définir une prodédure, il suffit d'associer un littéral à un exécutable. Par exemple, la définition d'un point par le code **`/A {1 2} def`** définit une procédure nommée *A*, procédure dont l'action est de déposer les nombres 1 et 2 sur la pile d'opérandes.

Une procédure peut être tout simplement constituée d'une liste de commandes. Par exemple, la définition de la commande **`tracerepere`** est

```
/tracerepere {
   traceaxes
   unites
   axesarrow
   ticks
   subticks
} def
```

Dans une procédure, on peut utiliser des variables, qui resteront locales pour peu que l'on prenne la peine d'utiliser un dictionnaire. Par exemple, la procédure **`addc`** (addition de deux nombres complexes) est définie comme suit :

```
%% syntaxe : a b a' b' addc --> z + z'
/addc {
4 dict begin
   /b' exch def
   /a' exch def
   /b exch def
   /a exch def
   a a' add
   b b' add
end
} def
```

On commence par ouvrir un dictionnaire pour les 4 variables *a*, *b*, a' et b', on récupère les données sur la pile pour les affecter à ces variables, puis on effectue les calculs et on dépose les résultats sur la pile. Pour finir, on jette le

dictionnaire (par **end**), ce qui a pour effet de nettoyer la mémoire de ces quatres variables. Si une de ces variables était définie avant l'ouverture du dictionnaire, elle retrouve son existence et sa valeur à la fermeture dusdit dictionnaire.

Pour des procédures plus élaborées, l'interpréteur postscript fournit toutes les structures de controle classiques, dont voici un bref résumé, extrait du manuel de référence postscript :

any **exec** $-\longrightarrow$ exécute un objet arbitraire

bool proc **if** $-\longrightarrow$ exécute *proc* si *bool* est $true$

bool $proc_1$ $proc_2$ **ifelse** $-\longrightarrow$ exécute $proc_1$ si *bool* est $true$, $proc_2$ si *bool* est $false$

int proc **repeat** $-\longrightarrow$ exécute *proc int* fois

proc **loop** $-\longrightarrow$ exécute *proc* un nombre indéfini de fois

Voir les annexes pour un listing complet des opérateurs postscript disponibles.

11. Paramètres graphiques

11.1 - Les paramètres postscript

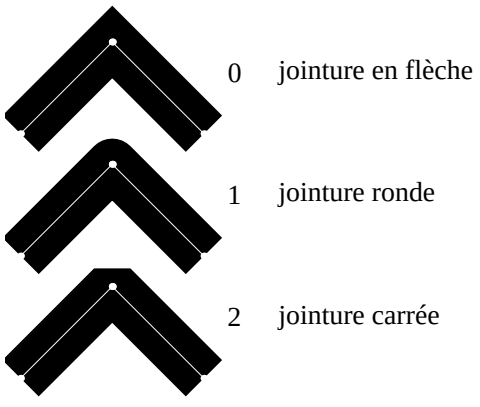
Le langage postscript possède de nombreux paramètres graphiques. On appelle *état graphique* à un instant donné l'ensemble de ces paramètres (*) avec leurs valeurs associées.

Citons quelques uns de ces paramètres :

- la largeur du trait est gérée par le paramètre *linewidth*. La commande **setlinewidth** permet de l'affecter, et la commande **currentlinewidth** renvoie la valeur courante ;
- le paramètre *linecap* sert à la gestion de la terminaison de ligne. Il peut prendre une des trois valeurs 0, 1 ou 2. On l'affecte avec la commande **setlinecap** et la commande **currentlinecap** renvoie la valeur courante

0 : Terminaison carrée. Le tracé est coupé à angle droit au point terminal du chemin. Il n'y a pas de projection au-delà de la fin du chemin.
1 : Terminaison ronde. Un arc semi-circulaire avec un diamètre égal à la largeur de la ligne est dessiné autour du point final et est rempli.
2 : Terminaison carrée projetée. Le tracé continue au-delà du point final du chemin sur une distance égale à la moitié de la largeur de la ligne et est carrée.

- le paramètre *linejoin* sert à la gestion de la jonction de ligne. Il peut prendre une des trois valeurs 0, 1 ou 2. On l'affecte avec la commande **setlinejoin** et la commande **currentlinejoin** renvoie la valeur courante

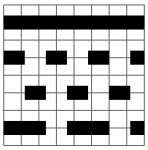


(*) plus beaucoup d'autres choses, comme le point courant, le chemin courant, le chemin d'incrustation, l'espace de couleur, etc. . . Voir le manuel de référence Posctscript pour une explication détaillée

- le paramètre *dash* sert à ajuster le motif du pointillé. On l'affecte avec la commande **`setdash`** dont la syntaxe est

  array offset **`setdash`**

  Si *array* est vide, on obtient une ligne continue. Sinon, *array* ne doit contenir que des nombres positifs, et non tous nuls, qui sont alors interprétés comme des distances le long du chemin (le tableau est lu cycliquement). Le nombre *offset* correspond à la « distance » le long du chemin à laquelle le motif doit commencer. La commande **`currentdash`** dépose sur la pile le tableau *array* et le nombre *offset*.



```
[] 0 setdash
[30] 0 setdash
[30] 30 setdash
[60 30] 0 setdash
```

L'interpréteur gère une pile des états graphiques. La commande **`gsave`** place une copie de l'état graphique courant sur cette pile, ce qui a pour effet de sauvegarder tous les éléments de l'état graphique. La commande **`grestore`** permet ensuite de retrouver l'état graphique précédemment sauvegardé.

11.2 - Les paramètres jps

11.2.1 - Les hachures

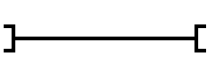
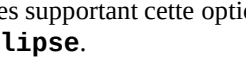
La commande **`hachure`** permet de hachurer l'ensemble de la fenêtre courante. Quatres paramètres sont utilisés :

- *hstep* : l'espace en points postscript séparant 2 hachures. **valeur par défaut :** 7
- *hangle* : l'angle en degrés que font les hachures avec l'horizontale. **valeur par défaut :** -45
- *hwidth* : l'épaisseur du trait en points postscript pour une hachure. **valeur par défaut :** $0,8$
- *hcolor* : couleur d'un hachure. **valeur par défaut :** {}

Voir un exemple d'utilisation dans le paragraphe « *Types de remplissage* ».

11.2.2 - Les fins de ligne

Un certains nombres de commandes de tracés du format jps admettent une option pour les terminaisons de ligne. Cette option est donnée sous la forme d'une chaîne de caractères. Le tableau ci-dessous récapitule les options disponibles.

valeur	exemple	effet
`-`		rien
`*-*`		disques centrés aux points terminaux
`o-o`		cercles centrés aux points terminaux
`<->`		flèches
`>-<`		flèches inversées
`\(-\)`		demi−cercles rentrant
`\)-\(`		demi−cercles sortant
`[-]`		crochets fermés
`]-[`		crochets ouverts

Ces options peuvent êtres mixées ; par exemple, **`<-*`** est une option valide.

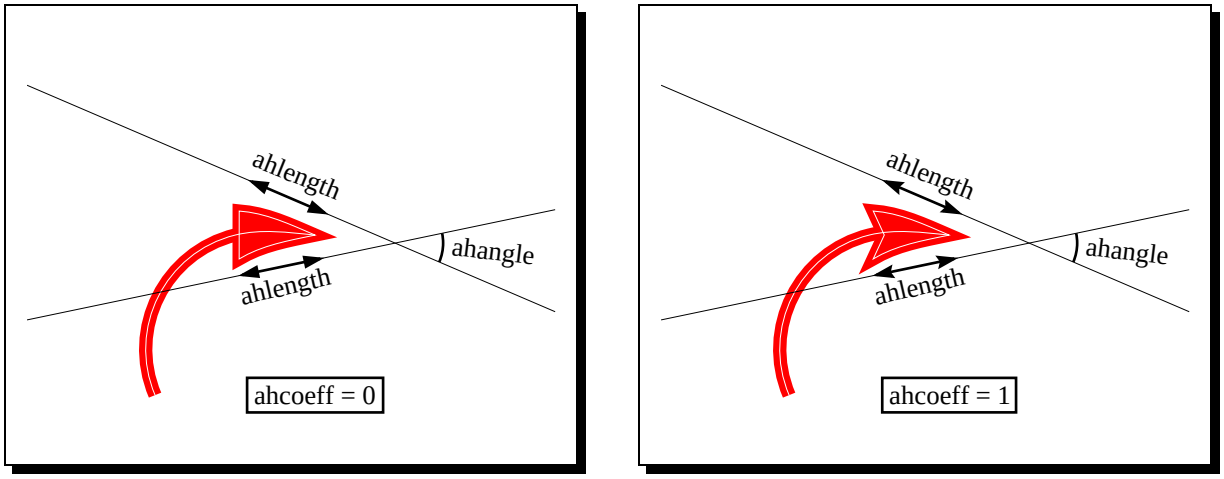
Pour le moment, les commandes supportant cette option sont **`courbeparam`**, **`Courbeparam`**, **`courbe`**, **`Courbe`**, **`ligne`**, **`line`**, **`Cercle`** et **`Ellipse`**.

11.2.3 - Les flèches

Le dessin des flèches s'obtient avec les options de tracés de fin de ligne. Trois paramètres permettent de modifier ces flèches :

- *ahlength* : longueur en picas des flèches de fin de ligne. **valeur par défaut :** 6
- *ahangle* : angle au sommet des flèches de fin de ligne. **valeur par défaut :** 30
- *ahcoeff* : coefficient entre 0 et 1 indiquant le niveau de décrochement sur la base de la flèche. **valeur par défaut :** 1

Pour obtenir les dessins attendus, il est nécessaire que la longueur (en picas) du chemin soit supérieur à *ahlength*.



En interne, le format utilise les commandes suivantes :

– **`arrowpath0`** *pathobj* $\longrightarrow$ chemin correspondant à l'axe d'une flèche potentielle au début du chemin courant lors du dernier appel à **`arrowpaths`**

– **`arrowpath1`** *pathobj* $\longrightarrow$ chemin correspondant à l'axe d'une flèche potentielle à la fin du chemin courant lors du dernier appel à **`arrowpaths`**

pathobj **`draw_arrow`** – $\longrightarrow$ considère *pathobj* comme l'axe d'une flèche et dessine la flèche correspondante

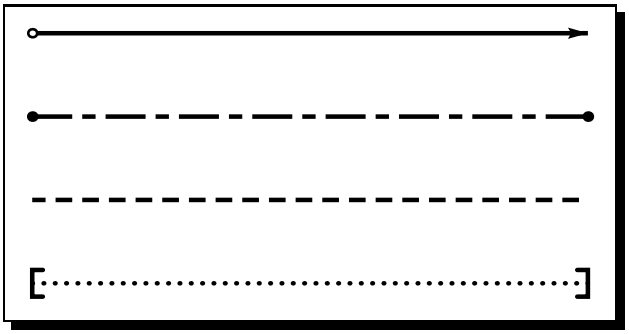
– **`arrowpaths`** $pathobj_1$ $pathobj_2$ $\longrightarrow$ depose sur la pile les 2 sous chemins en provenance du chemin courant necessaires à **`gere_arrowhead`**

11.2.4 - Type de tracés

On dispose pour le moment de 4 types de tracés prédéfinis : continu, pointillé, pointillé-tirets, ou mixte. Ainsi **`continu`** sélectionne le tracé continu, **`dotted`** sélectionne le tracé pointillé, **`pointilles`** sélectionne le tracé pointillé-tirets, et **`mixte`** sélectionne le tracé mixte.

Comme, pour des raisons que je ne m'explique pas, le rendu est différent suivant les outils, les versions et les options utilisées pour traiter les fichiers obtenus (par les Imagemagick, par gv, etc. . .) on dispose de 2 modes différents pour le rendu des pointillés : le *mode ps* et le *mode jpeg*. On positionne le mode avec les commandes **`psmode`** et **`jpegmode`**, le mode par défaut étant le mode ps.

Les anciennes commandes **`jpegpointilles`**, **`pspointilles`**, **`jpegmixte`** et **`psmixte`** ont disparu depuis la version 0.04*b* de février 2002. Le paramètre *curvelinewidth* a disparu depuis la version 0.05*g* d'août 2003.



source jps

```
autocrop
-6 6 setxrange

1.7 setlinewidth
continu
[-5 4 5 4] (o->)  ligne
mixte
[-5 2.5 5 2.5] (*-*) ligne
pointilles
[-5 1 5 1] ligne
dotted
[-5 -.5 5 -.5] ([-]) ligne
```

11.2.5 - Types de remplissages

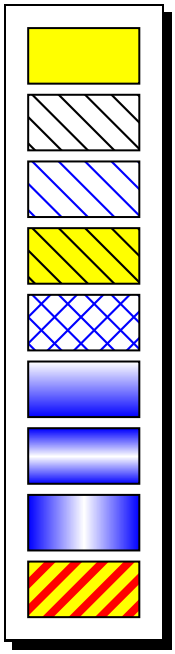
Lors du dessin d'un objet étoilé, le format utilise la procédure *fillstyle* (initialisée au vide par défaut) pour remplir le domaine plan délimité par l'objet.

Ainsi la définition **`/fillstyle {fill} def`** va provoquer le remplissage du fond avec la couleur courante (noir par défaut), alors que la définition **`/fillstyle {orange fill} def`** va changer la couleur courante avant le remplissage, cependant que la définition **`/fillstyle {hachure} def`** va demander de hachurer ce fond.

La commande **`gradientfill`** autorise un gradient de couleur. Sa syntaxe est :

gradientfill $couleur_1$ $couleur_2$ m $-\!\!\longrightarrow$ remplit le domaine d'incrustation en cours par un gradient de couleur passant de $couleur_1$ à $couleur_2$. Le nombre m appartient à $[0;1]$; il indique à quel moment doit atteindre la couleur $couleur_2$

- *gradangle* : Angle utilisé pour les gradients de couleurs. **valeur par défaut :** 0



source jps

```
autocrop
/dessin {
   0 5 2 1 rframe*
   0 -1.2 stranslate
} def

/fillstyle {jaune fill} def
dessin
/fillstyle {hachure} def
dessin
/fillstyle {bleu hachure} def
dessin
/fillstyle {jaune fill noir hachure} def
dessin
/fillstyle {
   blanc fill bleu hachure
   /hangle hangle 90 add store hachure
} def
dessin
/fillstyle {{bleu} {blanc} 0 gradientfill} def
dessin
/fillstyle {{bleu} {blanc} .5 gradientfill} def
dessin
/gradangle 90 def
/fillstyle {{bleu} {blanc} .5 gradientfill} def
dessin
%% en jouant sur les parametres des hachures
/hwidth 3 def /hcolor {rouge} def
/fillstyle {jaune fill hachure} def
dessin
```

11.2.6 - Types de points

Plusieurs commandes permettent de tracer un point. Les 6 prémières tracent de « vrais » points, les 3 dernières, destinées aux dessins statistiques, dessinent des rectangles entre le point et l'axe *Ox*. Toutes ont la même syntaxe :

A **`point`** $-\!\!\longrightarrow$ dessine un point en A dans le repère jps

A **`circ`** $-\!\!\longrightarrow$ dessine un point cerclé en A dans le repère jps

A **`times`** $-\!\!\longrightarrow$ dessine une croix au point A dans le repère jps

A **`square`** $-\!\!\longrightarrow$ dessine un carré au point A dans le repère jps

A **`plus`** $-\!\!\longrightarrow$ dessine une croix + au point A dans le repère jps

A **`diamond`** $-\!\!\longrightarrow$ dessine un losange au point A dans le repère jps

A **`rect`** $-\!\!\longrightarrow$ dessine un rectangle dont une base est porté par l'axe *Ox*, et tel que le point A soit le milieu du côté opposé

A **`rect`** $-\!\!\longrightarrow$ colorie en niveau de gris un rectangle dont une base est porté par l'axe *Ox*, et tel que le point A soit le milieu du côté opposé

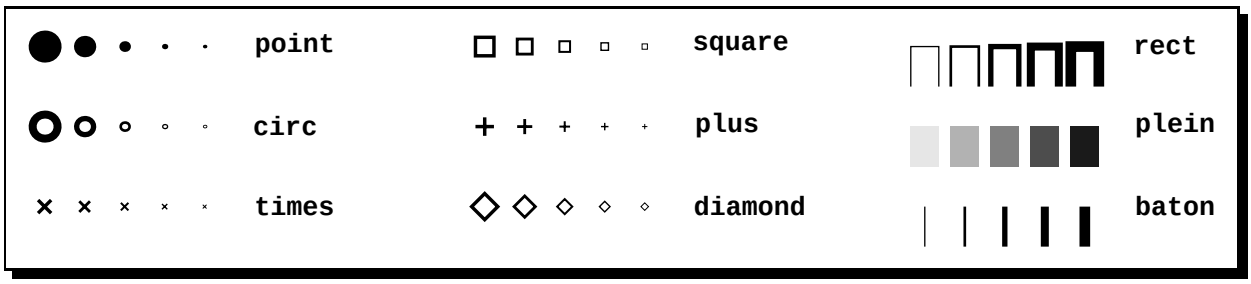
A **`baton`** $-\!\!\longrightarrow$ dessine un baton parallèle à l'axe *Oy*, dont une extrémité est le point A, et dont l'autre est sur l'axe *Ox*

Chacune de ces commandes se décline en 5 versions : on a par exemple **`point3`**, **`point2`**, **`point`**, **`point1`**,

point0 et de la même façon **times3**, **times2**, **times**, **times1**, **times0**, etc. . . Ces déclinaisons offrant différents paramétrages tout prêts de taille, de niveaux de gris etc. . .

Chacune des 6 premières commandes utilisent plusieurs paramètres :

- *dotsize* : dimension en points postcript paramétrant la taille des dessins de type point. **valeur par défaut :** 4
- *dotscale* : exécutable indiquant les échelles horizontale et verticale à appliquer pour les dessins de type point. **valeur par défaut :** { 1 1 }
- *dotangle* : paramètre indiquant l'angle en degrés de la rotation à appliquer pour les dessins de type point. **valeur par défaut :** 0



12. Les couleurs

Les commandes **bleu**, **rouge**, **vert**, **gris**, **jaune**, **noir**, **blanc**, **orange**, **rose**, **cyan**, **magenta**, définies dans le format *jps*, changent la couleur courante.

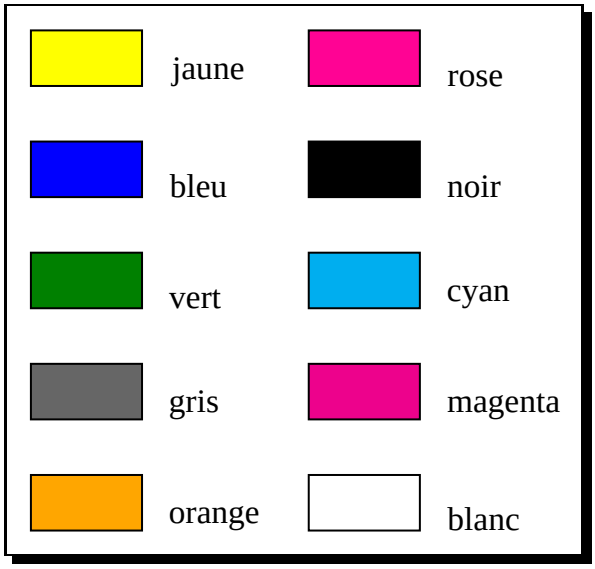
On peut également utiliser les ordres Postscript **setrgbcolor** ou **setcmykcolor**.

Syntaxe :

red green blue **setrgbcolor** – ⟶ définit la couleur RGB. Les nombres *red*, *green* et *blue* sont des réels compris entre 0 et 1

cyan magenta yellow black **setcmykcolor** – ⟶ définit la couleur CMYK. Les nombres *red*, *green* et *blue* sont des réels compris entre 0 et 1

Par exemple :



Pour des couleurs supplémentaires, voir le package 'color' qui propose plusieurs centaines d'autres couleurs prédéfinies.

III - Commandes de tracés supplémentaires

1. Autres tracés en géométrie

On dispose des commandes suivantes :

I A B **`arcp`** – ⟶ trace entre les points *A* et *B* l'arc du cercle de centre *I* et de rayon *IA* (sens trigonométrique)

I A B **`arcnp`** – ⟶ trace entre les points *A* et *B* l'arc du cercle de centre *I* et de rayon *IA* (sens inverse du sens trigonométrique)

I A α **`Arc`** – ⟶ trace au point *A* l'arc de cercle d'angle 2α centré en *A*

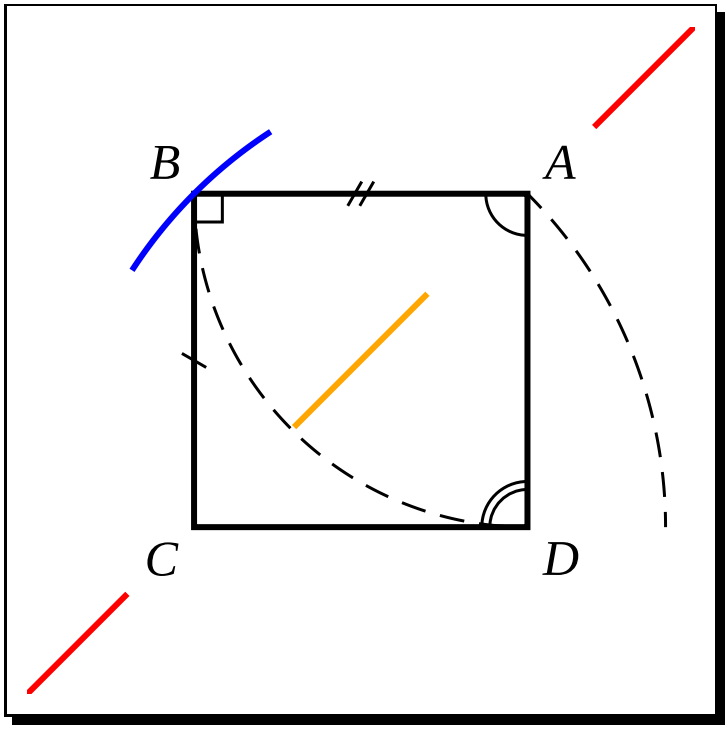
A B n **`marked`** – ⟶ marque le segment [*AB*] avec *n* traits inclinés

A B C **`angledroit`** – ⟶ dessine un angle droit en *B*

A B α **`trait`** – ⟶ calcule le point A' image de *A* par l'homothétie de centre *B* et de rapport $|\alpha|$, puis le point B' image de *B* par l'homothétie de centre *A* et de rapport $|\alpha|$. Si α est positif, trace la commande est équivalente à **`[A' B'] ligne`**, et si α est négatif, alors la commande invoque le tracé de la droite $(A'B')$ privée du segment $[A'B']$.

Ainsi que de la variable :

- *widthangledroit* : taille, en points postscript, d'un des côté de l'angle droit tracé par **`angledroit`**. **valeur par défaut :** 5



source jps

```
/A {2 2} def
/B {-2 2} def
/C {-2 -2} def

A B C parallelopoint /D defpoint

A D C .5 (=) tripointarc
B A D .5 (-) tripointarc

gsave
   pointilles
   A B D arcp
   C A D arcnp
grestore

1.5 setlinewidth
A B 2 marked
B C 1 marked
A B C angledroit
[A B C D] polygone

setTimesItalic
   (A) A urtext
   (B) B ultext
   (C) C dltext
   (D) D drtext

orange C A .7 trait
rouge C A -1.2 trait
bleu D B 12 Arc
```

2. Affichage de données discrètes

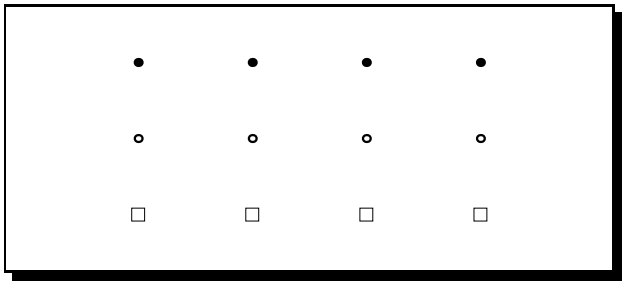
La commande **`plot`** permet de représenter des données discrètes. Utilisée avec un tableau de points comme seul argument, elle les représente avec la commande *dotstyle* (initialisée identique à la commande *point*).

En redéfinissant la commande *dotstyle*, on change le mode de représentation des points.

Une autre façon de changer le mode de représentation des points consiste à passer en plus en argument à la commande *plot* un exécutable à utiliser à la place de *dotstyle*.

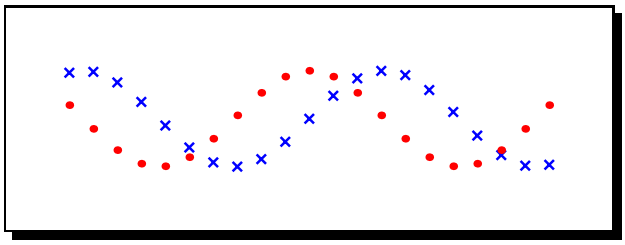
[A_0 A_1 … A_n] **`plot`** – ⟶ affiche les points A_i du tableau en utilisant la commande *dotstyle*

[A_0 A_1 … A_n] *proc* **`plot`** – ⟶ affiche les points A_i du tableau en utilisant la procédure *proc*



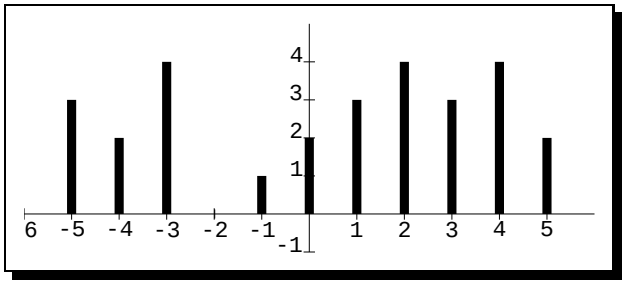
source jps

```
100 setheight
250 setwidth
-6 -1 setxrange
0 6 setyrange

%% par defaut
[-5 5 -4 5 -3 5 -2 5] plot

%% changer l'option de style
/dotstyle {circ} def
[-5 3 -4 3 -3 3 -2 3] plot

%% fusionner les tableaux
%% des abscisses et des ordonnees
[-5 -4 -3 -2] [1 1 1 1] fuz
{square2} plot
```



source jps

```
100 setheight
-2 2 setyrange
/f {Sin} def

bleu
[-5 5 .5 stepto] dup
(f) apply fuz {times3} plot
rouge
[-5 5 .5 stepto] {dup Cos} apply
{point} plot
```



source jps

```
250 setwidth
100 setheight
-1 5 setyrange
traceaxes
marks

[-5 5 1 stepto]
[3 2 4 0 1 2 3 4 3 4 2]
fuz {baton3} plot
```

3. Familles d'objets

Une famille d'objets est constitué par une table de ces objets. On dispose de quelques primitives permettant de manier ces tables.

On connaît déjà la fonction *apply* pour les tables de nombres. En fait cette fonction s'applique pour toute table d'objets unaires (les nombres, les tableaux, et donc les ellipses, les polygônes, etc. . .). On dispose également d'analogues pour les points, cercles et droites :

> [$a_0 \ldots a_n$] f **`apply`** [$b_0 \ldots b_n$] ou $- \longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer l'élément a_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.
>
> [$A_0 \ldots A_n$] f **`papply`** [$b_0 \ldots b_n$] ou $- \longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer le point A_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.
>
> **Exemple**

[0 0 1 1 2 2] {1} papply $\longrightarrow$ **[0 0 1 1 1 1 2 2 1]**
[0 0 1 1 2 2] {point} papply $\longrightarrow$ **-** applique la commande **point** aux points $(0,0)$, $(1,1)$ et $(2,2)$

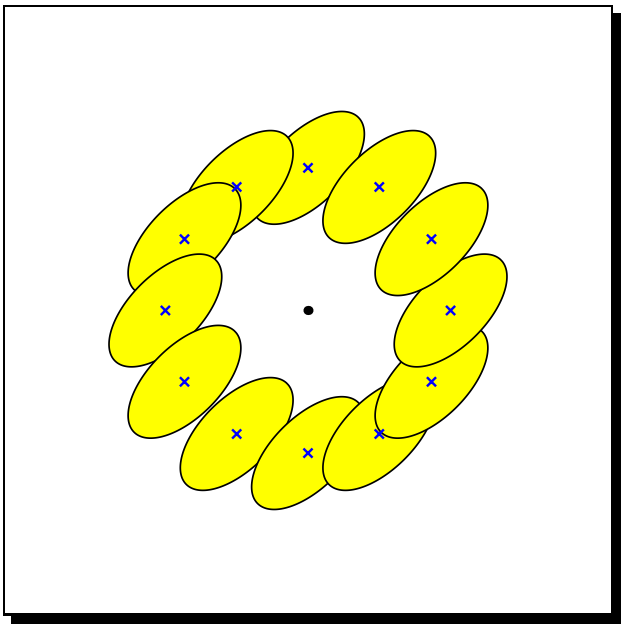
[$cerc_0 \ldots cerc_n$] f **capply** [$b_0 \ldots b_n$] ou $-$ $\longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer le cercle C_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

[$d_0 \ldots d_n$] f **dapply** [$b_0 \ldots b_n$] ou $-$ $\longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer la droite d_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

[$a_0 \ldots a_n$] f n_0 n_1 **Apply** [$b_0 \ldots b_n$] ou $-$ $\longrightarrow$ L'éxécutable f prend n_1 arguments et on décale de n_0 à chaque itération. Ainsi le premier paramètre de la 2ème itération est x_{n_0+1}. Par exemple, les commandes **3 3 Apply** et **capply** sont équivalentes.

le fichier jps

```
/A {0 3} def
/fillstyle {jaune fill} def
/ellipseangle 45 def

O point

[A 11 {2 copy O 30 rotatepoint} repeat] %% on genere la famille de points
dup                                     %% on en fait une copie
{1.5 .75} papply                        %% on genere la famille d'ellipse
(ellipse*) dapply                       %% que l'on represente
bleu
{times2} plot                           %% puis on marque les points
```

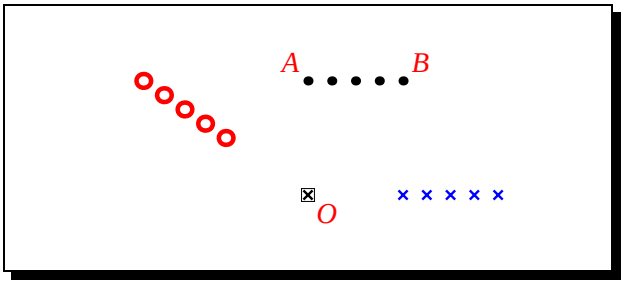


le fichier jps

```
100 setheight
250 setwidth
-1 3 setyrange

/A {0 2} def
/B {2 2} def

O times2                                %% on marque l'origine
O square2

[0 1 5 nto] {A B ABpoint} apply         %% on genere 5 points entre A et B
dup points                              %% on les represente avec des points
```

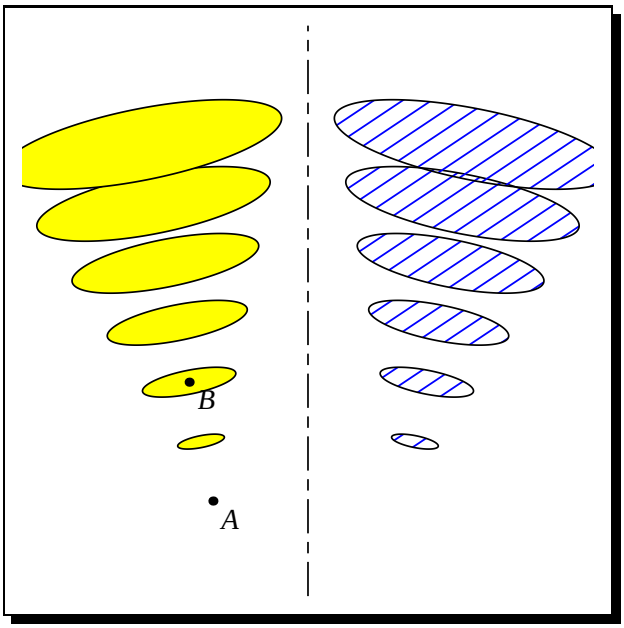
le fichier jps (suite)

```
{2 -2 translatepoint} papply          %% on les translate
bleu
dup {times2} plot                     %% on les represente avec des croix bleues
{O 150 rotatepoint} papply            %% rotation a 135 deg autour du point O
rouge
{circ2} plot                          %% on les represente avec des cercles rouges

setTimesItalic
   (A) A ultext
   (B) B urtext
   (O) O drtext
```



le fichier jps

```
/A {-2 -4} def
/B {-2.5 -1.5} def
/D {0 0 0 2} def                      %% la droite pour la symetrie
/ell [B .25 1 A B angle] def          %% l'ellipse originelle

/fillstyle {jaune fill} def


[.5 3 .5 stepto]                       %% genere la suite des rapports dans [0, 5]
{ell A 4 -1 roll homell} apply        %% genere la suite des
                                      %% ellipses axees sur (AB)
dup {ellipse*} apply                  %% on les dessine
{D axesymell} apply                   %% on en fait la symetrie
                                      %% par rapport a la droite D

/hcolor {bleu} def
/fillstyle {hachure} def
{ellipse*} apply                      %% puis on les dessine

mixte
D droite                              %% on represente D

[A B] points                          %% on marque les points A et B

setTimesItalic
   (A) A drtext
   (B) B drtext
```



4. Gestion étendue des courbes de Bézier

La gestion des courbes de Bézier par la commande **`bezier_curve`** est très efficace du point de vue des temps de calcul car elle est directement construite à partir de primitives postcript. Néammoins, il est nécessaire de connaître les points de contrôle pour l'utiliser, ce qui n'est pas vraiment souple.

Pour faciliter l'utilisation de ces courbes, on a essayé de reproduire les mécanismes en usage dans METAFONT et dans METAPOST. Cependant, une mauvaise interprétation de l'un des algorithmes de John Hobby m'a conduit à une mauvaise implémentation de cet algorithme, et il est parfois nécessaire d'utiliser l'incantation

```
/arg {argc} def
```

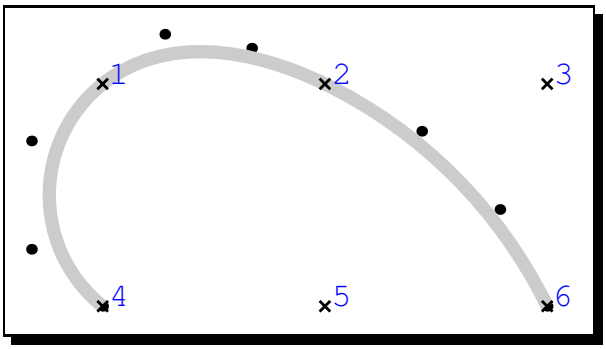
pour un fonctionnement correct de la commande **`draw`**.

> **Remarque** – Cet algorithme est utilisé à chaque fois qu'un des points clés ne comporte aucune direction spécifiée. Il peut alors arriver que la non-continuité de la fonction Arg produise des résultats étonnants. L'incantation proposée a pour but de remplacer la fonction non continue Arg par une fonction continue. Ce n'est hélas qu'un bricolage qui ne fonctionne pas toujours, et la réimplémentation de l'algorithme de Hobby est sur la liste des tâches qu'il me reste à accomplir. . .

4.1 - Cas général

La syntaxe la plus générale est **`[A1 .. A2 .. A3 .. A4] draw`** qui aura pour effet de tracer une courbe de Bézier passant par les points A_1, A_2, A_3 et A_4. Les points de contrôles sont calculés par le format et sont affichés ou non suivant l'option choisie (par **`withcontrolpoints`** ou **`withoutcontrolpoints`**).

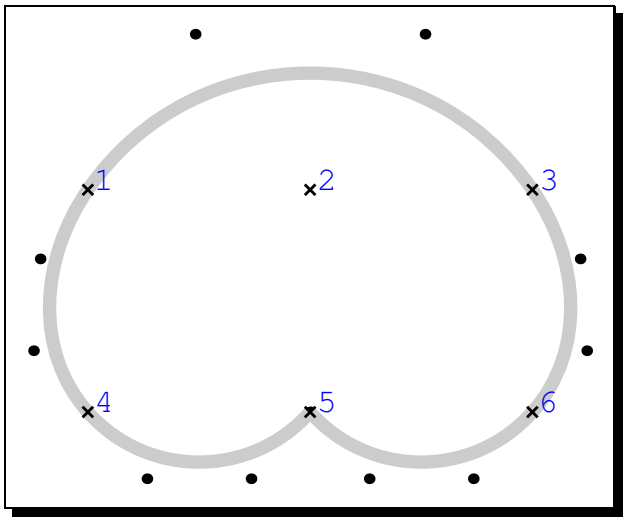
Voici un exemple qui fonctionne sans problème :



source jps

```
autocrop

-1 3 setxrange
-1 2 setyrange

/z1 {0 1} def   /z4 {0 0} def
/z2 {1 1} def /z5 {1 0} def
/z3 {2 1} def /z6 {2 0} def

withcontrolpoints

.8 setgray
5 setlinewidth

%% exemple 1 p 15 du Metafont Book,
[z4 .. z1 .. z2 .. z6] draw

noir
[z1 z2 z3 z4 z5 z6 ] {times2} plot
bleu
setCourier
   (1) z1 rtext   (4) z4 rtext
   (2) z2 rtext   (5) z5 rtext
   (3) z3 rtext   (6) z6 rtext
```

Et en voici un autre où l'incantation est nécessaire :



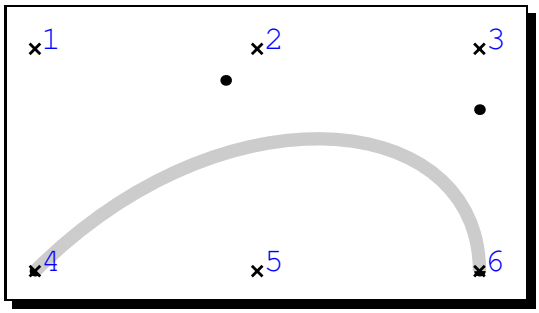
source jps

```
autocrop
-1 3 setxrange
-1 2 setyrange

/A1 {0 1} def   /A4 {0 0} def
/A2 {1 1} def /A5 {1 0} def
/A3 {2 1} def /A6 {2 0} def

withcontrolpoints

.8 setgray
5 setlinewidth

%% astuce temporaire pour corriger
%% un bug dans l'implementation du
%% calcul des directions aux points
%% cles : on rend la fonction arg
%% continue
/arg {argc} def

%% exemple 2 p 15 du Metafont Book,
[A5 .. A4 .. A1 .. A3 .. A6 .. A5] draw

noir
[A1 A2 A3 A4 A5 A6 ] {times2} plot
bleu
setCourier
   (1) A1 rtext   (4) A4 rtext
   (2) A2 rtext   (5) A5 rtext
   (3) A3 rtext   (6) A6 rtext
```

4.2 - Gestion des directions aux points clés

En chacun des points clés, il est possible d'imposer la direction de la tangente à droite ou a gauche (si cette tangente existe). Cette direction est donnée sous la forme d'un vecteur entre accolades à droite ou à gauche du point considéré. Quatres directions sont prédéfinies : **`up`**, **`down`**, **`right`** et **`left`**.

Dans l'exemple ci-dessous, la commande **`[A4 {A4 A2 vecteur} .. {down} A6] draw`** spécifie que la courbe à tracer comporte en A_4 et en A_6 des vecteurs tangents respectivement égaux à $\overrightarrow{A_4A_2}$, et à $(0;-1)$.

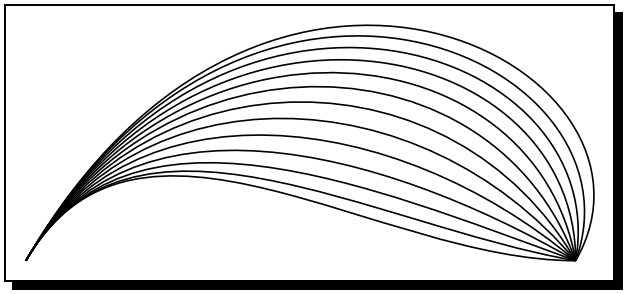


source jps

```
autocrop
-1 3 setxrange
-1 2 setyrange

/A1 {0 1} def   /A4 {0 0} def
/A2 {1 1} def /A5 {1 0} def
/A3 {2 1} def /A6 {2 0} def

withcontrolpoints

.8 setgray
5 setlinewidth

%% exemple 4 p 17 du Metafont Book
[A4 {A4 A2 vecteur} .. {down} A6] draw

noir
[A1 A2 A3 A4 A5 A6 ] {times2} plot
bleu
setCourier
   (1) A1 rtext   (4) A4 rtext
   (2) A2 rtext   (5) A5 rtext
   (3) A3 rtext   (6) A6 rtext
```

On peut également spécifier la direction par un angle (en degrés) grâce à la commande **`dir`** dont la syntaxe est

α **`dir`** $\vec{u} \longrightarrow \vec{u}$ est un vecteur correspondant à l'angle α (en degrés)



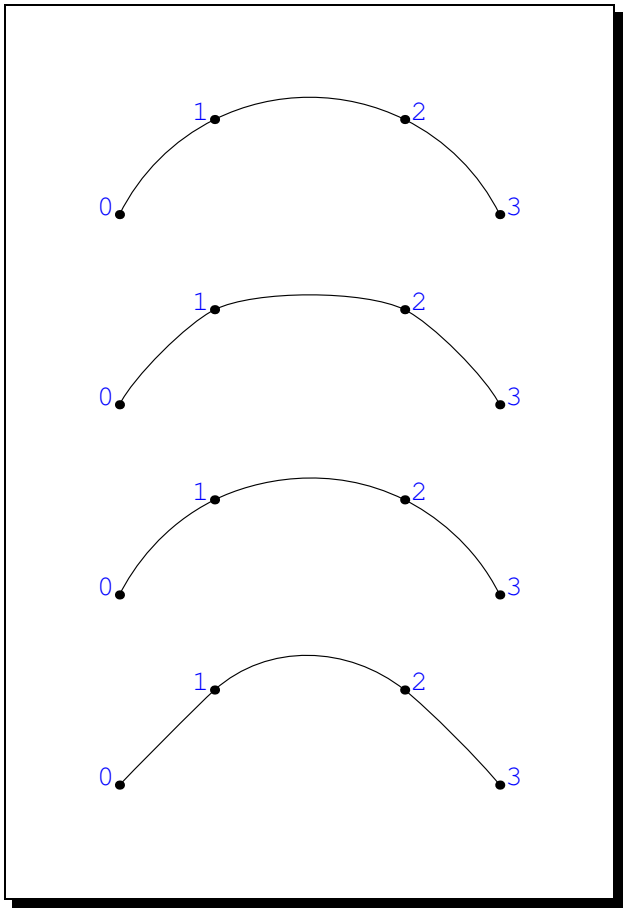
source jps

```
autocrop
-1 5 setxrange
-1 3 setyrange

%% exemple du Metafont Book p 18

/A {0 0} def
/B {4 0} def

/d 0 def

13 {
   [A {60 dir} .. {d neg dir} B] draw
   /d d 10 add store
} repeat
```

4.3 - Gestion des tensions

Entre deux points clés, le format calcule deux points de contrôle. Ce faisant il utilise deux paramètres qui sont les *tensions*. Celles-ci sont fixées à 1 par défaut, et usuellement données par la commande **`..`** définie par **`/.. {1 1} def`**. Ainsi, les commandes **`[A .. B .. C]`** et **`[A 1 1 B 1 1 C]`** sont équivalentes.

La modification des tensions permet une gestion plus fine des courbes obtenues :

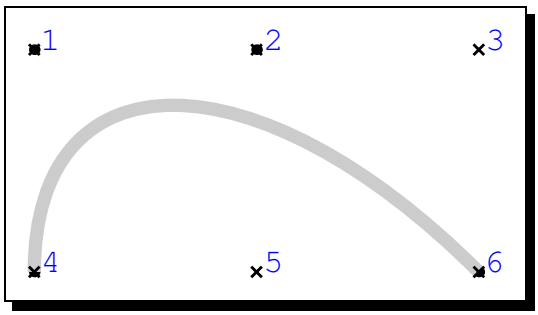


source jps

```
%% exemple inspire de la page 7 du
%% User's Manual for MetaPost

-.5 2.5 setxrange
-3.5 1 setyrange

/z0 {0 0} def    /z2 {1.5 .5} def
/z1 {.5 .5} def  /z3 {2 0} def
.5 setlinewidth

/la_legende {
   noir
   [z0 z1 z2 z3] points
   bleu
   setCourier
      (0) z0 ltext
      (1) z1 ltext
      (2) z2 rtext
      (3) z3 rtext
} def


%% les valeurs par defaut
gsave
   [z0 .. z1 .. z2 .. z3] draw
   la_legende
grestore

%% tensions plus fortes
gsave
   0 -1 stranslate
   /.. {1.5 1.5} def
   [z0 .. z1 .. z2 .. z3] draw
   la_legende
grestore

%% tensions desequilibrees
gsave
   0 -2 stranslate
   /.. {1.5 .8} def
   [z0 .. z1 .. z2 .. z3] draw
   la_legende
grestore

%% mise au point "manuelle"
gsave
   0 -3 stranslate
   %% retour a la situation par defaut
   /.. {1 1} def
   %% modif manuelle
   [z0 3 2 z1 .. z2 1.5 1.5 z3] draw
   la_legende
grestore
```

4.4 - Gestion directe des points de contrôle

Parfois, il peut s'avérer utilie de donner spécifiquement les points de contrôle entre deux points clés. Dans ce cas, on ne donne pas les tensions et on utilise la balise **`(controls)`** avant de préciser les points de contrôle.



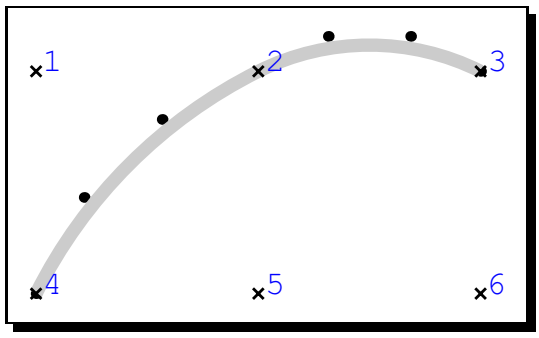
source jps

```
autocrop
-1 3 setxrange
-1 2 setyrange

/A1 {0 1} def       /A4 {0 0} def
/A2 {1 1} def       /A5 {1 0} def
/A3 {2 1} def       /A6 {2 0} def

withcontrolpoints

.8 setgray
5 setlinewidth

%% exemple 2 p 19 du Metafont Book
[A4 (controls) A1 A2 A6] draw

noir
[A1 A2 A3 A4 A5 A6 ] {times2} plot
bleu
setCourier
(1) A1 brtext   (4) A4 brtext
(2) A2 brtext   (5) A5 brtext
(3) A3 brtext   (6) A6 brtext
```

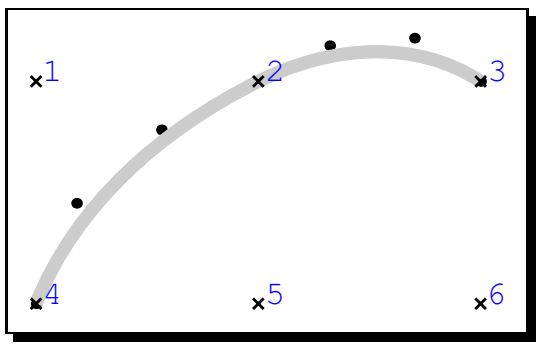
4.5 - Gestion des terminaisons de courbes

Il existe un paramètre permettant d'obtenir des terminaisons de courbes plus ou moins « tendues » : le paramètre **`curl`**, égal à 1 par défaut.

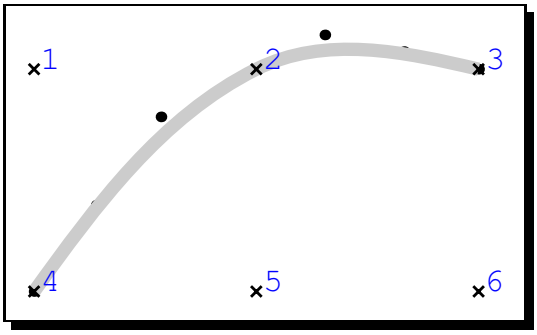
Par exemple la commande **`[A4  .. A2 {A4 A3 vecteur} .. A3] draw`** donnera



alors que **`[A4 {2 curl} .. A2 {A4 A3 vecteur} .. {2 curl} A3] draw`** (exemple 3 p 17 du Metafont Book) donnera



et que **`[A4 {0 curl} .. A2 {A4 A3 vecteur} .. {0 curl} A3] draw`** produira



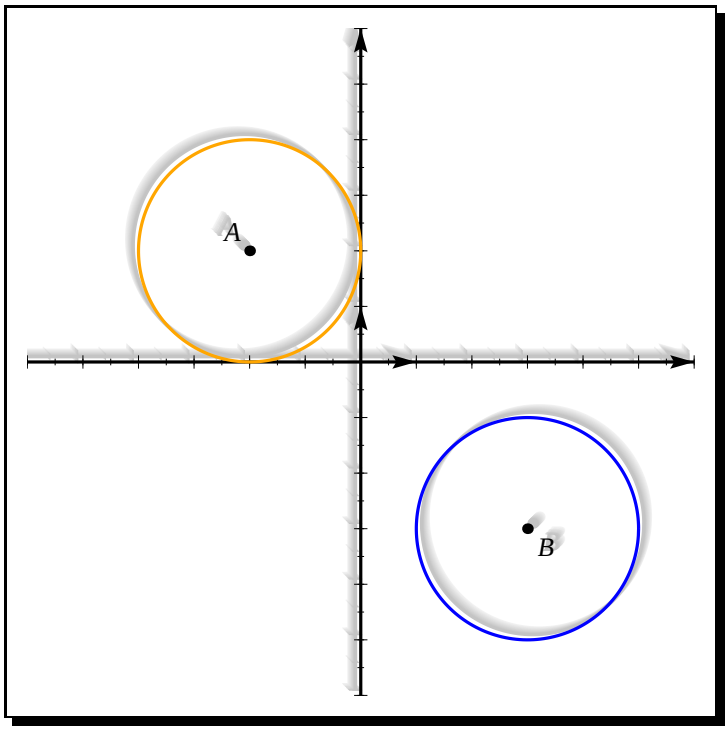
4.6 - Et la suite (?)

A priori il doît être possible d'implémenter une grande partie des commandes Metapost les plus usuelles. Reste à trouver le temps de le faire, d'autant qu'il faut également corriger le bug concernant la gestion des directions manquantes.

5. Effet de relief

La commande **`relief`** permet de faire facilement un effet graphique. Elle prend en argument l'exécutable qui produit la figure, et optionnellement 2 nombres indiquant la direction de l'ombre portée.

Par contre, l'utilisation de la commande **`setgray`** est proscrite dans le dessin à mettre en relief. Il faut donc s'en tenir aux commandes **`setrgbcolor`** et **`setcmykcolor`** (qui sont désactivées lors du tracé de l'ombre).



source jps

```
/A {-2 2} def
/B {3 -3} def

/dessin_1 {
   noir
   tracerepere
   A point
   (A) A ultext
   orange
   A 2 cercle
} def
/dessin_2 {
   bleu
   B 2 cercle
   noir
   B point
   (B) B drtext
} def

1.2 setlinewidth
setTimesItalic

%% par defaut
{dessin_1} relief
%% avec la direction imposee
{dessin_2} 1 1 relief
```

IV - Compléments

1. Les autres objets du format jps

1.1 - Les vecteurs

Un objet *vecteur* est défini par la donnée de 2 nombres, représentant ses coordonnées dans l'espace associé au repère jps. Par exemple **`-2 3`** représentera le vecteur de coordonnées $(-2, 3)$.

$A\ B$ **`vecteur`** $\vec{u} \longrightarrow A$ et B sont des points, et $\vec{u} = \overrightarrow{AB}$

$u\ u'$ **`addv`** $\vec{U} \longrightarrow \vec{U} = \vec{u} + \vec{u}'$ est la somme des vecteurs $\vec{u}$ et $\vec{u}'$

$u\ u'$ **`subv`** $\vec{U} \longrightarrow \vec{U} = \vec{u} - \vec{u}'$ est la différence des vecteurs $\vec{u}$ et $\vec{u}'$

$u\ a$ **`mulv`** $\vec{U} \longrightarrow \vec{U} = a\vec{u}$ où a est un nombre réel

$\vec{u}\ \vec{v}$ **`scalprod`** $\vec{u} \cdot \vec{v} \longrightarrow$ Le produit scalaire de $\vec{u}$ par $\vec{v}$

u **`norme`** $r \longrightarrow$ le réel $r = \|\vec{u}\|$

u **`normal`** $v \longrightarrow$ le vecteur v vérifie $\vec{u} \cdot \vec{v} = 0$. Plus présisément, si $\vec{u}(a, b)$ alors $\vec{v}(-b, a)$.

u **`arg`** $\theta \longrightarrow \theta \in]-180, 180]$ est l'angle que fait le vecteur $\vec{u}$ avec le vecteur unitaire de l'axe des abscisses

α **`dir`** $\vec{v} \longrightarrow \vec{v}$ est le vecteur de norme 1 d'angle $(\widehat{\vec{u}, \vec{v}}) = \alpha$ où $\vec{u}$ désigne le vecteur unitaire de l'axe des abscisses

– **`up`** $\vec{u} \longrightarrow \vec{u}$ est le vecteur $(0, 1)$

– **`down`** $\vec{u} \longrightarrow \vec{u}$ est le vecteur $(0, -1)$

– **`right`** $\vec{u} \longrightarrow \vec{u}$ est le vecteur $(1, 0)$

– **`left`** $\vec{u} \longrightarrow \vec{u}$ est le vecteur $(-1, 0)$

u **`normalize`** $v \longrightarrow v$ est le vecteur de norme 1 obtenu à partir de u, ie. $\vec{v} = \frac{1}{\|u\|}\vec{u}$

1.2 - Les nombres complexes

Un objet *nombre complexe* est défini par la donnée de 2 nombres, représentant ses parties réelles et imaginaire.. Par exemple **`-2 3`** représentera le nombre complexe $z = -2 + 3i$.

$z\ z'$ **`addc`** $Z \longrightarrow Z = z + z'$ est la somme des complexes z et z'

$z\ z'$ **`subc`** $Z \longrightarrow Z = z - z'$ est la différence des complexes z et z'

$z\ z'$ **`mulc`** $Z \longrightarrow Z = zz'$ est le produit des complexes z et z'

$z\ z'$ **`divc`** $Z \longrightarrow Z = z/z'$ est le quotient des complexes z et z'

z **`conjugue`** $\overline{z} \longrightarrow \overline{z}$ est le conjugué du complexe z

z **`module`** $r \longrightarrow$ le réel $r = |z|$

z **`arg`** $\theta \longrightarrow \theta = \mathrm{Arg}(z) \in]-180, 180]$

z **`nullc`** *bool* $\longrightarrow$ le booléen *bool* vaut **`true`** si le complexe z est nul, **`false`** sinon.

$z\ z'$ **`eqc`** *bool* $\longrightarrow$ le booléen *bool* vaut **`true`** si les complexes z et z' sont égaux, **`false`** sinon.

1.3 - Les tableaux

[$a_0 \ldots a_n$] [$b_0 \ldots b_n$] **`fuz`** [$a_0\ b_0 \ldots a_n\ b_n$] $\longrightarrow$ fusionne les 2 tableaux de même tailles donnés en entrée

[$a_0 \ldots a_n$] f **`apply`** [$b_0 \ldots b_n$] ou – $\longrightarrow$ construit un nouveau tableau en répétant l'opération suivante : déposer l'élément a_i puis exécuter f, pour i variant de 0 à n. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

Exemple

`[1 2 3] {1} apply` $\longrightarrow$ **`[1 1 2 1 3 1]`**
`[1 2 3] {dup mul} apply` $\longrightarrow$ **`[1 4 9]`**
`[1 2 3] {xtick} apply` $\longrightarrow$ **`-`** applique la commande **`xtick`** aux nombres 1, 2 puis 3

[$a_0 \ldots a_n$] *string* **`apply`** [$b_0 \ldots b_n$] ou – $\longrightarrow$ Comme la précédente, mais l'exécutable est cette fois désigné par une chaîne de caractères.

[$A_0 \ldots A_n$] i **`getp`** $A_i \longrightarrow$ donne le point d'indice i du tableau de points donné en entrée.

[$a_0 \ldots a_n$] **`sum`** $s \longrightarrow$ le réel s est la somme $s = \sum_{i=0}^{n} a_i$

[$a_0 \ldots a_n$] **moyenne** $m \longrightarrow$ le réel m est la moyenne arithmétique de la série des a_i. $m = \left(\sum_{i=0}^{n} a_i\right) / (n+1)$.

[$a_0 \ldots a_n$] **variance** $v \longrightarrow$ le réel v est la variance de la série des a_i.

[$a_0 \ldots a_n$] **ecarttype** $\sigma \longrightarrow$ le réel σ est l'écart-type de la série des a_i.

[$a_0 \ldots a_n$] **mediane** $m \longrightarrow$ le réel m est la médiane de la série des a_i.

$array_1$ **bubblesort** $array_2 \longrightarrow$ le tableau de réels $array_2$ est le résultat du tri à bulle sur le tableau de réels $array_1$.

$array_1$ $array_2$ **covariance** $c \longrightarrow$ le réel c est la covariance de la série double définie par les tableaux de réels $array_1$ et $array_2$

$array_1$ $array_2$ **correlation** $r \longrightarrow$ le réel r est la coefficient de corrélation de la série double définie par les tableaux de réels $array_1$ et $array_2$

$array_1$ $array_2$ **Mayer** $d \longrightarrow$ la droite d est la droite de Mayer définie par les tableaux de réels $array_1$ et $array_2$ définissant respectivement les abscisses et les ordonnées d'un nuage de points

$array_1$ $array_2$ **regyx** $d \longrightarrow d$ est la droite de régression des y en x de la série double définie par les tableaux de réels $array_1$ et $array_2$

$array_1$ $array_2$ **regxy** $d \longrightarrow d$ est la droite de régression des x en y de la série double définie par les tableaux de réels $array_1$ et $array_2$

$array_1$ **doublebubblesort** $array_2$ $array_3 \longrightarrow array_3$ est obtenu en triant $array_1$ par ordre croissant et $array_2$ correspond à la position des indices de départ dans le tableau d'arrivé, ie si $array_1 = [13, 12, 14, 11]$, alors $array_2 = [3, 1, 0, 2]$

1.4 - Les matrices

Une matrice (m, n) (m lignes et n colonnes) est représentée par un tableau de m tableaux. Chacun des m tableaux comportant lui-même n éléments. Tous ces tableaux sont indexés à partir de 0.

Attention, comme pour tous les objets complexes, il faut penser que postscript manipule le plus souvent des *pointeurs* (ou *références*) à ces objets plutôt que les objets eux-mêmes.

M **dimmatrix** $m\ n \longrightarrow$ dépose sur la pile les dimensions de la matrice M (m lignes, n colonnes)

$A\ x\ y\ M$ **printmatrix** $- \longrightarrow$ Affiche la matrice M. Le coefficient a_{00} est affiché en A, et on utilise les décalages $(x, 0)$ et $0, y$ pour les autres coefficients

M **dupmatrix** $M\ M' \longrightarrow$ dépose une nouvelle instance de M sur la pile

$m\ n$ **newmatrix** $M \longrightarrow$ dépose une nouvelle matrice nulle (m, n) sur la pile

$m\ n$ **idmatrix** $M \longrightarrow$ dépose une nouvelle matrice identité (m, n) sur la pile

$M\ i\ j\ any$ **put_ij** $- \longrightarrow$ affecte le coefficient a_{ij} de la matrice M à any

$A\ B$ **addm** $M \longrightarrow$ additionne les matrices A et B et dépose le résultat sur la pile

$A\ B$ **mulm** $M \longrightarrow$ multiplie les matrices A et B et dépose le résultat sur la pile

$M\ \alpha$ **smulm** $M' \longrightarrow M'$ est la matrice produit de la matrice M par le scalaire α

$M\ i$ **get_Ci** $array \longrightarrow$ le tableau $array$ représenta la colonne d'indice i de la matrice M

$M\ i$ **get_Li** $array \longrightarrow array$ représente la ligne d'indice i de la matrice M

$M\ i\ L$ **put_Li** $- \longrightarrow$ remplace dans la matrice M la ligne d'indice i par L

$M\ i\ j$ **get_ij** $a \longrightarrow a$ est le coefficient d'indice (i, j) de la matrice M

$M\ i\ j$ **exch_l** $- \longrightarrow$ échange les lignes d'indice i et d'indice j dans la matrice M

$B\ A$ **solve_syst** $X \longrightarrow A$ est une matrice carrée de déterminant non nul, B est un vecteur colonne, et X est le vecteur colonne solution de l'équation matricielle $AX = B$

M **view_square_matrix** $a_{1,1} \ldots a_{1,n}$ **()** $a_{1,1} \ldots a_{1,n}$ **()** $\ldots$ **()** $a_{n,1} \ldots a_{n,n} \longrightarrow$ dépose sur la pile les coefficients de la matrice carrée M

mulm

$$\begin{matrix}1 & 2\\ 4 & 5\\ 3 & 6\end{matrix} \quad x \quad \begin{matrix}1 & -1 & 2\\ 0 & 1 & 2\end{matrix} \quad = \quad \begin{matrix}1 & 1 & 6\\ 4 & 1 & 18\\ 3 & 3 & 18\end{matrix}$$

source jps

```
autocrop
15 setxunit
/L1 [1 2] def /L2 [4 5] def /L3 [3 6] def
/M [L1 L2 L3] def
/N [[1 -1 2] [0 1 2]] def
/Q M N mulm def
%% l'affichage
setCourierBold
(mulm) -1 1 crtext
setTimes
-4.5 0     1 -1 M printmatrix
(x)  -2.25 -1 cctext
-1   -.5   1 -1 N printmatrix
(=) 2.25 -1 cctext
3.5  0     1 -1 Q printmatrix
```

1.5 - Les chemins continus paramétrés

1.5.1 - L'objet *chemin continu paramétré*

Un objet *chemin continu paramétré* est une structure complexe représentant une application du type :

$$\begin{aligned} f &: I \to \mathbb{R}^2 \\ t &\mapsto (x, y) \end{aligned}$$

Cet objet possède 2 composantes : le tableau des valeurs du paramètre, et le tableau des points (x, y) correspondants. Dans la pratique, les coordonnées des points sont exprimées soit dans le repère jps, soit dans le repère postscript sous-jacent. Les commandes **`cppathtocpath`** et **`cpathtocppath`** permettent de passer d'un type de chemin à l'autre, et les commandes **`cpathpointstable`** et **`cpathparamtable`** permettent d'accéder en lecture à chacune des composantes de l'objet.

La commande **`drawcpath`** permet de dessiner un chemin continu paramétré passé en argument.

cpathobj$_1$ **`cppathtocpath`** *cpathobj*$_2$ $\longrightarrow$ transforme le chemin continu paramétré *cpathobj*$_1$ exprimé dans le repère postscript en le chemin continu paramétré *cpathobj*$_2$ exprimé dans le repère jps

cpathobj$_1$ **`cpathtocppath`** *cpathobj*$_2$ $\longrightarrow$ transforme le chemin continu paramétré *cpathobj*$_1$ exprimé dans le repère jps en le chemin continu paramétré *cpathobj*$_2$ exprimé dans le repère postscript

cpathobj **`cpathpointstable`** *array* $\longrightarrow$ *array* est le tableau de points du chemin continu paramétré *cpathobj*

cpathobj **`cpathparamtable`** *array* $\longrightarrow$ *array* est le tableau des paramètres du chemin continu paramétré *cpathobj*

cpathobj string **`drawcpath`** – $\longrightarrow$ Dessine le chemin continu représenté par *cpathobj*. *string* est un paramètre optionnel indiquant le type de terminaison de ligne

1.5.2 - Création d'un chemin continu paramétré

Chaque fois que l'une des commandes de tracé du format jps est exécutée (**`frame`**, **`droite`**, **`cercle`**, **`ellipse`**, **`courbe`**, **`courbeparam`**, **`sarc`**, **`sarcn`**, **`tripointarc`**, **`draw`**, **`bezier_curve`**), le chemin correspondant est sauvegardé sous forme de chemin continu paramétré, le paramètre appartenant à l'intervalle [0; 100]. Les commandes **`lastcpath`** et **`lastcppath`** permettent alors d'accéder à ce chemin, exprimé dans le repère jps ou dans le repère postscript sous-jacent.

On peut également créer un chemin continu paramétré à partir du chemin courant : **`stockcurrentcpath`** transforme le chemin continu courant en chemin paramétré et réaffecte les commandes **`lastcpath`** et **`lastcppath`**. On dispose également de 4 autres commandes permettant d'accéder, sans modifier les variables **`lastcpath`**, au chemin continu paramétré défini par le chemin continu courant.

– **`lastcpath`** *cpathobj* $\longrightarrow$ l'objet cpath, coordonnées dans le repère jps, associé au dernier chemin continu dessiné

– **`lastcppath`** *cpathobj* $\longrightarrow$ l'objet cpath, coordonnées dans le repère postcript, associé au dernier chemin continu dessiné

– **`stockcurrentcpath`** – $\longrightarrow$ sauvegarde le chemin continu courant sous forme de chemin continu paramétré, et réaffecte **`lastcpath`** et **`lastcppath`** en conséquence

– **currentcpathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère jps

– **currentcppathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère postscript

– **currentcpathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère jps)

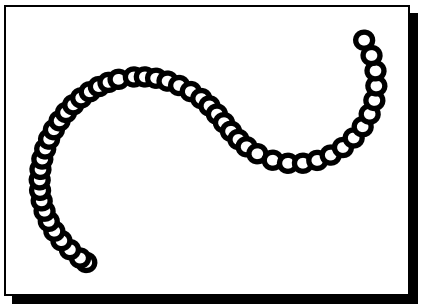
– **currentcppathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère postscript)

1.5.3 - Points d'un chemin continu paramétré

cpathobj **cpathstartpoint** A ⟶ A est le premier point du chemin continu paramétré *cpathobj*

cpathobj **cpathendpoint** B ⟶ B est le dernier point du chemin continu paramétré *cpathobj*

t cpathobj **cpathpoint** M ⟶ M est le point de paramètre t du chemin continu paramétré *cpathobj*



source jps

```
autocrop
[-3 -3 .. -1 0 .. 0 -1 .. 2 1] draw
/path1 lastcpath def
[0 2 100 {} for] {path1 cpathpoint circ2} apply
```

1.5.4 - Opérations sur les chemins paramétrés

$cpathobj_1$ **reversecpathobj** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu obtenu à partir de $cpathobj_1$ en inversant le sens de parcours

cpathobj **cpathlongueur** ℓ ⟶ longueur de chemin continu parmétré *cpathobj*

t cpathobj **cpathlongueurs** ℓ_1 ℓ_2 ⟶ ℓ_1 et ℓ_2 sont les longueurs respectives des sous-chemins $\overset{\frown}{AM}$ et $\overset{\frown}{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et où A et B sont respectivement les premier et dernier points de *cpathobj*

t cpathobj **cpathslongueur** ℓ ⟶ longueur du sous-chemin continu $\overset{\frown}{AM}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et A est le premier point de *cpathobj*

t cpathobj **cpathelongueur** ℓ ⟶ longueur du sous-chemin continu $\overset{\frown}{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et B est le dernier point de *cpathobj*

t cpathobj **splitcpath** $cpathobj_1$ $cpathobj_2$ ⟶ sépare, à partir du point de paramètre t, le chemin continu paramétré *cpathobj* en 2 sous-chemins paramétrés

$cpathobj_1$ **normalizecpath** $cpathobj_2$ ⟶ Les chemins décrits par $cpathobj_1$ et $cpathobj_2$ sont confondus, mais le tableau des paramètres de $cpathobj_2$ a été modifié, de telle facon que ceux-ci soient dans l'intervalle [0; 100], et qu'ils soient répartis de façon proportionnelle à la longueur du chemin

$cpathobj_1$ *u* **translatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la translation de vecteur *u*

$cpathobj_1$ **projxcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe *Ox*

$cpathobj_1$ **projycpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe *Oy*

$cpathobj_1$ *D* **orthoprojcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur la droite *D*

$cpathobj_1$ *I* α **rotatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la rotation de centr(e *I* et d'angle α

$cpathobj_1$ *I k* **homcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par l'homothétie de centre *I* et de rapport *k*

$cpathobj_1$ *I* **symcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la symétrie de centre *I*

$cpathobj_1$ *D* **axesymcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la symétrie axiale d'axe *D*

2. Chemin, définition d'un domaine plan

2.1 - Création d'un chemin

En postscript, pour faire un tracé, on procède en deux temps : tout d'abord on crée le chemin du tracé, puis on l'encre. Toutes les commandes *jps* vues jusque là réalisent les 2 opérations successivement. Mais ce chemin peut servir également à délimiter un domaine plan qui servira de masque.

2.1.1 - Commandes de construction de chemin

Pour créer un nouveau chemin, on utilise la commande postscript **`newpath`**. Il faut ensuite initialiser ce chemin en donnant un point courant avec les commandes **`moveto`** (relatif au repère postscript) ou **`smoveto`** (relatif au repère *jps*). Le tracé proprement dit est ensuite réalisé en donnant des ordres de déplacement au curseur. L'instruction **`closepath`** permet de fermer le chemin en court.

Plus précisément, on dispose des commandes postcsript suivantes (toutes relatives aux coordonnées dans le repère postscript lié à la Bounding Box) :

- **`newpath`** – ⟶ initialise et vide le chemin courant
- **`currentpoint`** $x\ y$ ⟶ renvoie les coordonnées du point courant

$x\ y$ **`moveto`** – ⟶ définit le point courant à (x, y)

$dx\ dy$ **`rmoveto`** – ⟶ **`moveto`** relatif

$x\ y$ **`lineto`** – ⟶ ajoute une ligne droite jusqu'en (x, y)

$dx\ dy$ **`rlineto`** – ⟶ **`lineto`** relatif

$x\ y\ r\ ang_1\ ang_2$ **`arc`** – ⟶ ajoute un arc dans le sens contraire des aiguilles d'une montre

$x\ y\ r\ ang_1\ ang_2$ **`arcn`** – ⟶ ajoute un arc dans le sens des aiguilles d'une montre

$x_1\ y_1\ x_2\ y_2\ r$ **`arct`** r ⟶ ajoute un arc tangent

$x_1\ y_1\ x_2\ y_2\ r$ **`arcto`** $xt_1\ yt_1\ xt_2\ yt_2$ ⟶ ajoute un arc tangent

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ **`curveto`** – ⟶ ajoute une section cubique de Bézier

$dx_1\ dy_1\ dx_2\ dy_2\ dx_3\ dy_3$ **`rcurveto`** – ⟶ **`curveto`** relatif

- **`closepath`** – ⟶ connecte le sous-chemin à son point de départ
- **`flattenpath`** – ⟶ convertit les courbes en suites de segments de droites
- **`reversepath`** – ⟶ renverse la direction du chemin courant

On dispose également d'un jeu de commandes *jps*, relatives cette fois au repère *jps* :

$x\ y$ **`smoveto`** – ⟶ définit le point courant à (x, y) dans le repère *jps*

$dx\ dy$ **`srmoveto`** – ⟶ **`smoveto`** relatif

$x\ y$ **`slineto`** – ⟶ ajoute une ligne droite jusqu'en (x, y) dans le repère *jps*

$dx\ dy$ **`srlineto`** – ⟶ **`slineto`** relatif

$x\ y\ r\ ang_1\ ang_2$ **`sarc`** – ⟶ ajoute un arc dans le sens contraire des aiguilles d'une montre (coordonnées dans le repère *jps*)

$x\ y\ r\ ang_1\ ang_2$ **`sarcn`** – ⟶ ajoute un arc dans le sens des aiguilles d'une montre (coordonnées dans le repère *jps*)

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ **`scurveto`** – ⟶ ajoute une section cubique de Bézier (coordonnées dans le repère *jps*)

$dx_1\ dy_1\ dx_2\ dy_2\ dx_3\ dy_3$ **`srcurveto`** – ⟶ **`scurveto`** relatif

2.1.2 - Commandes complémentaires pour la construction de chemin

Le format *jps* propose également des commandes « underscore » correspondant aux chemins des principaux objets du format. Ainsi, la séquance **`0 90 A 2 Cercle_`** ajoute au chemin courant le quart supérieur droit du cercle de centre *A* et de rayon 2.

Plus précisément, on dispose des commandes suivantes :

ell **`ellipse_`** – ⟶ ajoute au chemin courant le chemin de l'ellipse spécifiée

$\alpha\ \beta$ *ell* **`Ellipse_`** – ⟶ ajoute au chemin courant la portion de l'ellipse entre les points de paramètres respectifs α et β (dans ce sens).

cerc **`cercle_`** – ⟶ ajoute au chemin courant le cercle spécifié

$\alpha\ \beta$ *cerc* **`Cercle_`** – ⟶ ajoute au chemin courant la portion de cercle allant du point de paramètre α au point de paramètre β

A B **`frame_`** – ⟶ ajoute au chemin courant le rectangle dont les points *A* et *B* sont repectivement les coins inférieur droit et supérieur gauche

A L ℓ **rframe_** $-\!\!\longrightarrow$ ajoute au chemin courant le rectangle dont le point *A* est le coin inférieur droit, de dimension horizontale *L* et de dimension verticale ℓ

A L ℓ **cframe_** $-\!\!\longrightarrow$ ajoute au chemin courant trace le rectangle dont le point *A* est le centre, de dimension horizontale *L* et de dimension verticale ℓ

A L ℓ **mframe_** $-\!\!\longrightarrow$ ajoute au chemin courant le rectangle dont le point *A* est le milieu du côté inférieur, de dimension horizontale *L* et de dimension verticale ℓ

$\{f\}$ **courbe_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe représentative de la fonction f sur l'intervalle [*xmin*; *xmax*]

proc **courbe_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe représentative sur l'intervalle [*xmin*; *xmax*] de la fonction définie par l'exécutable *proc*

a b $\{f\}$ **Courbe_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe représentative de la fonction f pour *x* allant de *a* à *b*

a b proc **Courbe_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe représentative pour *x* allant de *a* à *b* de la fonction définie par l'exécutable *proc*

$\{X\}$ $\{Y\}$ **courbeparam_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle [*tmin*; *tmax*]

$proc_1$ $proc_2$ **courbeparam_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe paramétrée définie sur l'intervalle [*tmin*; *tmax*] par les exécutables $proc_1$ et $proc_2$

a b $\{X\}$ $\{Y\}$ **Courbeparam_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe paramétrée $t \mapsto (X(t); Y(t))$ pour *t* variant de *a* à *b*

a b $proc_1$ $proc_2$ **Courbeparam_** $-\!\!\longrightarrow$ ajoute au chemin courant la courbe paramétrée définie, pour *t* variant de *a* à *b*, par les exécutables $proc_1$ et $proc_2$

α β *A r* **wedge_** $-\!\!\longrightarrow$ ajoute au chemin courant la portion de camembert de centre *A*, de rayon *r*, délimité par les angles α et β

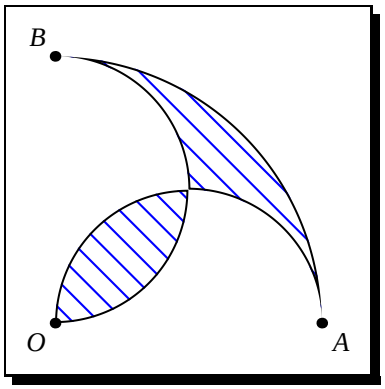
array **polygone_** $-\!\!\longrightarrow$ ajoute au chemin courant le polygône défini par le tableau de points *array*

array **ligne_** $-\!\!\longrightarrow$ ajoute au chemin courant la ligne définie par le tableau de points *array*

2.2 - Encrage d'un chemin, masquage

Une fois le chemin d'incrustation réalisé, la commande postscript **stroke** permet de l'encrer, alors que la commande **clip** réalise un masque, délimitant le domaine plan décrit.

Masques et chemins font partie de l'état graphique, et ils sont donc sauvegardés par les commandes **gsave** et **grestore**.

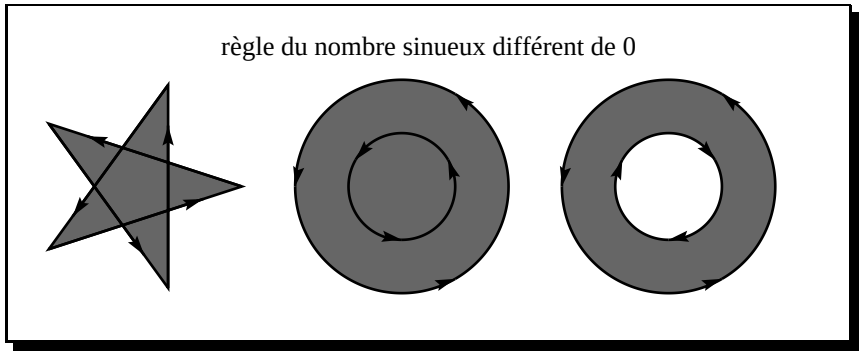


source jps

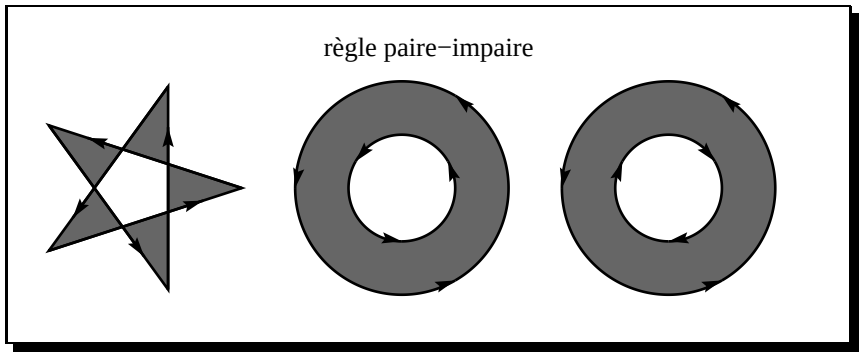
```
autocrop
25 setxunit
/a {4} def   /A {a 0} def   /B {0 a} def
%% la definition et le traitement du domaine plan
gsave
   %% definition du chemin
   newpath
      %% initialisation du point courant
      A smoveto
      %% le 1/4 de cercle AB
      0 90 O a Cercle_
      %% le 1/2 cercle BO
      90 -90 O B milieu a 2 div Cercle_
      %% le 1/2 cercle OA
      180 0 O A milieu a 2 div Cercle_
   clip      %% creation du masque
      bleu hachure
      1.5 setlinewidth noir
   stroke    %% encrage de la frontiere
grestore

[O A B] points

setTimesItalic
   (A) A drtext
   (B) B ultext
   (O) O dltext
```

Pour déterminer si un point est « à l'intérieur » ou non d'une région déterminée par le chemin courant, on peut choisir l'une ou l'autre des deux règles suivantes :

– **Règle du onmbre sinueux différent de zéro**
pour un point donné, on trace tous les rayons partant de ce point vers l'infini, puis, pour chaque rayon, on examine s'il coupe ou non le chemin. Plus précisément : on part de 0, et on ajoute 1 à chaque fois que le rayon coupe le chemin de gauche à droite, et on soustrait 1 à chaque fois que le rayon coupe le chemin de droite à gauche. Après avoir considéré tous les croisements, si le résultat est égal à zéro, alors le point est considéré *en dehors* de la région, sinon il est *à l'intérieur.*



– **Règle paire-impaire**
pour un point donné, on trace tous les rayons partant de ce point vers l'infini, puis, pour chaque rayon, on compte le nombre de segments de chemins traversés. Si ce nombre est pair, le point est considéré *à l'intérieur* de la région, et s'il est impair, le point est considéré *en dehors* de la région.



Pour le remplissage, l'opérateur **`fill`** utilise la couleur courante ou le motif courant pour peindre la région déterminée par le chemin courant, et c'est la règle du nombre sinueux différent de zéro qui est utilisée.

L'opérateur **`eofill`** fonctionne de la même façon que l'opérateur **`fill`**, mais en utilisant la règle paire-impaire. Et, de manière analogue, l'opérateur **`clip`** utilise la règle du nombre sinueux différent de zéro pour déterminer l'intérieur de la région à masquer, alors que **`eoclip`** utilise la règle du paire-impaire.

Pour le masquage, le format *jps* propose quelques opérateurs supplémentaires :

– **`masque_`** – ⟶ ajoute au chemin courant le rectangle de coin inférieur gauche le point (*xmin*, *ymin*) et de coin supérieur droit (*xmax*, *ymax*). Le chemin est parcouru dans le sens inverse des aiguilles d'une montre

– **`masque-_`** – ⟶ ajoute au chemin courant le rectangle de coin inférieur gauche le point (*xmin*, *ymin*) et de coin supérieur droit (*xmax*, *ymax*). Le chemin est parcouru dans le sens des aiguilles d'une montre

3. Opérateurs de pile

n p **`rollp`** – ⟶ considère la pile comme une file circulaire de *n* points, et la tourne de *p* crans

A **`dupp`** *A A* ⟶ duplique le point au dessus de la pile

cerc **`dupc`** *cerc cerc* ⟶ duplique le cercle au dessus de la pile

D **`dupd`** *D D* ⟶ duplique la droite au dessus de la pile

A **`popp`** – ⟶ enlève le point au sommet de la pile

cerc **`popc`** – ⟶ enlève le cercle au sommet de la pile

D **`popd`** – ⟶ enlève la droite au sommet de la pile

4. Opérateurs en géométrie

4.1 - Points

A B **milieu** *I* ⟶ le point *I* est le milieu du segment [*AB*]

A B **distance** ℓ ⟶ le nombre réel ℓ est la distance séparant les points *A* et *B*

A B C **parallelopoint** *D* ⟶ le point *D* tel que *ABCD* soit un parallélogramme

A B C **bissectrice** *D* ⟶ la droite *D* bissectrice de l'angle $\widehat{ABC}$

A B **ordonnepoints** *A′ B′* ⟶ range les points *A* et *B* par ordre d'ordonnée décroissante si possible, par ordre d'abscisse décroissante sinon

A B **angle** α ⟶ α est l'angle en dégré défini par le vecteur $\overrightarrow{AB}$ dans le repère **orthonormé** jps.

A B **pangle** α ⟶ α est l'angle en dégré défini par le vecteur $\overrightarrow{AB}$ dans le repère postscript

A B **eqp** *bool* ⟶ le booléen *bool* vaut **true** si les points *A* et *B* sont confondus, **false** sinon

4.2 - Droites

a **verticale** *D* ⟶ dépose sur la pile la droite verticale *D* d'équation $x = a$

b **horizontale** *D* ⟶ dépose sur la pile la droite horizontale *D* d'équation $y = b$

D **verticale?** *bool* ⟶ vrai si la droite *D* est verticale, faux sinon

D **coeffdir** *a* ⟶ *a* est le coefficient directeur de la droite *D* si celle-ci n'est pas verticale, erreur sinon

D **ordorig** *b* ⟶ *b* est l'ordonnée à l'origine de la droite *D* si celle-ci n'est pas verticale, erreur sinon

D A **perp** *D′* ⟶ *D′* est la droite perpendiculaire à la droite *D* passant par le point *A*

D A **paral** *D′* ⟶ *D′* est la droite parallèle à la droite *D* passant par le point *A*

A B **mediatrice** *D* ⟶ *D* est la médiatrice du segment [*AB*]

A B C **bissectrice** *D* ⟶ *D* est la bissectrice de l'angle $\widehat{ABC}$

D D′ **interdroite** *A* ⟶ si les droites *D* et *D′* sont sécantes, alors *A* est leur point d'intersection. Erreur sinon

x D **xdpoint** *A* ⟶ si la droite *D* n'est pas verticale, alors *A* est le point de *D* d'abscisse *x*. Erreur sinon

y D **ydpoint** *A* ⟶ si la droite *D* n'est pas horizontale, alors *A* est le point de *D* d'ordonnée *y*. Erreur sinon

OY *D* ⟶ dépose la droite $D = Oy$ sur la pile

OX *D* ⟶ dépose la droite $D = Ox$ sur la pile

4.3 - Cercles

A B **diamcercle** *cerc* ⟶ *cerc* est le cercle de diamètre [*AB*]

I A **IAcercle** *cerc* ⟶ *cerc* est le cercle de centre *I* passant par *A*

A B C **ABcercle** *cerc* ⟶ *cerc* est le cercle passant par les points *A*, *B* et *C*

D cerc **interdroitecercle** *A A′* ⟶ les points *A* et *A′* sont les points d'intersection de la droite *D* avec le cercle *cerc*, triés par la fonction *ordonnepoints*

$cerc_1$ $cerc_2$ **intercercle** *A A′* ⟶ les points *A* et *A′* sont les points d'intersection du cercle $cerc_1$ avec le cercle $cerc_2$, triés par la fonction *ordonnepoints*. Comme d'habitude, l'appel de cette fonction provoque une erreur si ces cercles n'ont pas de point commun.

Commandes de tracés

I A B **ABCercle** – ⟶ trace l'arc du cercle $\mathcal{C}$ inscrit dans l'angle $\widehat{AIB}$ où $\mathcal{C}$ désigne le cercle de centre *I* passant par *A*

I A B **ABCercle*** – ⟶ version étoilée de **ABCercle**

I A B **ABCercle_** – ⟶ version underscore de **ABCercle**

4.4 - Ellipses

ell **ellcentre** *A* ⟶ le points *A* est le centre de l'ellipse *ell*

ell **ellangle** α ⟶ α est l'angle de l'ellipse *ell*

ell **ella** *a* ⟶ *a* est la longueur du demi-axe horizontal de l'ellipse *ell*

ell **ellb** *b* ⟶ *b* est la longueur du demi-axe vertical de l'ellipse *ell*

ell **ell2pol** *pol* ⟶ le polygône *pol* est constitué des 4 sommets de l'ellipse

pol **pol2ell** *ell* ⟶ le polygône *pol* est constitué des 4 sommets de l'ellipse *ell*

D ell **interdroiteell** *A A'* ⟶ les points *A* et *A'* sont les points d'intersection de la droite *D* avec l'ellipse *ell*, triés par la fonction *ordonnepoints*

5. Transformations

5.1 - Translations

A $\vec{u}$ **translatepoint** *A'* ⟶ le point *A'* est l'image du point *A* par la translation de vecteur $\vec{u}$

D $\vec{u}$ **translatedroite** *D'* ⟶ la droite *D'* est l'image de la droite *D* par la translation de vecteur $\vec{u}$

ell $\vec{u}$ **translateell** *ell'* ⟶ l'ellipse *ell'* est l'image de l'ellipse *ell* par la translation de vecteur $\vec{u}$

pol $\vec{u}$ **translatepol** *pol'* ⟶ le polygône *pol'* est l'image du polygône *pol* par la translation de vecteur $\vec{u}$

cerc $\vec{u}$ **translatecercle** *cerc'* ⟶ le cercle *cerc'* est l'image du cercle *cerc* par la translation de vecteur $\vec{u}$

$pathobj_1$ *u* **translatepath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin image de $pathobj_1$ par la translation de vecteur *u*

5.2 - Rotations

A I α **rotatepoint** *A'* ⟶ le point *A'* est l'image du point *A* par la rotation de centre *I* et d'angle α

D I α **rotatedroite** *D'* ⟶ la droite *D'* est l'image de la droite *D* par la rotation de centre *I* et d'angle α

ell I α **rotateell** *ell'* ⟶ l'ellipse *ell'* est l'image de l'ellipse *ell* par la rotation de centre *I* et d'angle α

pol I α **rotatepol** *pol'* ⟶ le polygône *pol'* est l'image du polygône *pol* par la rotation de centre *I* et d'angle α

cerc I α **rotatecercle** *cerc'* ⟶ le cercle *cerc'* est l'image du cercle *cerc* par la rotation de centre *I* et d'angle α

$pathobj_1$ *I* α **rotatepath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin image de $pathobj_1$ par la rotation de centr(e *I* et d'angle α

5.3 - Homothéties

A I α **hompoint** *A'* ⟶ le point *A'* est l'image du point *A* par l'homothétie de centre *I*, de rapport α. Autrement dit $\overrightarrow{IA'} = \alpha\overrightarrow{IA}$

ell I α **homell** *ell'* ⟶ l'ellipse *ell'* est l'image de l'ellipse *ell* par l'homothétie de centre *I*, de rapport α.

pol I α **hompol** *pol'* ⟶ le polygône *pol'* est l'image du polygône *pol* par l'homothétie de centre *I*, de rapport α.

cerc I α **homcercle** *cerc'* ⟶ le cercle *cerc'* est l'image du cercle *cerc* par l'homothétie de centre *I*, de rapport α.

α *A B* **ABpoint** *A'* ⟶ le point *A'* est l'image du point *B* par l'homothétie de centre *A*, de rapport α. Autrement dit $\overrightarrow{AA'} = \alpha\overrightarrow{AB}$

$pathobj_1$ *I k* **hompath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin image de $pathobj_1$ par l'homothétie de centre *I* et de rapport *k*

5.4 - Projections

A **projx** *A'* ⟶ le point *A'* est le projeté orthogonal du point *A* sur l'axe *Ox*

$pathobj_1$ **projxpath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur l'axe *Ox*

A **projy** *A'* ⟶ le point *A'* est le projeté orthogonal du point *A* sur l'axe *Oy*

$pathobj_1$ **projypath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur l'axe *Oy*

A D **orthoproj** *A'* ⟶ le point *A'* est le projeté orthogonal du point *A* sur la droite *D*

$pathobj_1$ *D* **orthoprojpath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur la droite *D*

5.5 - Symétries centrales

A I **sympoint** *A'* ⟶ le point *A'* est le symétrique du point *A* par rapport au point *I*

ell I **symell** *ell′* ⟶ l'ellipse *ell′* est la symétrique de l'ellipse *ell* par rapport au point *I*

pol I **sympol** *pol′* ⟶ le polygône *pol′* est le symétrique du polygône *pol* par rapport au point *I*

cerc I **symcercle** *cerc′* ⟶ le cercle *cerc′* est le symétrique du cercle *cerc* par rapport au point *I*

pathobj$_1$ *I* **sympath** *pathobj*$_2$ ⟶ *pathobj*$_2$ est le chemin image de *pathobj*$_1$ par la symétrie de centre *I*

5.6 - Symétries axiales

A D **axesympoint** *A′* ⟶ le point *A′* est le symétrique du point *A* par rapport à la droite *D*

d D **axesymdroite** *d′* ⟶ la droite *d′* est la symétrique de la droite *d* par rapport à la droite *D*

ell D **axesymell** *ell′* ⟶ l'ellipse *ell′* est la symétrique de l'ellipse *ell* par rapport à la droite *D*

pol D **axesympol** *pol′* ⟶ le polygône *pol′* est la symétrique du polygône *pol* par rapport à la droite *D*

cerc D **axesymcercle** *cerc′* ⟶ le cercle *cerc′* est la symétrique du cercle *cerc* par rapport à la droite *D*

pathobj$_1$ *D* **axesympath** *pathobj*$_2$ ⟶ *pathobj*$_2$ est le chemin image de *pathobj*$_1$ par la symétrie axiale d'axe *D*

6. Méthodes numériques

6.1 - Équations

a b c **solve2nddegre** x_1 x_2 ⟶ Les réels x_1 et x_2 sont les racines réelles de l'équation $ax^2 + bx + c = 0$, où $a \neq 0$ et où $b^2 - 4ac \geqslant 0$

a b ε {*f*} **dich_solve** *x* ⟶ {*f*} est un exécutable, le produit $f(a) \times proc(b)$ est négatif, et ε est un réel strictement positif. Alors *x* est un réel tel que $f(x+\varepsilon) \times f(x-\varepsilon) < 0$

x_0 ε ε {*f*} {*f′*} **newton_solve** *x* ⟶ {*f*} et {*f′*} sont des exécutables, et f' désigne la fonction dérivée de f. Le réel *x* est obtenu par la méthode des tangentes de Newton, avec la valeur initiale x_0 et la tolérance ε

B A **solve_trig** *X* ⟶ *A* est une matrice triangulaire supérieure et *X* est l'unique vecteur solution de l'équation $AX = B$

B A **solve_syst** *X* ⟶ *A* est une matrice carrée de déterminant non nul et *X* est l'unique vecteur solution de l'équation $AX = B$

6.2 - Intégrales

a b {*f*} *n* **simpson** *real* ⟶ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode de Simpson pour $n+1$ points (*n* est un entier pair)

a b {*f*} *n* **methodetrapeze** *real* ⟶ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode des trapèzes pour $n+1$ points (*n* est un entier)

a b {*f*} ε **romberg** *real* ⟶ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode de Romberg pour une valeur de convergence fixée à ε

6.3 - Équations différentielles

x {*f*} **df-** *a* ⟶ Calcule une valeur approchée du nombre dérivé à gauche de la fonction f en *x*

x {*f*} **df+** *a* ⟶ Calcule une valeur approchée du nombre dérivé à droite de la fonction f en *x*

Ayant parfois eu quelques soucis avec les arrondis de mon interpréteur postscript, chacune des méthodes ci-dessous a été déclinée en deux versions suivant que l'on précise le pas *h* entre deux points calculés ou le nombre *n* de points à calculer. Pour l'interpréteur, la décision se fait en observant le type du dernier argument transmis (entier ou non).

f step$_1$ *step*$_2$ ℓ **champvecteur** − ⟶ Trace les vecteurs de norme ℓ définis par $y' = f(x,y)$, en partant de $(xmin, ymin)$ et jusqu'à $(xmax, ymax)$ et en tenant compte des pas $step_1$ (sur Ox) et $step_2$ (sur O_y)

a {*f*} x_0 y_0 *h* **baseeuler** x_0 y_0 x_1 y_1 ... x_n y_n ⟶ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[x_0, a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Plus précisement : $y_{i+1} = y_i + hy'_i$ où $y'_i = f(x_i, y_i)$, et $x_n = a$. Attention : on peut avoir $a < x0$, mais dans ce cas *h* doit être négatif

a {*f*} x_0 y_0 *n* **baseeuler** x_0 y_0 x_1 y_1 ... x_n y_n ⟶ *n* étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[x_0, a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas *h* est calculé en fonction de l'entier *n*.

a b {*f*} x_0 y_0 *h* **Euler** x_{-n} y_{-n} ... x_{-1} y_{-1} x_0 y_0 x_1 y_1 ... x_n y_n ⟶ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[a,b]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre *h* est un réel positif, il détermine le pas entre chaque point calculé

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Euler** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$\{f\}\ x_0\ y_0\ h$ **euler** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$\{f\}\ x_0\ y_0\ n$ **euler** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$a\ \{f\}\ x_0\ y_0\ h$ **baseeulermod** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas h doit être négatif

$a\ \{f\}\ x_0\ y_0\ n$ **baseeulermod** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas h est calculé en fonction de l'entier n.

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Eulermod** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Eulermod** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$\{f\}\ x_0\ y_0\ h$ **eulermod** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$\{f\}\ x_0\ y_0\ n$ **eulermod** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$a\ \{f\}\ x_0\ y_0\ h$ **baserungekutta** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas h doit être négatif

$a\ \{f\}\ x_0\ y_0\ n$ **baserungekutta** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas h est calculé en fonction de l'entier n.

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Rungekutta** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Rungekutta** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$\{f\}\ x_0\ y_0\ h$ **rungekutta** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$\{f\}\ x_0\ y_0\ n$ **rungekutta** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$a\ \{f\}\ x_0\ y_0\ h$ **basemilne** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas h doit être négatif

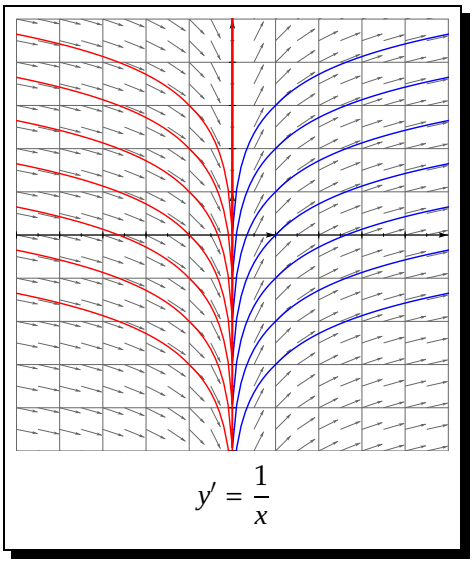
$a\ \{f\}\ x_0\ y_0\ n$ **basemilne** $x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas h est calculé en fonction de l'entier n.

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Milne** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Milne** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

$\{f\}\ x_0\ y_0\ h$ **milne** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

$\{f\}\ x_0\ y_0\ n$ **milne** $x_{-n}\ y_{-n}\ \ldots\ x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1\ \ldots\ x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.



source jps

```
-5 5 setxrange
-7 5 setyrange
-5 5 setyrange
30 setxunit

gsave
masque quadrillage
/f {
   pop 1 exch div
} def
.5 setlinewidth .4 setgray
/arrowscale {.5 dup} def
-5 -.1 setxrange
%% on separe en 2 pour eviter les pbms en x=0
%% rque : on aurait egalement pu prendre
%% -4.9 5 setxrange => decalage a droite de .1,
%% et on ne passe plus par x=0
-5 -.1 setxrange
{f} .5 .5 .5 (->) champvecteur
.5 5 setxrange
{f} .5 .5 .5 (->) champvecteur

-5 5 setxrange
/arrowscale {.75 dup} def
.7 setlinewidth
noir tracerepere 1 setlinewidth
%% on represente quelques solutions
-3 1 3 {
   /i exch def
   bleu
   [-5 Exp xmax {f} 1 i .1 Euler] ligne
   rouge
   [xmin -5 neg Exp {f} -1 i .1 Euler] ligne
} for

grestore
#tex# $y' = {1\over x}$
0 -6 [2 dup] cctexlabel
```

7. Échelles du repère

7.1 - Cas général

Lorsque l'utilisateur demande le tracé d'un point à partir de ses coordonnées dans le repère jps, le format calcule les coordonnées de ce point dans le repère postscript avant de le dessiner. Ce faisant, il utilise deux procédures (**xscale** et **yscale**, initialisées à l'identité) pour déterminer l'échelle à appliquer sur chaque axe.

Ainsi, si les deux procédures sont respectivement affectées aux fonctions numériques f et g, le point de coordonnées (x, y) dans le repère jps sera représenté au point de coordonnées $(f(x), g(y))$.

Néammoins, si l'on utilise ces changements d'échelle, il faut prendre garde de bien spécifier au script les xrange et yrange après transformation. Par exemple, si $x \in [a; b]$, il faudra calculer « manuellement » les valeurs $f(a)$ et $f(b)$ puis déclarer

$f(a)\ f(b)$ **setxrange**
$a\ b$ **setxrange**

pour avoir ce que l'on veut.

Aux deux procédures **xscale** et **yscale** correspondent les procédures réciproques **xscale-1** et **yscale-1**.

7.2 - Échelles logarithmiques et semi-logarithmiques

Pour les échelles logarithmiques, quelques macros sont proposées :

a b **log_seq** 10^a 2.10^a 3.10^a ... 9.10^a 10^{a+1} 2.10^{a+1} ... 9.10^{a+1} ... 9.10^b $\longrightarrow$ *a* et *b* sont des entiers. Génère une séquence pour une échelle logarithmique de graduations entre 10^a et 10^b

n **log_xmark** $-\!\!\longrightarrow$ *n* est un entier. affiche la numérotation correspondant à 10^n sur l'axe *Ox*

n **log_ymark** $-\!\!\longrightarrow$ *n* est un entier. affiche la numérotation correspondant à 10^n sur l'axe *Oy*

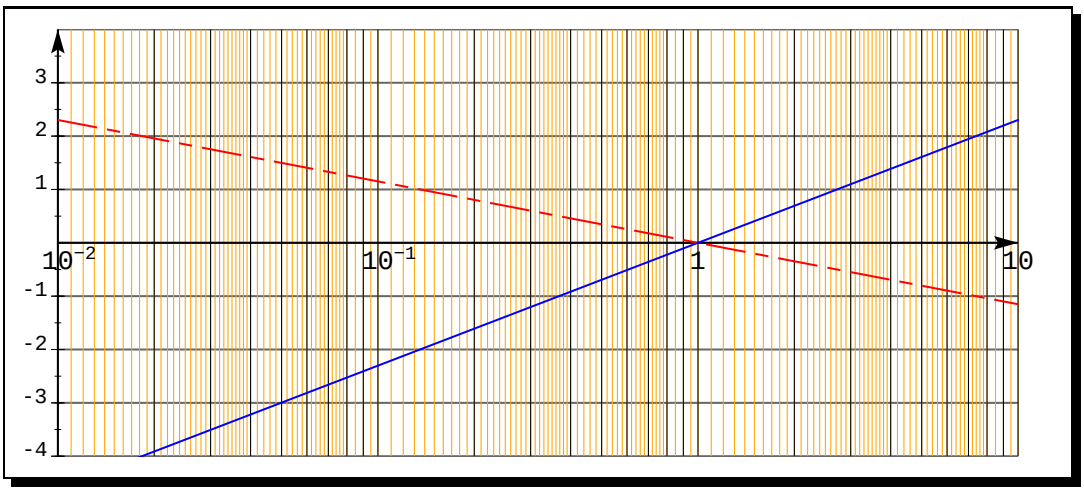
i **log_xbande** $-\!\!\longrightarrow$ *i* est un entier. trace 10 traits verticaux $\left]10^{i-1};10^{i}\right]$ en utilisant la commande **vrule**

i **log_ybande** $-\!\!\longrightarrow$ *i* est un entier. trace 10 traits horizontaux $\left]10^{i-1};10^{i}\right]$ en utilisant la commande **hrule**

i **log_bande** $-\!\!\longrightarrow$ *i* est un entier. trace 10 traits horizontaux et 10 traits verticaux sur $\left]10^{i-1};10^{i}\right]$ en utilisant les commandes **hrule** et **vrule**

le fichier jps

```
120 setxunit
20 setyunit

%% x est dans [ 0.01 ; 10], donc on regle le xrange
%% entre log (0.01) et log (10). (avec .1 de plus pour la legende)
%% Le .1 est a ajuster : il correspond a 10% de xunit
-2.1 1.1 setxrange
-4 4 setyrange

/xscale {log} def             %% on definit maintenant l'echelle sur Ox
/xscale-1 {10 ln mul Exp} def %% ainsi que l'echelle reciproque
0.01 10 setxrange             %% puis on regle le xrange avec la nouvelle echelle
0.01 0 setorigine             %% Enfin, on redefinit l'origine du repere

gsave               %% maintenant on raisonne avec la nouvelle echelle
   0.4 setgray      %% grillage horizontal
   [ymin truncate ymax truncate 1 stepto] {shrule} apply

   .4 setlinewidth  %% grillage fin vertical
   orange
   [-2 1 1 sub 1 stepto] {log_xbande} apply

   noir %% grillage simple vertical
   [-2 1 log_seq] {svrule} apply
grestore

yticks ysubticks ymarks           %% la numerotation sur l'axe Oy
[-2 1 1 stepto] {log_xmark} apply  %% la numerotation sur l'axe Ox
traceaxes
axesarrow

/f {setxvar
#rpn#   ln (x)
} def
/g {setxvar
#rpn#   ln (1 / sqrt (x))
} def

bleu 0.75 setlinewidth
%% on separe les intervalles pour avoir une meilleure resolution
%% (methode a pas fixe)
[xmin 1 1 10 10 xmax] {{f} Courbe} papply   %% trace de f
rouge mixte {g} courbe                      %% trace de g
```



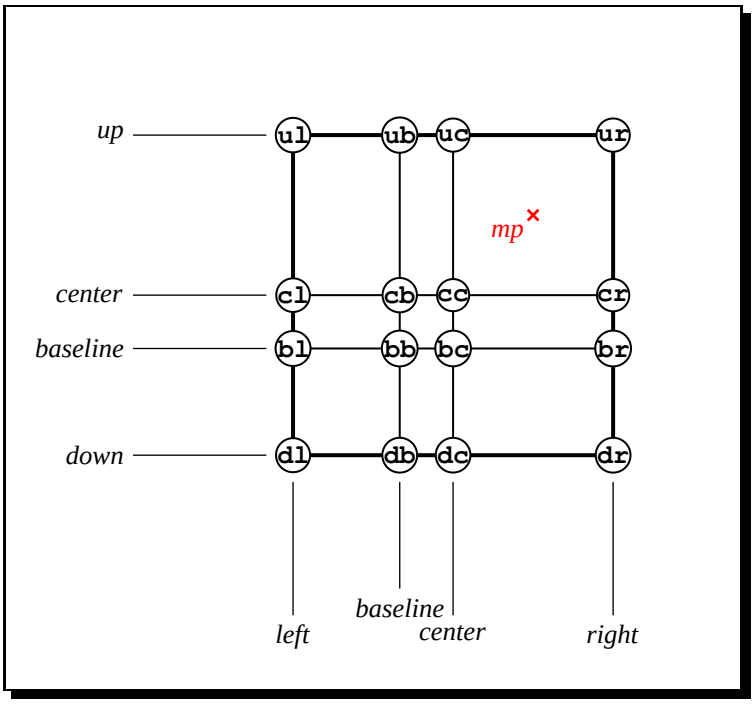
8. L'environnement 'picture'

Le but de cet environnement est de proposer un ensemble de commandes standardisées pour le positionnement des objets dans le repère *jps*. C'est en particulier lui qui gère le positionnement du texte ou des labels TEX.

L'utilisateur peut définir ses propres objets et utiliser toutes les facilités de placement ou d'encadrement proposées par l'environnement.

8.1 - Les points de référence

Chaque objet de l'environnement 'picture' est contenu dans un cadre rectangulaire : sa BoundingBox. Ce cadre définit 2 autres lignes : les médianes du rectangle. Pour finir, on considère la *baseline* sur chaque axe. Au total 4 lignes horizontales et 4 lignes verticales; leurs intersections nous donnent les 16 points de référence de l'objet considéré.



En plus de ces 16 points, vous voyez apparaître un point particulier sur le schéma ci-dessus. Il s'agit d'un point *spécial*, qui portera le nom **`/mp`** (*mon point*) pour les exemples qui vont suivre. Un objet peut comporter aucun ou plusieurs points spéciaux, et l'utilisateur peut en ajouter à des objets existants.

Si l'objet considéré est un texte, la baseline verticale sera souvent confondue avec le bord gauche du cadre (ligne *left*).

8.2 - Les commandes de positionnement

Les 17 commandes de positionnement se terminent toutes par **`pict`**, seul le préfixe change.

Les 4 préfixes *bb*, *bc*, *cb* et *cc* fonctionnent de la même façon et désignent le point de référence. Ainsi la commande

```
A (mon_objet) bbpict
```

signifie « dessiner l'objet *mon_objet* de telle sorte que son point *bb* soit exactement au point *A* ».

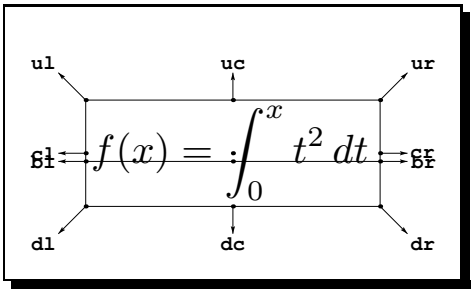
Les 12 préfixes standards qui restent désignent non plus les points de références, mais des directions. Ainsi la commande

```
A (mon_objet) ucpict
```

signifie « dessiner l'objet *mon_objet* dans la direction *uc* par rapport au point *A* ». La différence par rapport à précédemment, c'est que le format a tendance à rajouter un petit déplacement dans la direction adéquate. Ainsi, l'exemple précédent aura pour effet de placer le point *dc* de l'objet exactement au point *A*, puis d'ajouter un décalage vertical avant de tracer l'objet demandé.

Les composantes du décalage sont stockées dans les variables *hadjust* (décalage horizontal) et *vadjust*. Elles sont initialisées à $3,75$ par défaut et représentent des dimensions en points postscript.

La figure ci-dessous montre les directions de déplacement en fonction des préfixes, dans le cas particulier où l'objet est un label TEX.



Pour finir la commande **spict** permet de placer un point spécial. Ainsi la commande

```
A (mon_objet) /mp spict
```

signifie « dessiner l'objet *mon_objet* de telle sorte que son point *mp* soit exactement au point *A* ».

Si le point spécial n'est pas nommé, on peut utiliser la syntaxe

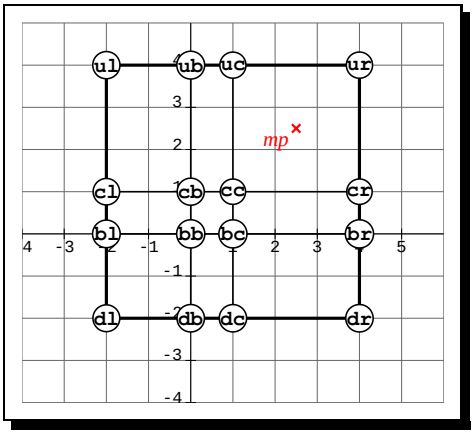
```
A (mon_objet) x y spict
```

où (x, y) représente les coordonnées du point spécial dans un repère *jps* dont le point *bb* de l'objet serait l'origine.

Ainsi, lorsque l'on reprend l'objet précédent, et qu'on le trace dans un repère gradué avec la commande **`0 0 (mon_objet) bbpict`**, on voit que le point *mp* a pour coordonnées $(2,5; 2,5)$. Les commandes

```
A (mon_objet) /mp spict
```
et
```
A (mon_objet) 2.5 2.5 spict
```

sont alors équivalentes.



8.3 - Les options

Chacune des commandes précédentes permet en option de faire subir à l'objet un décalage supplémentaire, un agrandissement/réduction et une rotation.

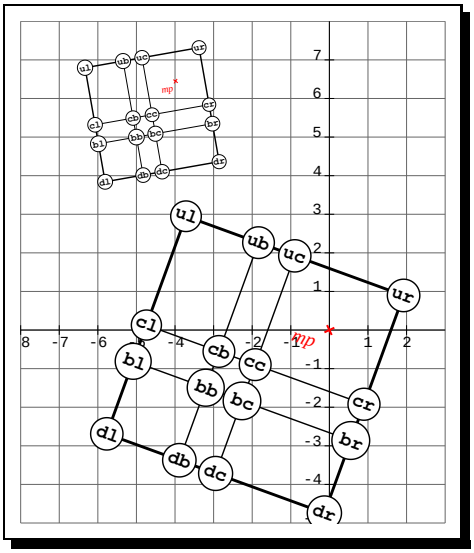
La syntaxe générale est

A (*dx dy*) [$scale_x$ $scale_y$] {α} **(mon_objet) bbpict**

où (*dx*, *dy*) est le déplacement en picas à rajouter au déplacement initial, ($scale_x$, $scale_y$) est le facteur d'agrandissement, et α l'angle de la rotation autour du point *A*. Il est également possible de donner la chaîne vide **()** pour le déplacement. Dans ce cas, le format annule tout déplacement; par exemple, la commande **A () (mon_objet) urpict** va placer le point *dl* de *mon_objet exactement* au point *A*.

Par exemple, la figure ci-dessous est obtenue avec les commandes

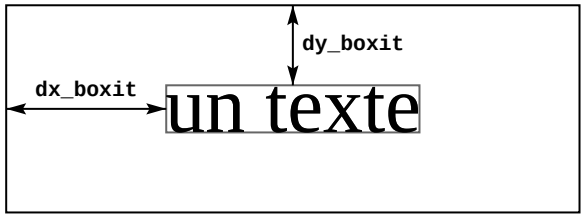
-5 5 [.5 dup] {10} (objet) bbpict et **O {-20} (objet) /mp spict**



8.4 - Mises en boites ou en cercles

Tous les objets affichés avec la famille de commandes **urpict**, **ucpict**, etc. . . peuvent être encadrés ou encerclés de différentes manières.

La commande **boxit** provoquera l'encadrement par un rectangle du prochain objet affiché. L'espace laissé entre le cadre et la boîte contenant l'objet est géré par les variables *dx_boxit* et *dy_boxit* comme l'indique le dessin ci-dessous :

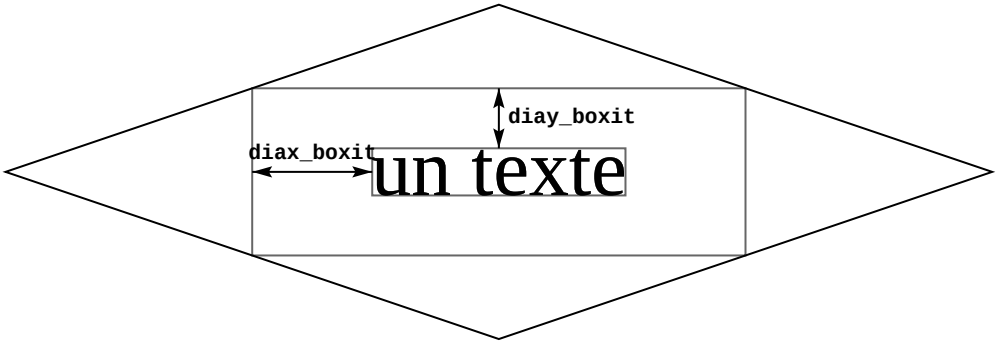


La commande **circleit** tracera un cercle passant par les 4 sommets du rectangle défini ci-dessus. L'espace laissé entre le cercle et la boite contenant l'objet est donc là encore géré par les variables **dx_boxit** et **dy_boxit** (dimensions en points postscript) initialisées à 0 par défaut.

La commande **Circleit** tracera un cercle de centre le centre de la boite contenant l'objet, et de rayon le contenu de la variable *Circleradius*. Cette commande permet ainsi d'encercler divers objets avec des cercles d'un rayon fixé.

La commande **diaboxit** provoquera l'encadrement par un losange du prochain objet affiché. L'espace laissé entre le cadre et la boîte contenant l'objet est géré par les variables *diax_boxit* et *diay_boxit* comme l'indique le dessin

ci-dessous :



Le losange est le losange d'aire minimale contenant la boite rectangulaire définie ci-dessus.

La commande **`ovalit`** tracera un ovale (une ellipse) passant par les 4 sommets du losange défini ci-dessus. L'espace laissé entre la boite incluse dans l'ellipse et la boite contenant l'objet est donc là encore géré par les variables **`diax_boxit`** et **`diay_boxit`** (dimensions en points postscript) initialisées à 0 par défaut.

La commande **`boxit_all`** (resp. **`circleit_all`**, **`Circleit_all`**, **`diaboxit_all`**, **`ovalit_all`**) provoquera l'encadrement de tous les objets suivants, et elle sera annulée par la commande **`boxit_none`** (resp. **`circleit_none`**, **`Circleit_none`**, **`diaboxit_none`**, **`ovalit_none`**).

Pour ces encadrements, le format utilise les versions étoilées des commandes **`cercle`**, **`ellipse`** et **`frame`**, ce qui autorise l'utilisation de **`fillstyle`** pour des cadres non transparents.

8.5 - Pour aller plus loin

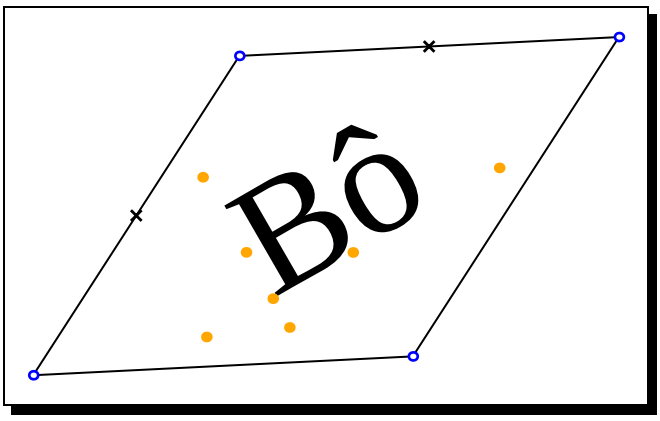
L'environnement 'picture' dispose d'un dictionnaire spécifique (le dictionnaire *Pictdict*), dans lequel il stocke les coordonnées des différents points de référence de l'objet affiché. Ces dernières sont réactualisées à chaque nouvel affichage.

Chaque objet affiché comporte au moins les 16 points de référence classiques, nommés *ul*, *ub*, *uc*, *ul*, *cl*, etc. . .

Si l'objet a été encadré par un losange ou une ellipse (une « *diabox* » ou une « *ovalbox* »), alors il y a 4 points de référence supplémentaires *udia*, *ddia*, *rdia* et *ldia* qui correspondent aux quatre sommets du losange ou de l'ellipse considérée (les 16 autres points de référence correspondant à ceux de la boite rectangulaire incluse dansle losange ou l'ellipse).

La commande **`pictget`** permet d'accéder à ces points. Sa syntaxe est la suivante

> *name* **`pictdict`** *x y* ⟶ dépose sur la pile les coordonnées associées au nom *name* dans le dictionnaire *Pictdict*



source jps

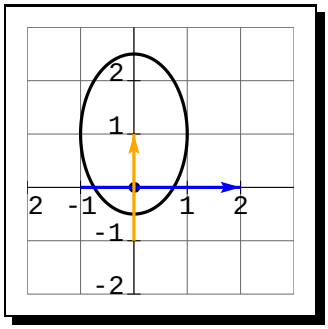
```
autocrop
setTimes
/diax_boxit 5 def
/diay_boxit 2 def
diaboxit
(Bô) 0 0 [6 dup] {30} cctext

/ul pictget times2 %% croix en ul
/ur pictget times2 %% croix en ur
bleu
[  %% ronds sur les sommets
   /rdia /ldia
   /udia /ddia
] {pictget circ} apply
orange
/dotscale {.8 dup}
[  %% points sur les 2 baseline
   /bl /bb /bc /br
   /ub /db /cb
] {pictget dot} apply
```

Il peut arriver que l'on ne souhaite pas le calcul et le stockage de ces 16 points de référence. On peut désactiver cette fonctionnalité avec la commande **`pictpointsOff`**. On la réactive avec la commande **`pictpointsOn`**.

8.6 - Mettre un objet dans l'environnement 'picture'

8.6.1 - Création d'un nouvel objet

On commence par faire un beau dessin que l'on veut réutiliser. Pour l'exemple, on le trace avec des coordonnées absolues dans le repère jps. Ici le dessin sera constitué d'une ellipse, de deux flèches (une bleue et une orange), et un point.



source jps

```
-2 3 setxrange
-2 3 setyrange
20 setxunit
quadrillage
traceaxes
marks

1.2 setlinewidth

%% on definit le dessin de
%% notre nouvel objet
0 0 point
0 1 1 1.5 ellipse
bleu
[-1 0 2 0] (->) ligne
orange
[0 -1 0 1 ] (->) ligne
```

Une fois l'objet dessiné, on relève les coordonnées, dans le repère jps, de sa Bounding Box associée. Dans notre exemple, on obtient les points $(-1;-1)$ et $(2;2,5)$

8.6.2 - Enregistrement d'un nouvel objet

On associe alors la procédure de dessin de l'objet à un littéral (**`/mon_dessin`**) dans notre exemple non sans avoir encapsulé le tout dans un **`gsave`**. . . **`grestore`**, et après avoir rajouté l'incantation **`currentpoint translate`** qui permet de tracer le dessin au point courant (l'environnement picture assure l'existence du point courant au moment de l'appel).

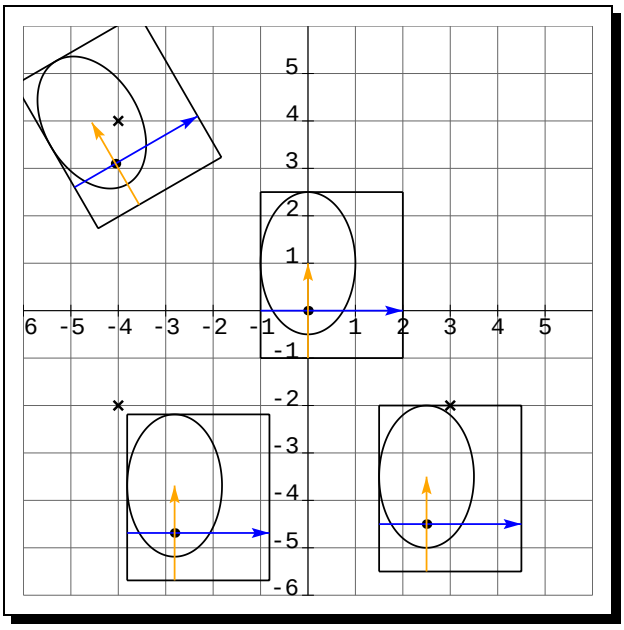
Ne reste plus qu'à créer une procédure dont le nom est celle du dessin du nouvel objet, suffixé par **`_dim`** (ce qui donnera **`mon_dessin_dim`** dans notre exemple), procédure qui est censee donner la BB du nouvel objet lorsqu'on l'appelle. Nota : cette BB doit être donnée dans le repère postscript. Si on ne la connaît qu'en coordonnées jps, il faut utiliser la fonction **`jtoppoint`** qui se chargera du changement de repere.

le fichier jps

```
20 setxunit
quadrillage
traceaxes
marks

%% on definit le dessin de notre nouvel objet
/mon_dessin {
gsave
   currentpoint translate      %% incantation a rajouter pour pouvoir
                               %% utiliser cette procedure dans
                               %% l'environnement 'picture'
   %% Maintenant le dessin
   0 0 point
   0 1 1 1.5 ellipse
   bleu
   [-1 0 2 0] (->) ligne
   orange
   [0 -1 0 1 ] (->) ligne
grestore
} def

%% puis la procedure donnant ses dimensions
/mon_dessin_dim {
   %% les dimensions, dans le repere ps, de la BB du dessin;
   -1 -1 jtoppoint
   2 2.5 jtoppoint
} def
```

le fichier jps (suite)

```
%% on encadre les dessins produits, pour mieux voir la Bounding Box
boxit_all
%% puis on place les dessins
0 0 (mon_dessin) bbpict                 %% point bb au point (0, 0)

-4 4 [1 dup] {30} (mon_dessin) ccpict   %% point cc au point (-4, 4)
                                        %% echelle (1, 1)
                                        %% rotation : 30 degre
-4 4 times2                             %% une croix pour bien voir (-4, 4)

-4 -2 (mon_dessin) drpict               %% dans la direction down/right
                                        %% par rapport au point (-4, -2)
                                        %% d'ou un leger decalage
                                        %% ajoute par le format
-4 -2 times2                            %% une croix pour bien voir (-4, -2)

3 -2 () (mon_dessin) dcpict             %% dans la direction down/center
                                        %% par rapport au point (3, -2)
                                        %% mais le () supprime le decalage
                                        %% ajoute par le format
3 -2 times2                             %% une croix pour bien voir (3, -2)
```



8.6.3 - Complément : objet avec arguments

Pour créer un nouvel objet *monobjet*, il faut donc écrire les deux procédures **mon_objet** et **monobjet_dim** qui seront utilisées par l'environnement *picture*. Dans cet environnement, on rappelle que la syntaxe générale d'un appel est

A (*dx dy*) [$scale_x$ $scale_y$] {α} **(monobjet) bbpict**

où *A* désigne le point de référnce pour l'affichage, et où les *dx*, *dy*, $scale_x$, $scale_y$ et α sont optionnels.

Imaginons que l'objet *monobjet* utilise deux arguments arg_1 et arg_2 et que sa taille dépende de ces arguments. La syntaxe d'appel deviendra alors

arg_1 arg_2 **A** (*dx dy*) [$scale_x$ $scale_y$] {α} **(monobjet) bbpict**

L'environnement va d'abord lire le nom de l'objet puis les options, avant d'appeler la procédure **monobjet_dim**. À ce moment, la pile sera la suivante :

arg_1 arg_2 **A**

La procédure **monobjet_dim** devant laisser la pile intacte, elle devra au préalable copier ses arguments (avec **4 copy** par exemple) si elle veut utiliser arg_1 ou arg_2.

Enfin, la procédure lit le point *A*, procéde aux ajustements nécessaires, puis appelle la procédure **monobjet**. À ce moment, la pile sera la suivante :

$$arg_1\ arg_2$$

et la procédure **monobjet** peut utiliser directement arg_1 et arg_2.

8.7 - Points spéciaux

Chaque objet peut disposer de son propre dictionnaire, dont le nom est celui de l'objet, suffixé par **_dic**. Ainsi, le dictionnaire éventuel associé à l'objet **mon_objet** sera **mon_objet_dic**. L'environnement 'picture' utilise lui le dictionnaire *Pictdic*. À chaque appel, on nettoie le dictionnaire *Pictdic*, puis on y copie le dictionnaire de l'objet. On fait ensuite subir à tous les points qui y sont définis les mêmes transformations qu'à l'objet.

Si le dictionnaire de l'objet n'existe pas, la commande

```
/mon_objet_dic 2 dict def
```

en créera un en lui réservant la place mémoire pour 2 points spéciaux.

À partir du moment où le dictionnaire est crée, la commande

```
mon_objet_dic /bb {0 0} put
```

définira alors le point *bb* de coordonnées $(0, 0)$ dans le repère jps lié à l'objet mon_objet.

Lors de l'appel au dessin d'un objet, l'environnement 'picture' fait donc subir à tous les points spéciaux les mêmes transformations qu'à l'objet, ce qui permet de les retrouver plus tard. Ainsi, si l'on demande le dessin de mon_objet, la commande

```
/bb pictget
```

déposera sur la pile les coordonnées du point *bb* **lors du dernier appel de l'environnement 'picture'**. Si l'on souhaite utiliser ces coordonnées plus tard, il faut les sauvegarder, par exemple par une commande du type

```
/bb pictget /bb defpoint
```

qui a pour effet de sauvegarder ces coordonnées sous le nom *bb* dans le dictionnaire courant.

Si on souhaite conserver tous les points spéciaux, la commande

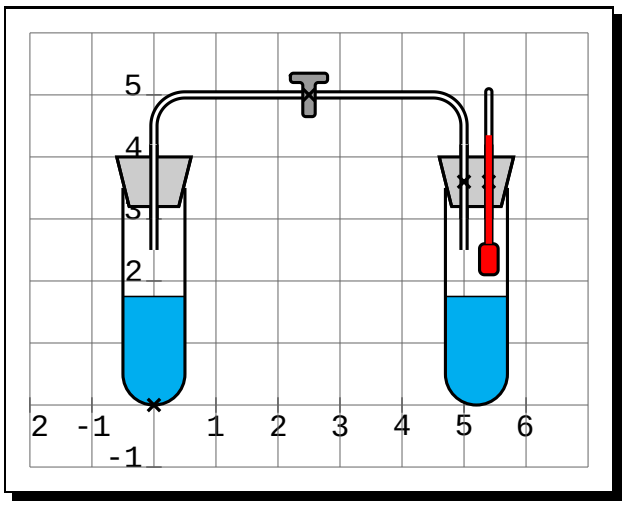
```
Pictdic currentdict copy
```

copiera l'intégralité du dictionnaire *Pictdic* dans le dictionnaire courant.

Ci-dessous un exemple d'application où tout le montage de chimie est lié par les points spéciaux.

le fichier jps

```
uselabo
20 setxunit
-2 7  setxrange
-1 6 setyrange
1 setlinewidth

1 setlinewidth
marks
quadrillage

/aspectLiquides [{cyan}] def
/niveauLiquides [50] def

%% on dessine le 1er tube a essais avec bouchon et tubeU
withbouchon
/Tubes [(TubeU)] def
O (TubeEssais) bbpict

O times2                                %% une croix sur O
/uc pictget /A defpoint                 %% on recupere le point /uc du tubeU et on le nomme A
/ConnectOut pictget times2              %% une croix sur /ConnectOut
                                               %% on dessine le 2eme tube a essais avec
/Tubes [(relax) dup] def                       %% un bouchon a 2 trous mais pas de tube
/ConnectOut pictget (TubeEssais) /trou1 spict  %% de telle facon que son point /trou1
                                               %% soit au point /ConnectOut du tubeU
/trou1 pictget times2                   %% une croix sur /trou1
/trou2 pictget times2                   %% une croix sur /trou2
/trou2 pictget (Thermometre) ccpict     %% on dessine le centre du Thermometre au point
                                        %% /trou2 du tube a essais

A (RobinetTube) ccpict                  %% puis le robinet au point /uc du tubeU,
                                        %% prealablement sauvegarde en A
A times2                                %% une croix sur A
```



9. Gestion des packages

Trois packages font partie de la distribution de *jps2ps* : *labo*, *logic* et *color*, que l'on appelle avec les instructions respectives **`uselabo`**, **`uselogic`** et **`usecolor`**.

Il est possible à d'ajouter des packages au format existant.

Pour rajouter un package dont le nom est *monpackage* (ce nom ne doit comporter que des caractères alphabétiques), procéder comme suit :

- Créer un fichier **`monpackage.pps`** contenant des définitions postscript et des définitions du format *jps*;
- Ajouter à ce fichier une ligne contenant l'intruction **`/usemonpackage {} def`**;
- Placer le fichier **`monpackage.pps`** dans l'arborescence *jps2ps/package/*;
- Exécuter le script **`jpshash.pl`** qui se chargera de mettre à jour la table des dépendances.

Le package est maintenant disponible, moyennant l'instruction **`usemonpackage`** qui déclenchera le chargement du package et des fichiers nécessaires à son fonctionnement.

V - Tracés en 3d

1. Tracés en 3d

1.1 - Définition du point de vue

L'ensemble de ces commandes a été écrit par Philippe Saadé (**psaade.jps@free.fr**).

Pour utiliser les commandes des paragraphes suivants, il faut au préalable déclarer la position de la caméra et calculer les vecteurs adéquats par **ComputeCamera**. Pour ce faire, on dispose des commandes suivantes :

$x\ y\ z$ **SetCamPos** – ⟶ Positionne la caméra au point (x, y, z)

– **GetCamPos** $x\ y\ z$ ⟶ Dépose sur la pile les coordonnées de la caméra

$V_x\ V_y\ V_z$ **SetCamVec** – ⟶ *Set Camera Looking vector.*

– **GetCamVec** $V_x\ V_y\ V_z$ ⟶ *Get Camera Looking vector.*

$U_x\ U_y\ U_z$ **SetCamUp** – ⟶ *Set Camera Up vector.*

– **GetCamUp** $U_x\ U_y\ U_z$ ⟶ *Set Camera Up vector.*

– **ComputeCamera** – ⟶ *Compute vectors usefull to CamView.*

$x\ y\ z$ **CamView** $X\ Y$ ⟶ On projete le point 3d sur le plan de représentation de la caméra, selon le mode de représentation

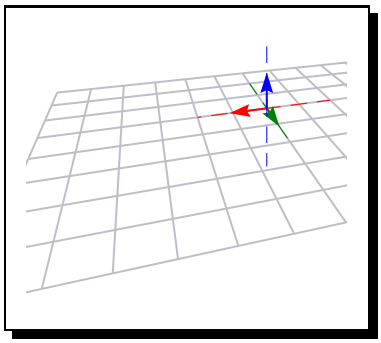
*array*1 **tab3dto2d** *array*2 ⟶ transforme un tableau de points 3d en tableau de points 2d

getp3d ⟶

- *representationtype* : Chaîne de caractère spécifiant le type de perspective : (perspective) ou (ortho). **valeur par défaut :** (perspective)
- *ScreenDist* : Distance par rapport à l'écran. **valeur par défaut :** 0.1
- *ZoomFactor_x* : Facteur de zoom en *x*. **valeur par défaut :** 100
- *ZoomFactor_y* : Facteur de zoom en *y*. **valeur par défaut :** 100

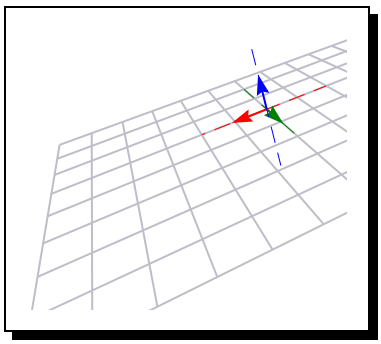
On rappelle que la position de la caméra est donnée par le point *CamPos*, et que son orientation est définie par les 2 vecteurs *CamVec* et *CamUp* qui déterminent la ligne de visée et le plan de visée.

Pour faciliter le positionnement de la caméra, on dispose maintenant de la commande **SetCamView** qui oriente la caméra vers un point fixé et qui recalcule le vecteur *CamUp* pour avoir une base orthonormale du plan de visée.



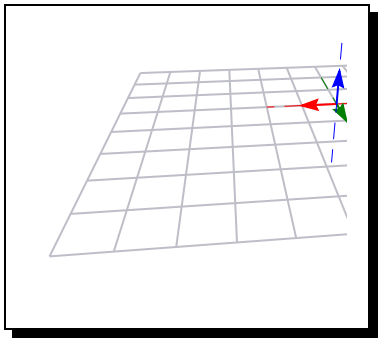
source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
0 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```



source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
0 60 cos 60 sin SetCamUp
0 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```



source jps

```
-6 2 setxrange
-5 2 setyrange
15 setxunit
3 10 5 SetCamPos
2 0 0 SetCamView
qplanxy
-2 2 1 axesRVB
```

De plus quelques nouvelles commandes sont venues enrichir la librairie 3d :

M **`SetCamView`** – ⟶ Oriente la visée de la caméra vers le point $M(x, y, z)$ et recalcule tous les vecteurs nécessaires

1.2 - Les axes et quadrillages

xmin xmax **`setxrange3d`** – ⟶ affecte les variables *xmin*3*d* et *xmax*3*d* définissant l'intervalle de travail sur l'axe *Ox* en 3*d*

ymin ymax **`setyrange3d`** – ⟶ affecte les variables *ymin*3*d* et *ymax*3*d* définissant l'intervalle de travail sur l'axe *Oy* en 3*d*

zmin zmax **`setzrange3d`** – ⟶ affecte les variables *zmin*3*d* et *zmax*3*d* définissant l'intervalle de travail sur l'axe *Oz* en 3*d*

– **`qplanxy`** – ⟶ Trace un quadrillage du plan XY

– **`qplanxz`** – ⟶ effectue un quadrillage du plan *xOz*

– **`qplanyz`** – ⟶ effectue un quadrillage du plan *yOz*

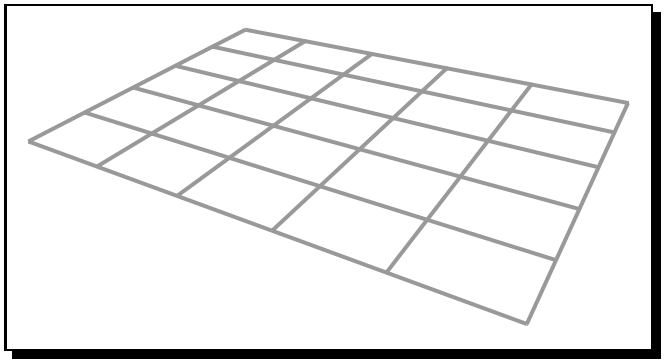
xmin xmax ymin ymax zmin zmax **`quadrilleXYZ`** – ⟶ Effectue un quadrillage d'unité 1 sur le produit $[xmin; xmax] \times [ymin; ymax] \times [zmin; zmax]$

xmin xmax ℓ **`axeR`** – ⟶ $[xmin; xmax]$ = étendue du pointille, ℓ = longueur du vecteur

ymin ymax ℓ **`axeV`** – ⟶ $[ymin; ymax]$ = étendue du pointille, ℓ = longueur du vecteur

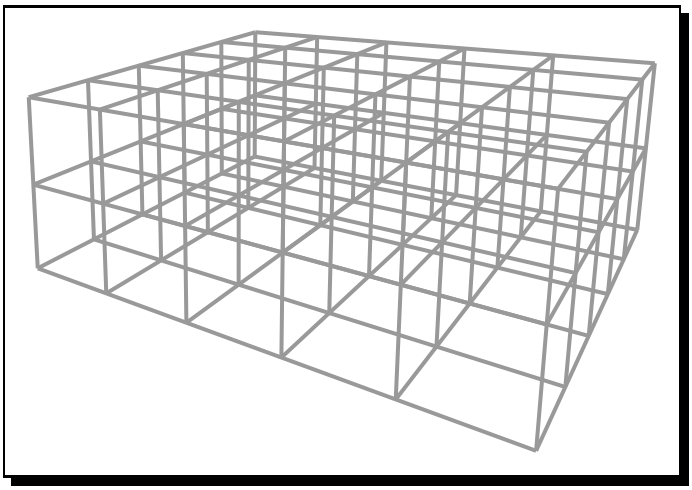
zmin zmax ℓ **`axeB`** – ⟶ $[zmin; zmax]$ = étendue du pointille, ℓ = longueur du vecteur

min max ℓ **`axesRVB`** – ⟶ $[min; max]$ = étendue des pointillés, ℓ = longueur des vecteurs



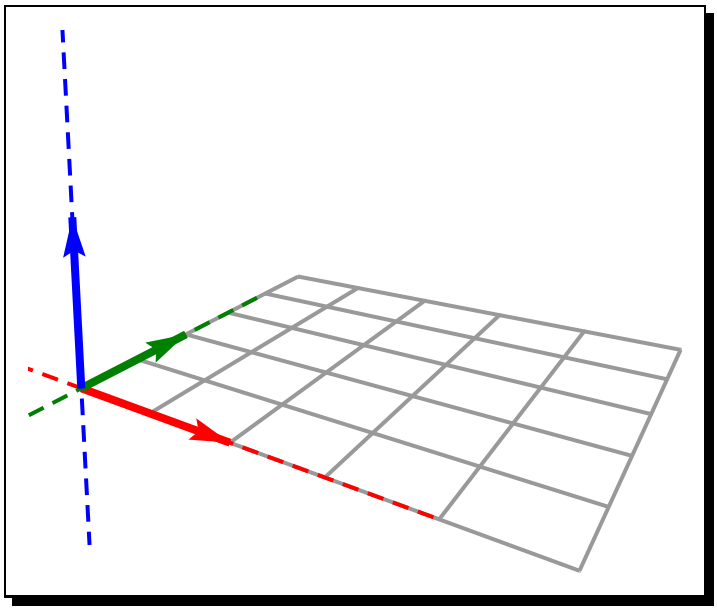
source jps

```
autocrop
%% échelle et étendue de l'image
30 setxunit
-1 9 setxrange
-5 2 setyrange
%% Positionnement de la Caméra
6 -6 4 SetCamPos
-1 1.1 -0.3 SetCamVec
0.03 0.03 1 SetCamUp
ComputeCamera
%% traces
1.5 setlinewidth
.6 setgray
0 5 0 5 0 0 quadrilleXYZ
```



source jps

```
autocrop
%% échelle et étendue de l'image
30 setxunit
-1 9 setxrange
-5 2 setyrange
%% Positionnement de la Caméra
6 -6 4 SetCamPos
-1 1.1 -0.3 SetCamVec
0.03 0.03 1 SetCamUp
ComputeCamera
%% traces
1.5 setlinewidth
.6 setgray
0 5 0 5 0 2 quadrilleXYZ
```



source jps

```
autocrop
%% échelle et étendue de l'image
30 setxunit
-1 9 setxrange
-5 2 setyrange
%% Positionnement de la Caméra
6 -6 4 SetCamPos
-1 1.1 -0.3 SetCamVec
0.03 0.03 1 SetCamUp
ComputeCamera
%% traces
1.5 setlinewidth
.6 setgray
0 5 0 5 0 0 quadrilleXYZ
3 setlinewidth
/arrowscale {2 2} def
-2 4 2 axesRVB
```

1.3 - Opérateurs

1.3.1 - Sur les points

[$A_0 \ldots A_n$] f **`papply3d`** [$b_0 \ldots b_n$] ou $- \longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer le point A_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

[$A_0 \ldots A_n$] **`isobarycentre3d`** $G \longrightarrow$ le point G est le barycentre du système $[(A_0, 1); \ldots; (A_n, 1)]$

[$A\ a\ B\ b$] **`barycentre3d`** $G \longrightarrow$ le point G est le barycentre du système $[(A, a); (B, b)]$

$M\ A\ \alpha$ **`hompoint3d`** $M' \longrightarrow M'$ est l'image de M par l'homothétie de centre A et de rapport α

$M\ A$ **`sympoint3d`** $M' \longrightarrow M'$ est l'image de M par la symétrie de centre A

$M\ u$ **`translatepoint3d`** $M' \longrightarrow M'$ est l'image de M par la translation de vecteur $\vec{u}$

$x\ y\ z\ k_1\ k_2\ k_3$ **`scaleOpoint3d`** $k_1x\ k_2y\ k_3z \longrightarrow$ opère une « dilatation » des coordonnées du point $M(x, y, z)$ sur les axes Ox, Oy et Oz suivant les facteurs k_1, k_2 et k_3

$M\ \alpha_x\ \alpha_y\ \alpha_z$ **`rotateOpoint3d`** $M' \longrightarrow M'$ est l'image de M par la rotation de centre O et d'angles respectifs $\alpha_x\ \alpha_y\ \alpha_z$ sur les axes Ox, Oy, Oz

$x\ y\ z$ **`3dto2d`** $X\ Y \longrightarrow$ calcule les coordonnées du point projeté sur l'écran pour la représentation 3d. cette commande est synonyme de la commande **`CamView`**

$A\ B$ **`distance3d`** $d \longrightarrow$ calcule la distance $d = AB$

A **`dupp3d`** $A\ A \longrightarrow$ Dupplique le point 3d au dessus de la pile

$x\ y\ z\ lit$ **`defpoint3d`** $- \longrightarrow$ Associe le littéral lit au point (x, y, z)

$M\ A\ \vec{v}$ **orthoprojplane3d** $M' \longrightarrow$ Le point M' est le projeté du point M sur le plan P défini par le point A et le vecteur $\vec{v}$, normal à P.

$A\ B$ **milieu3d** $I \longrightarrow I$ est le milieu de $[AB]$

1.3.2 - Sur les vecteurs

$A\ B$ **vecteur3d** $u \longrightarrow u = \overrightarrow{AB}$

$\vec{u}$ **norme3d** $r \longrightarrow r$ est la norme du vecteur $\vec{u}$

u **dupv3d** $u\ u \longrightarrow$ Dupplique le vecteur u au dessus de la pile

$\vec{u}\ \vec{v}$ **addv3d** $\vec{w} \longrightarrow \vec{w} = \vec{u} + \vec{v}$

$\vec{u}\ \lambda$ **mulv3d** $\vec{v} \longrightarrow \vec{v} = \lambda\vec{u}$

$\lambda\ \vec{u}$ **lambdav3d** $\vec{v} \longrightarrow$ Le vecteur $\vec{v}$ vérifie $\vec{v} = \lambda\vec{u}$

$\vec{u}\ \vec{v}$ **vectprod3d** $\vec{w} \longrightarrow \vec{w} = \vec{u} \wedge \vec{v}$

$\vec{u}\ \vec{v}$ **subv3d** $\vec{w} \longrightarrow \vec{w} = \vec{u} + \vec{v}$

$\vec{u}$ **unitaire3d** $\vec{v} \longrightarrow$ Si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

$\vec{u}$ **normalize3d** $\vec{v} \longrightarrow$ Sinonyme de **unitaire3d** : si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

$\vec{u}\ \vec{v}$ **scalprod3d** $s \longrightarrow$ Produit scalaire : $s = \vec{u} \cdot \vec{v}$

1.4 - Commandes de tracés

A **plus3d** $-\longrightarrow$ Analogue 3d de la commande **plus**

A **point3d** $-\longrightarrow$ Analogue 3d de la commande **point**

array **points3d** $-\longrightarrow$ Analogue 3d de la commande **points**

array **ligne3d** $-\longrightarrow$ Analogue 3d de la commande **ligne**

array **polygone3d** $-\longrightarrow$ Analogue 3d de la commande **polygone**

array **polygone*3d** $-\longrightarrow$ Analogue 3d de la commande **polygone***

$r\ \theta\ \phi$ **rtp2xyz** $x\ y\ z \longrightarrow$ Passage des coordonnées sphériques vers les coordonnées cartésiennes

$r\ \theta_1\ \phi_1\ r\ \theta_2\ \phi_2$ **arcspherique** $-\longrightarrow$ trace l'arc de cercle entre les points A et B de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

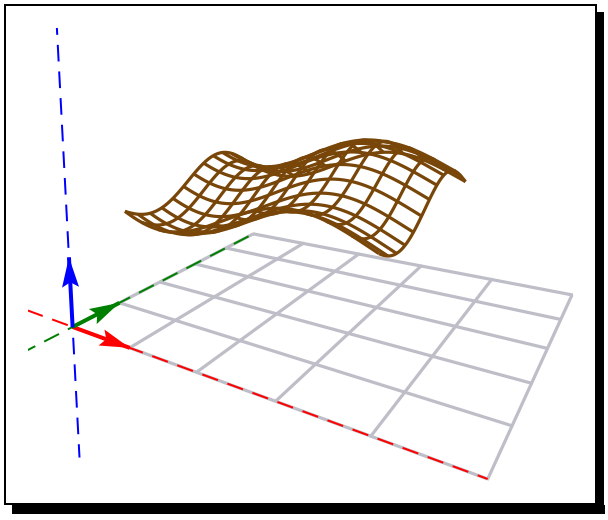
$r\ \theta_1\ \phi_1\ r\ \theta_2\ \phi_2$ **geodesique_sphere** $-\longrightarrow$ trace le cercle passant par les points A et B de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

$A\ B\ C$ **trianglespherique** $-\longrightarrow$ trace le triangle sphérique ABC, où les points A, B et et C sont donnés par leurs coordonnées sphériques respectives (r, θ_1, ϕ_1), (r, θ_2, ϕ_2) et (r, θ_3, ϕ_3)

$A\ B\ C$ **trianglespherique*** $-\longrightarrow$ version étoilée de **trianglespherique**

1.5 - Surfaces

xmin pas_x *xmax ymin* pas_y *ymax* f **surfaceparam3d** $-\longrightarrow$ Dessine la surface $f(x, y) = z$ sur $[xmin; xmax] \times [ymin; ymax]$. f est un éxécutable.



source jps

```
autocrop
%% échelle et étendue de l'image
25 setxunit
-1 9 setxrange
-5 2 setyrange
%% Positionnement de la Caméra
6 -6 4 SetCamPos
-1 1.1 -0.3 SetCamVec
0.03 0.03 1 SetCamUp
ComputeCamera
%% traces
1.2 setlinewidth
190 255 div
190 255 div
200 255 div setrgbcolor
0 5 0 5 0 0 quadrilleXYZ

1.5 setlinewidth
/arrowscale {1.5 dup} def
-2 5 1 axesRVB

%% la fonction z = f (x, y)
/f { % x y
2 dict begin
   /y exch def
   /x exch def
#rpn# Cos (x-y-1)* 0.5 * Cos (x+y+1) + 2
end
} def

1.2 setlinewidth
120 255 div
70 255 div
9 255 div setrgbcolor

/pas 0.25 def
1 pas 4 0 pas 3 {f} surfaceparam3d
```

1.6 - Placement de texte ou de labels TEX

On dispose des commandes analogues aux commandes 2d, la seule différence étant que le point de placement dispose de 3 coordonnées au lieu de 2.

Les 32 commandes disponibles sont donc :

`urtexlabel3d`	`uctexlabel3d`	`ubtexlabel3d`	`ultexlabel3d`
`crtexlabel3d`	`cctexlabel3d`	`cbtexlabel3d`	`cltexlabel3d`
`brtexlabel3d`	`bctexlabel3d`	`bbtexlabel3d`	`bltexlabel3d`
`drtexlabel3d`	`dctexlabel3d`	`dbtexlabel3d`	`dltexlabel3d`
`urtext3d`	`uctext3d`	`ubtext3d`	`ultext3d`
`crtext3d`	`cctext3d`	`cbtext3d`	`cltext3d`
`brtext3d`	`bctext3d`	`bbtext3d`	`bltext3d`
`drtext3d`	`dctext3d`	`dbtext3d`	`dltext3d`

2. Solides

Ce paragraphe présente un ensemble de macros dont le but est la manipulation et la représentation de solides dans une scène 3d. Un effort particulier a été fait pour respecter la philosophie de la programmation « orientée objet », avec contrôle de typage et messages d'erreurs adéquats.

Ce travail a été initié par les travaux de Manuel Luque [(1)] et de Christophe Poulain [(2)] ainsi que le cours d'infographie du Prof. Daniel Thalmann dans la section « *Computer Graphics* » du *Virtual Reality Lab* [(3)] .

Il a été poursuivi en collaboration avec Manuel Luque dans le but de réaliser le package *pst-solides3d* pour PSTricks.

(1) `melusine.eu.org/syracuse/mluque/pst-v3d/`
(2) `melusine.eu.org/syracuse/poulecl/macros/`
(3) `vrlab.epfl.ch/teaching/teaching_index.html`

2.1 - Le type *solid*

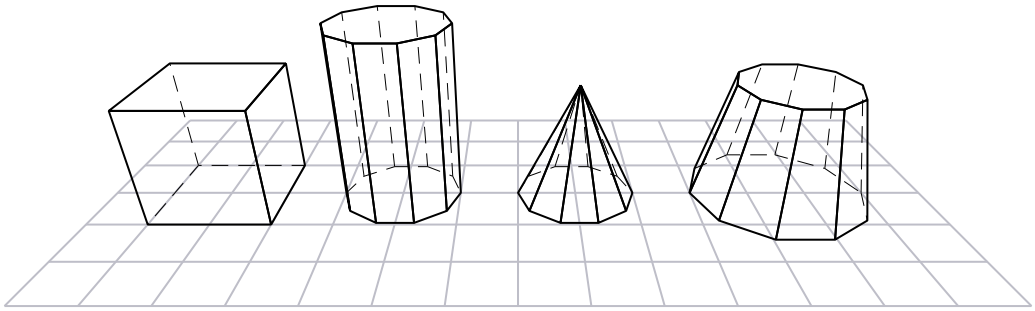
Le type *solid* est un type à plusieurs composantes. Les deux composantes essentielles sont :

- Un tableau des coordonnées (3d) des sommets du solides. Ce tableau est indexé à partir de l'indice 0.
- Un tableau des *faces* du solide. Ce tableau est indexé à partir de l'indice 0.

Et l'on désigne par *face* un tableau des indices des sommets de la face considérée, avec la convention importante suivante : les **sommets sont rangés dans l'ordre trigonométrique** si l'on regarde la face considérée depuis l'extérieur du solide.

2.2 - Solides précalculés

Certains solides simples sont précalculés : le cube, le cylindre, le cône, le tronc de cône, la sphère, le tétraèdre, l'octaèdre, l'icosaèdre, et le dodécaèdre.



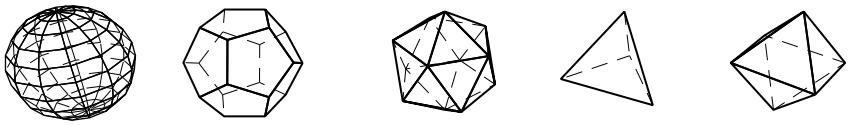
Voici la liste des commandes associées :

a option **newcube** *solid* ⟶ crée un nouveau cube, de type *solid*, de centre *O*, d'arête *a*. *option* indique le maillage ou le mode.

z_0 *r* z_1 *option* **newcylindre** *solid* ⟶ crée un nouveau cylindre, de type *solid*, d'axe *Oz*, de rayon *r*, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

z_0 *r* z_1 *option* **newcone** *solid* ⟶ crée un nouveau cône, de type *solid*, d'axe *Oz*, de rayon *r*, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

z_0 r_0 z_1 r_1 *option* **newtronccone** *solid* ⟶ crée un nouveau tronc de cône, de type *solid*, d'axe *Oz*, de rayon de base r_0 (sur le plan $z = z_0$) et de rayon au sommet r_1 (sur le plan $z = z_1$). *option* indique le maillage ou le mode.



sphere **dodecaedre** **icosaedre** **tetraedre** **octaedre**

r option **newsphere** *solid* ⟶ crée une nouvelle sphère, de type *solid*, de centre *O*, de rayon *r*. Le paramètre $mode \in \{0, 1, 2, 3, 4\}$ est optionnel; il indique le niveau de résolution souhaité (0 = mini, 4 = maxi). *option* indique le maillage ou le mode.

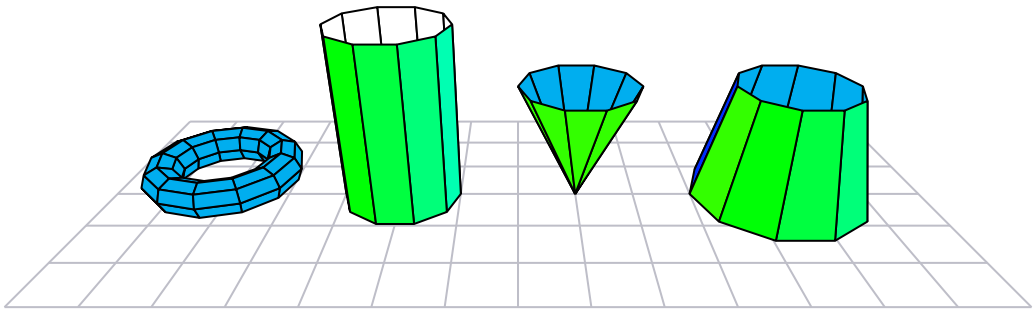
r **newtetraedre** *solid* ⟶ crée un nouveau tétraèdre régulier, de type *solid*, inscrit dans la sphère de centre *O* et de rayon *r*

r **newoctaedre** *solid* ⟶ crée un nouvel octaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

r **newicosaedre** *solid* ⟶ crée un nouvel isocaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

r **newdodecaedre** *solid* ⟶ crée un nouveau dodécaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

D'autres solides sont disponibles : le tronc de cône creux, le cylindre creux, le cône creux, le tore, le vecteur, la calotte sphérique et la calotte sphérique creuse :



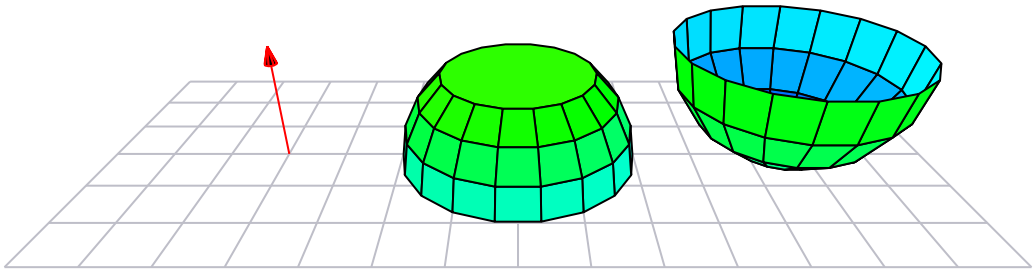
r R option **`newtore`** *solid* ⟶ crée un nouveau tore, de type *solid*, de centre *O*, de rayon principal *R*, et de rayon du tube *r*. *option* indique le maillage ou le mode.

z_0 z_1 r_1 *option* **`newcylindrecreux`** *solid* ⟶ crée un nouveau cylindre creux, de type *solid*, d'axe *Oz*, de rayon *r*, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

z_0 *r* z_1 **`newcone`** *solid* ⟶ crée un nouveau cône, de type *solid*, d'axe *Oz*, de rayon *r*, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

z_0 r_0 z_1 r_1 *option* **`newtroncconecreux`** *solid* ⟶ crée un nouveau tronc de cône creux, de type *solid*, d'axe *Oz*, de rayon de base r_0 (sur le plan $z = z_0$) et de rayon au sommet r_1 (sur le plan $z = z_1$). *option* indique le maillage ou le mode.

Mais aussi le vecteur, la calotte sphérique, et la calotte sphérique creuse :



r ϕ θ **`newcalottesphere`** *solid* ⟶ crée une nouvelle calotte sphérique, de rayon *r* et d'angles d'ouverture ϕ et θ. *option* indique le maillage ou le mode.

r ϕ θ **`newcalottespherecreuse`** *solid* ⟶ crée une nouvelle calotte sphérique creuse, de rayon *r* et d'angles d'ouverture ϕ et θ. *option* indique le maillage ou le mode.

x y z **`newvecteur`** *solid* ⟶ crée un nouveau vecteur, de coordonnées (x, y, z).

2.3 - Dessiner un solide

Pour dessiner un solide, on dispose des commandes **`drawsolid`** (dessin « fil de fer »), **`drawsolid*`** (avec coloriage des faces visibles) et **`drawsolid**`** (avec coloriage des faces visibles et algorithme du peintre), cette dernière – qui trace les facettes en commençant par la plus éloignée de l'observateur – n'étant utile que pour les solides non convexes.

solid **`drawsolid`** – ⟶ dessine le solide *solid*

solid **`drawsolid*`** – ⟶ dessine le solide *solid* avec coloriage des faces visibles

solid **`drawsolid**`** – ⟶ dessine le solide *solid* avec coloriage des faces visibles et en utilisant l'algorithme du peintre

On dispose de plus d'un booléen permettant d'activer ou non la représentation des arêtes cachées.

- *aretescachees* : booléen indiquant à la procédure **`drawsolid`** si l'on doit ou non représenter les arêtes cachées. **valeur par défaut :** *true*

2.4 - Définir la couleur des faces d'un solide

Par défaut, et comme à l'habitude, la couleur de coloriage des faces est définie par *fillstyle*. Pour une scène non éclairée (voir plus loin), plusieurs autres méthodes sont prévues pour spécifier la couleur d'une face donnée :

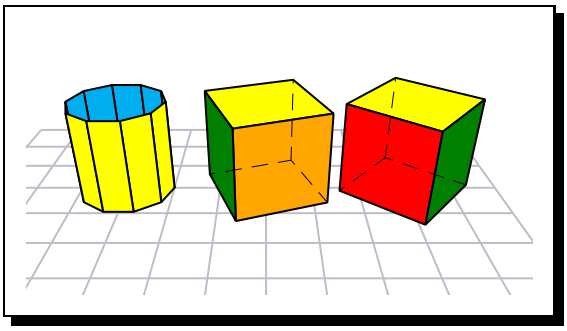
- la commande **`outputcolors`** permet de spécifier une couleur pour toutes les faces.
- la commande **`inoutputcolors`** permet de spécifier une couleur pour toutes les faces internes et une autre pour toutes les faces externes.
- la commande **`solidputfcolors`** permet de spécifier les couleurs de chacune des faces. On lui passe en argument un tableau de chaîne de caractères. La chaîne d'indice *i* indique la couleur de la face d'indice *i*.
- la commande **`solidputfcolor`** (noter le singulier) permet de spécifier la couleur d'une face donnée.

Pour un solide donné, on obtient le tableau des couleurs de faces avec la commandes **`solidgetfcolors`**. L'élement d'indice *i* du tableau *fcolor* correspond à la face d'indice *i* du solide.

Dans l'exemple ci-dessous, on dessine un cylindre creux, dont on précise que les faces internes doivent êtres colorées en cyan bleu et les faces externes en jaune. On définit ensuite un nouveau cube, que l'on stocke dans la variable *moncube*, dont on définit la couleur pour chacune des 6 faces. On représente ensuite 2 instances de ce cube, l'une ayant subi une rotation de 120° autour de l'axe *Oz*.

Dans cet exemple, on pourra remarquer un phénomène étonnant au premier abord : dans la variable *moncube* est stocké le cube de centre *O* et d'arête 2. On fait ensuite subir à ce solide une rotation de 20° autour de l'axe *Oz*, puis une translation de vecteur $\vec{k}$ (vecteur unitaire dirigeant l'axe *Oz*). La variable *moncube* désigne alors, non pas le cube d'origine, mais le cube transformé. Ce phénomène s'explique par le fait que pour les objets complexes (les solides, mais aussi les matrices, les tableaux, etc...), ce n'est pas l'objet que postscript dépose sur la pile, mais une référence à l'objet (un *pointeur* comme l'on dit en C, en Pascal ou en ADA). Or les transformations effectuées agissent sur l'objet lui-même...

Si l'on veut éviter ce phénomène, et si l'on souhaite avoir 2 solides identiques, mais avec des instances séparées, on utilisera la commande **`dupsolid`** qui construit une nouvelle instance, copie conforme de celle passée en argument.



source jps

```
-4.5 5 setxrange
-2 3 setyrange
20 setxunit
0 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin

0 1 2 newcylindrecreux
dup (cyan) (jaune) inoutputcolors
{3 0 0 translatepoint3d} solidtransform
drawsolid**
30 setfontsize
setTimes
/moncube
   2 newcube dup
   [(jaune) (orange) (bleu) (rouge)
      (vert) (0 0 0 1 setcymkcolor)]
   solidputfcolors
def
moncube
   {0 0 20 rotateOpoint3d} solidtransform
   {0 0 1 translatepoint3d} solidtransform
drawsolid*
moncube
   {0 0 120 rotateOpoint3d} solidtransform
   {-3 0 0 translatepoint3d} solidtransform
drawsolid*
```

En résumé :

solid str **`outputcolors`** $-\!\!\longrightarrow$ affecte la couleur définie par *str* à toutes les faces du solide *solid*

solid str_1 str_2 **`inoutputcolors`** $-\!\!\longrightarrow$ affecte la couleur définie par str_2 aux faces externes du solide *solid*, et la couleurs définie par str_1 aux faces internes

solid array **`solidputfcolors`** $-\!\!\longrightarrow$ affecte au solide *solid* le tableau *array* en tant que tableau des couleurs des faces

solid **`solidgetfcolors`** *array* $\longrightarrow$ le tableau *array* est le tableau des couleurs des faces pour le solide *solid*

2.5 - Dégradés de couleurs

Il est possible d'utiliser un dégradé de couleurs sur les faces d'un solide, que ce soit sur les faces internes, les faces externes, ou l'ensemble des faces. On utilise pour cela les commandes respectivess **`solidputinhuecolors`**, **`solidputhuecolors`** et **`solidputinouthuecolors`**.

solid array **`solidputhuecolors`** – ⟶ Affecte les couleurs des faces externes du solide *solid* selon le dégradé désigné par *array*

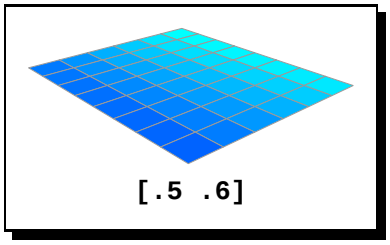
solid array **`solidputinhuecolors`** – ⟶ Affecte les couleurs des faces internes du solide *solid* selon le dégradé désigné par *array*

solid array **`solidputinouthuecolors`** – ⟶ Affecte les couleurs de l'ensemble des faces externes et internes du solide *solid* selon le dégradé désigné par *array*

Le type de dégradé est caractérisé par le tableau *array* passé en argument. Ce tableau contient 2, 4, 6, 7 ou 8 éléments.

2.5.1 - Dégradé dans l'espace HSB, saturation et brillance maximales

Il y a 2 arguments : **`[`**h_0 h_1**`]`** où les nombres h_0 et h_1 vérifiant $0 \leqslant h_0 < h_1 \leqslant 1$ indiquent les bornes du premier paramètre dans l'espace HSB.

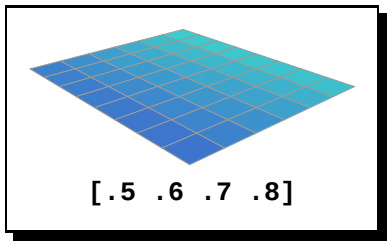


source jps

```
autocrop
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
6 6 3 SetCamPos
0 0 0 SetCamView
2 setlinejoin

setCourierBold
([.5 .6]) -1.2 -3 uctext

.1 setlinewidth
.6 setgray
0 3 -2 2 [.5 .5] newgrille
dup [.5 .6] solidputhuecolors
drawsolid*
```

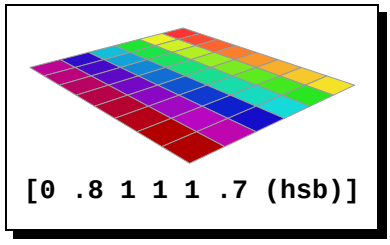
2.5.2 - Dégradé dans l'espace HSB, saturation et brillance fixes

Il y a 4 arguments : **`[`**h_0 h_1 s b**`]`** où les nombres h_0 et h_1 vérifiant $0 \leqslant h_0 < h_1 \leqslant 1$ indiquent les bornes du premier paramètre dans l'espace HSB et où s et b sont les paramètres respectifs *saturastion* et *brillance.*



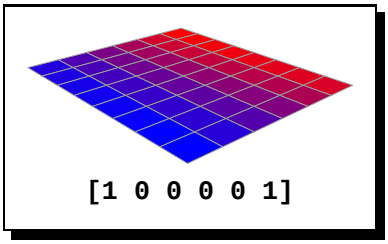
2.5.3 - Dégradé dans l'espace HSB, cas général

Il y a 7 arguments : **`[`**h_0 s_0 b_0 h_1 s_1 b_1 **`(hsb)]`** où les nombres h_i, s_i et b_i indiquent les bornes des paramètre HSB.



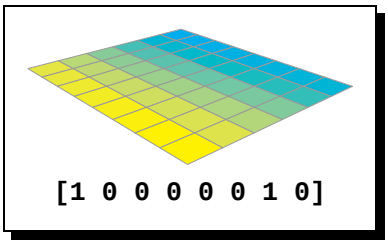
2.5.4 - Dégradé dans l'espace RGB

Il y a 6 arguments : **[** r_0 g_0 b_0 r_1 g_1 b_1 **]** où les nombres r_i, g_i et b_i indiquent les bornes respectives des 3 paramètres RGB.



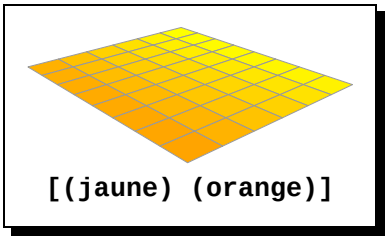
2.5.5 - Dégradé dans l'espace CMYK

Il y a 8 arguments : **[** c_0 m_0 y_0 k_0 c_1 m_1 y_1 k_1 **]** où les nombres c_i, m_i, y_i et k_i indiquent les bornes respectives des 4 paramètres CMYK.



2.5.6 - Dégradé entre 2 couleurs nommées

Il y a deux paramètres **[(***couleur1***) (***couleur2***)]** où *couleur1* et *couleur2* sont des noms de couleurs définies.

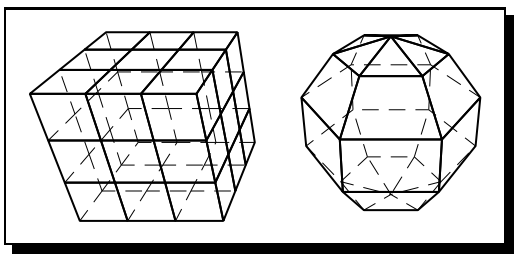


2.6 - Définition du maillage – Les modes

Pour la plupart des solides prédéfinis, on peut définir la maillage par le passage d'une option de type tableau.

Par exemple, le maillage de la sphère est défini par 2 paramètres. Un appel du type **`2 [4 6] newsphere`** définira une sphère de centre *O*, de rayon 2, avec un maillage de 4 × 6. Idem pour les cônes, cylindres, troncs de cône, tores, calottes sphériques, et les solides creux associés.

Pour certains solides, comme le cube ou les prismes, il n'y a qu'un seul paramètre pour le maillage. Par exemple **`2 [6] newcube`** définit un cube d'arête 2 de maille 6.



source jps

```
autocrop
-7 7 setxrange
-2.5 3 setyrange
20 setxunit
0 10 6 SetCamPos
0 0 0 SetCamView
2 setlinejoin

2 [4 6] newsphere
drawsolid

3 [3] newcube
{5 0 0 translatepoint3d} solidtransform
drawsolid
```

Il est également possible d'utiliser des maillages prédéfinis. On utilise pour cela la notion de *mode*. Il y a 5 modes distincts, numérotés de 0 à 4. Le mode 0 correspond au maillage prédéfini le plus grossier, et le 4 correspond au maillage prédéfini le plus fin.

Pour les solides permettant de définir le maillage, on peut passer en option un exécutable donnant le mode souhaité. Ainsi, **`2 {0} newsphere`** définit une sphère avec le maillage correspondant au mode 0. Lorsque l'on ne définit ni le maillage, ni le mode, le programme utilise le mode par défaut, qui est stocké dans la variable *defaultsolidmode*.

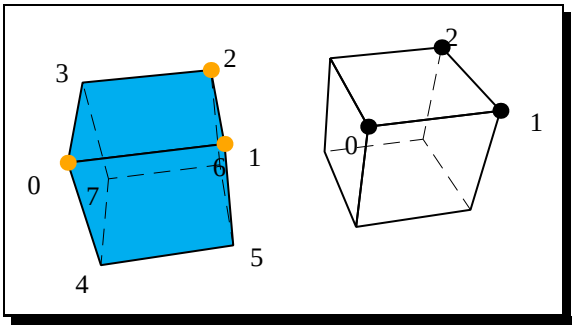
Ceci permet de préparer une scène en basse résolution (mode 0), et de ne calculer la résolution la plus fine, lourde en temps de calcul, qu'au dernier moment.

- *defaultsolidmode* : mode de résolution par défaut pour les solides. **valeur par défaut :** 2

2.7 - Numéroter les faces ou les sommets

Il est possible de pointer tout ou partie des sommets, et de numéroter tout ou partie des sommets. On utilise pour cela les commandes **`solidshowsommets`** et **`solidnumsommets`** :

> *solid array* **`solidnumsommets`** – ⟶ Si *array* est présent, numérote les sommets désignés dans le tableau d'indices *array*. Numérote tous les sommets sinon.
>
> *solid array* **`solidshowsommets`** – ⟶ Si *array* est présent, pointe les sommets désignés dans le tableau d'indices *array*. Pointe tous les sommets sinon.

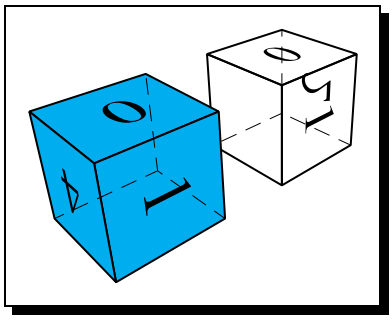


source jps

```
autocrop
-5 5 setxrange
-4 3 setyrange
20 setxunit
5 10 8 SetCamPos
3 0 0 SetCamView
2 setlinejoin

setTimes
3 newcube /A exch def

/dotscale {1.5 dup} def
A
dup [0 1 2] solidnumsommets
dup [0 1 2] solidshowsommets
drawsolid

A
{6 0 0 translatepoint3d} solidtransform
dup (cyan) outputcolors
dup drawsolid*
dup solidnumsommets
orange
[0 1 2] solidshowsommets
```

Il est possible de numéroter tout ou partie des faces, en tenant compte ou non de la visibilité de la face considérée. On utilise pour cela la commande **`solidnumfaces`** :

> *solid array bool* **`solidnumfaces`** – ⟶ Si *array* est présent, numérote les faces désignées dans le tableau d'indices *array*. Numérote toutes les faces sinon. Le booléen optionnel *bool* permet de définir si on doit ou non tenir compte de la visibilité de la face considérée.



source jps

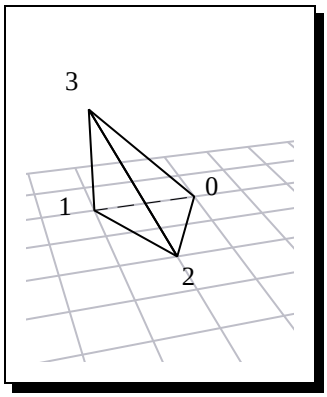
```
autocrop
-5 5 setxrange
-4 3 setyrange
20 setxunit
10 10 7 SetCamPos
0 0 0 SetCamView
2 setlinejoin

35 setfontsize
setTimes
3 newcube /A exch def

A
dup drawsolid
[0 1 2] false solidnumfaces

A
{5 0 0 translatepoint3d} solidtransform
dup (cyan) outputcolors
dup drawsolid*
solidnumfaces
```

2.8 - Construire un nouveau solide

2.8.1 - À partir du scratch

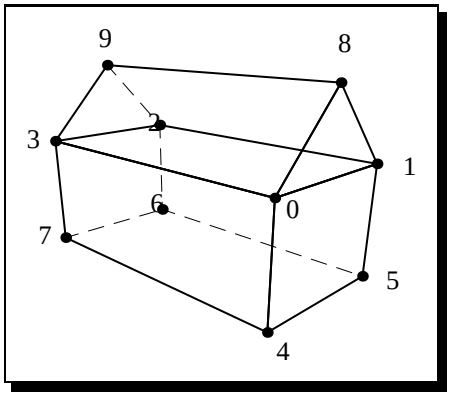
On crée un tableau des coordonnées des sommets, puis le tableau des faces, qui est lui-même un tableau de tableaux, ces derniers contenant, pour une face donnée, les indices des sommets de la face, rangés par ordre trigonométrique lorsque l'on regarde le solide de l'extérieur. On dépose alors les 2 tableaux (sommets et faces) sur la pile et on invoque la fonction **`generesolid`**.

> $array_1$ $array_2$ **`generesolid`** *solid* ⟶ construit un solide dont le tableaux des sommets est $array_1$ et le tableau des faces est $array_2$



source jps

```
-2 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
[
   -1 -1 0 %% sommet 0
    1 -1 0 %% sommet 1
    0  1 0 %% sommet 2
    1 -1 2 %% sommet 3
]
[
   [2 1 0]
   [0 1 3]
   [1 2 3]
   [2 0 3]
]
generesolid
dup drawsolid
solidnumsommets
```



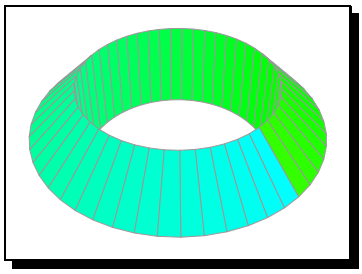
source jps

```
autocrop
-4.5 5 setxrange
-2 5 setyrange
20 setxunit
10 10 6 SetCamPos
0 0 0 SetCamView
2 setlinejoin

/S [
    2  4  3    -2  4  3
   -2 -4  3     2 -4  3
    2  4  0    -2  4  0
   -2 -4  0     2 -4  0
    0  4  5     0 -4  5
] def

/F [
   [0 1 2 3]    [7 6 5 4]
   [0 3 7 4]    [3 9 2]
   [1 8 0]      [8 9 3 0]
   [9 8 1 2]    [6 7 3 2]
   [2 1 5 6]    [0 4 5 1]
] def

S F generesolid
dup solidshowsommets
dup solidnumsommets
drawsolid
```

2.8.2 - Anneaux

La commande **newanneau** permet de construire des solides de révolution.

array n **newanneau** *solid* ⟶ crée un nouvel anneau, de type *solid*, dont la section est définie par *array*, un tableau de points (x_i, z_i). Le maillage est défini par *n*. *n* peut être un entier représentant le nombre de mailles, ou un exécutable représentant un mode..



source jps

```
autocrop
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
6 6 7 SetCamPos
0 0 0 SetCamView

2 setlinejoin

.1 setlinewidth
.6 setgray

[3 0 2 1 2 -1] 49 newanneau
dup [.3 .5] solidputhuecolors
drawsolid**
```

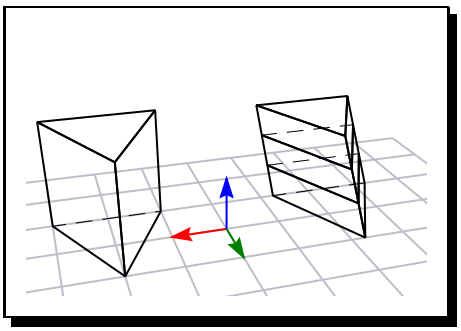
2.8.3 - Prismes

Pour construire un prisme, on construit d'abors un tableau des coordonnées 2d des sommets d'une face sur le plan *xOy*, puis on invoque l'une des fonctions **newprismedroit** ou **newprisme**.

array z_0 z_1 **newprismedroit** *solid* ⟶ construit un prisme droit d'axe *Oz* à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique ⇒ face de normale *Oz*, dans le sens des *z* croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$

array z_0 z_1 $\vec{u}$ **newprisme** *solid* ⟶ construit un prisme oblique d'axe $(O, \vec{u})$ à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique

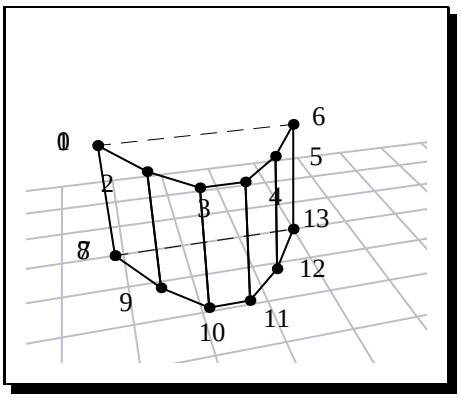
$\Rightarrow$ face de normale Oz, dans le sens des z croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$



source jps

```
-3 3 setxrange
-1 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
0 1 1 axesRVB
2 setlinejoin
/table [-1 -1 1 -1 0 1] def
table 0 2 newprismedroit
{2 0 0 translatepoint3d} solidtransform
drawsolid

table 0 2 .5 -.75 1 [3] newprisme
{-3 0 0 translatepoint3d} solidtransform
drawsolid
```

Remarque : Pour cette dernière méthode, le tableau décrivant la face initiale peut-être obtenue du chemin continu courant en utilisant la commande *currentcpathpointstable*. Par exemple, voici un prisme droit généré à partir de la courbe $y = 2\sin x$:

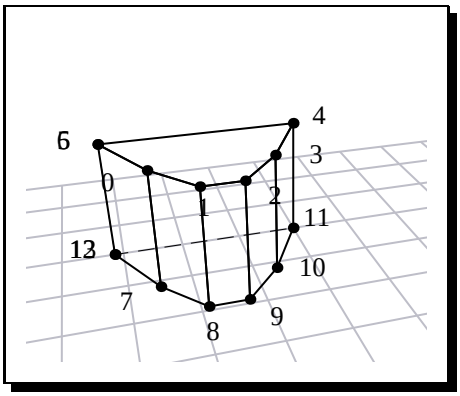


source jps

```
-4 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
6 setresolution
newpath
   pi 0 smoveto
   pi 0 {Sin 2 mul} Courbe_
currentcpathpointstable
0 2 newprismedroit
dup drawsolid
dup solidshowsommets
solidnumsommets
```

On remarque que sur l'exemple ci-dessus, la face supérieure du prisme n'est pas correctement rendue. C'est que si la face de base est donnée par $n + 1$ sommets $(A_0, A_1, \ldots, A_n)$ et si on note G le centre de gravité de ces $n + 1$ points, alors il faut avoir les angles $(\overrightarrow{GA_0}, \overrightarrow{GA_1})$ et $(\overrightarrow{GA_n}, \overrightarrow{GA_0})$ positifs pour avoir une orientation correcte.

L'astuce consiste alors à faire subir au tableau de base une rotation circulaire avec la commande **`rollparray`** :



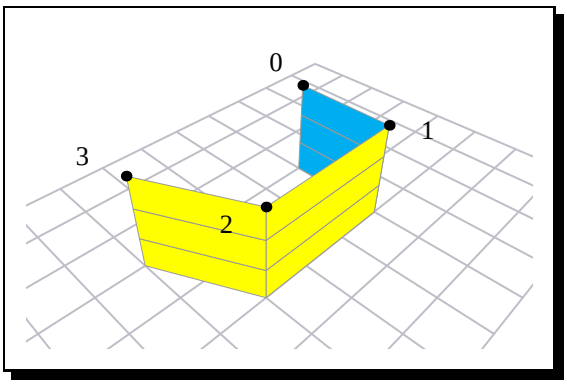
source jps

```
-4 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
6 setresolution
newpath
   pi 0 smoveto
   pi 0 {Sin 2 mul} Courbe_
currentcpathpointstable -2 rollparray
0 2 newprismedroit
dup drawsolid
dup solidshowsommets
solidnumsommets
```

2.8.4 - Rubans

La commande **newruban** permet de construire un solide ruban.

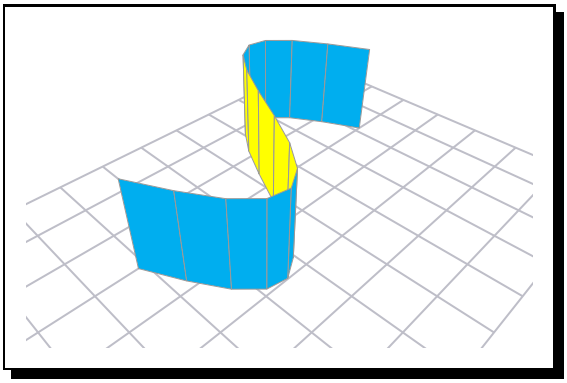
array z option **newruban** *solid* ⟶ *array* est un tableau de points en 2d indiquant les sommets du ruban sur le plan *xOy*, et *z* est un nombre indiquant la hauteur du ruban. L'argument optionnel *option* permet de spécifier un mode ou un maillage.



source jps

```
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
6 6 7 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin

.1 setlinewidth
.6 setgray

[-1 0 -1 2 2 2 3 0] 2 [3] newruban
dup (jaune) (cyan) inoutputcolors
dup drawsolid**
noir
dup [0 1 3 {} for] solidshowsommets
[0 1 3 {} for] solidnumsommets
```

Là encore, le tableau de points peut être issu d'un chemin :



source jps

```
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
6 6 7 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin

.1 setlinewidth
.6 setgray

16 setresolution
newpath
   pi 0 smoveto
   pi pi neg {Sin 2 mul} Courbe_
currentcpathpointstable
2 newruban
dup (jaune) (cyan) inoutputcolors
drawsolid**
```

2.8.5 - Solides plans

array **newmonoface** *solid* ⟶ construit un solide monoface, dans le plan *xOy*, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique ⇒ face de normale *Oz*, dans le sens des *z* croissants)

array **newbiface** *solid* ⟶ construit un solide biface, dans le plan *xOy*, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique ⇒ face de normale *Oz*, dans le sens des *z* croissants)

2.9 - Grilles, surfaces, surfaces paramétrées

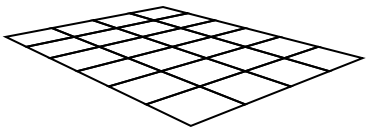
2.9.1 - La grille

Il existe un solide « grille ». La syntaxe est :

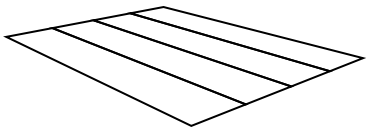
xmin xmax ymin ymax option **newgrille** *solid* ⟶ crée une nouvelle grille, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou un exécutable et représenter un mode.

L'option peut être un tableau définissant le maillage. Dans ce cas il est de la forme **[**n_1 n_2**]**, où les n_i sont des nombres. Si n_1 (resp. n_2) est entier, il définit le le nombre de mailles suivant l'axe *Ox* (resp. *Oy*), et s'il est réel, alors il définit le pas pour le maillage.

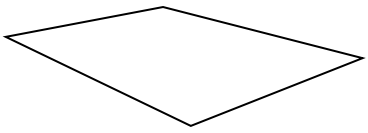
Par exemple, **[1 1]** définit une maille par axe, alors que **[1. 1.]** (noter le **.**) définit un pas d'une unité par maille sur chaque axe.



0 4 −3 3 newgrille



0 4 −3 3 [1. 1] newgrille



0 4 −3 3 [1 1] newgrille

L'option peut également être un exécutable indiquant le mode de maillage (autrement dit un entier entre 0 et 4). Par exemple, **0 4 -3 3 {0} newgrille** et **0 4 -3 3 [1 1] newgrille** sont équivalents.

Si aucune option n'est présente, la grille est construite avec le mode par défaut.

2.9.2 - Surfaces

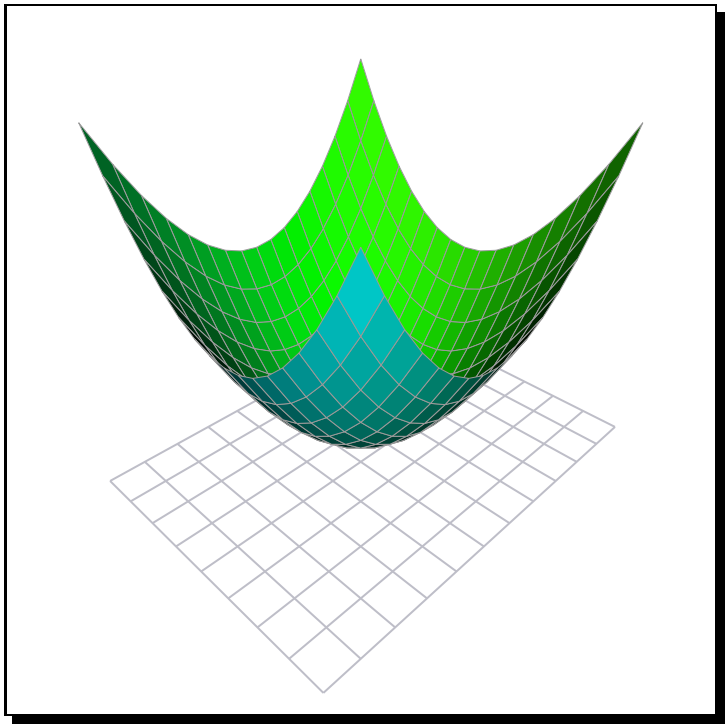
On peut représenter des surfaces ayant une équation du type

$$z = f(x, y)$$

avec la commande **newsurface**. La syntaxe est parfaitement analogue à celle de **newgrille**.

xmin xmax ymin ymax option {*f*} **newsurface** *solid* ⟶ crée une nouvelle surface, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou

un exécutable et représenter un mode. L'exécutable f est une fonction de $[xmin, xmax] \times [ymin, ymax]$ dans $\mathbb{R}$.



source jps

```
-5 5 setxrange
-4 6 setyrange
25 setxunit
10 10 12 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
10 setresolution

10 10 10 setlightsrc
.1 setlinewidth
.6 setgray
/f {
2 dict begin
   /y exch def
   /x exch def
#rpn# (x^2 + y^2)/4
end
} def

-4 4 -4 4 [.5 .5] {f} newsurface
dup [.3 .5] solidputinouthuecolors
drawsolid**
```

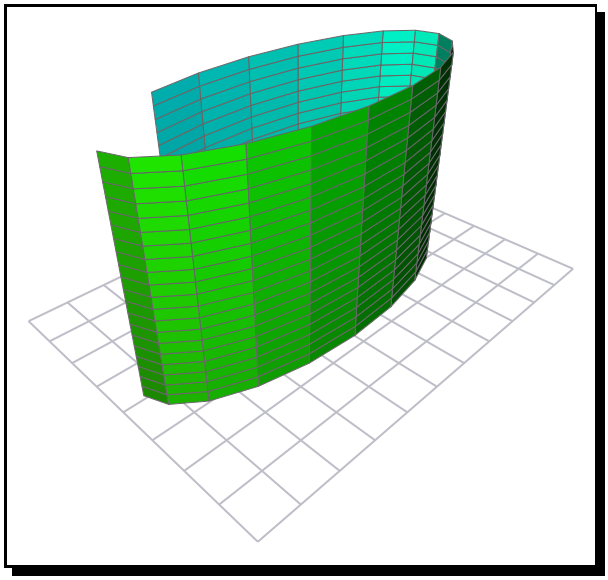
2.9.3 - Surfaces paramétrées

On peut représenter des surfaces paramétrées ayant une équation du type

$$(x, y, z) = f(u, v)$$

avec la commande **`newsurfaceparametree`**. La syntaxe est parfaitement analogue à celle de **`newgrille`** et **`newsurface`**.

xmin xmax ymin ymax option $\{f\}$ **`newsurfaceparametree`** *solid* $\longrightarrow$ crée une nouvelle surface paramétrée, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou un exécutable et représenter un mode. L'exécutable f est une fonction de $[xmin, xmax] \times [ymin, ymax]$ dans $\mathbb{R}^3$.



source jps

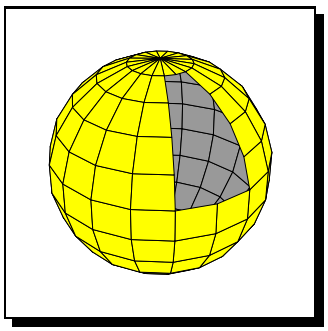
```
autocrop
-6 6 setxrange
-6 6 setyrange
25 setxunit
10 10 10 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
10 10 10 setlightsrc

/f {
2 dict begin
   /v exch def
   /u exch def
#rpn# 5 * Cos (u)
#rpn# 2 * Sin (u)
   v
end
} def

.2 setlinewidth
.4 setgray
   0 pi 2 mul 1 sub
   -1 5
   [.3 .3]
{f} newsurfaceparametree
dup [.3 .5] solidputinouthuecolors
drawsolid**
```

2.10 - Évider un solide – Enlever des faces

La commande **`videsolid`** ajoute à un solide ses faces intérieures.

On dispose également des commandes **`solidrmface`** et **`solidrmfaces`** qui permettent d'enlever respectivement une ou plusieurs faces d'un solide.

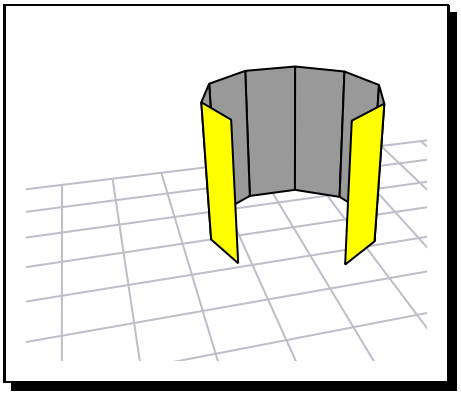
Par exemple, voici une ouverture dans une sphère :



source jps

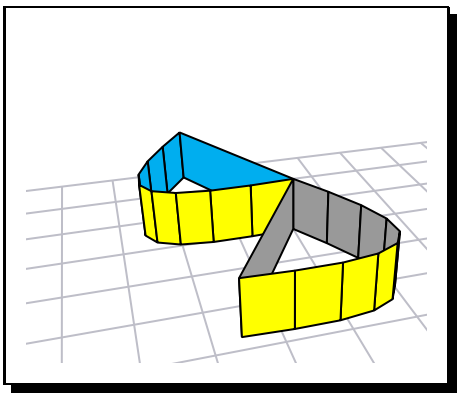
```
-2 2 setxrange
-2 2 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
2 setlinejoin
.1 setlinewidth

2 [10 18] newsphere
dup [40 41 58 59 76 77 94 95] solidrmfaces
dup videsolid
dup (.6 setgray) (jaune) inoutputcolors
drawsolid**
```

La commande **`creusesolid`** fonctionne comme **`videsolid`**, mais elle procède également à la suppression des faces d'indice 0 et 1 du solide considéré. Elle peut être utilisée sur les cylindres, les troncs de cône ou les prismes.



source jps

```
-4 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
0 1.5 2.5 newcylindre
dup [3 4] solidrmfaces
dup creusesolid
dup (.6 setgray) (jaune) inoutputcolors
drawsolid**
```



source jps

```
-4 2 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
10 setresolution
newpath
   pi 0 smoveto
   pi 0 {Sin 2 mul} Courbe_
currentcpathpointstable 1 0 1 newprismedroit
dup creusesolid
dup (cyan) (jaune) inoutputcolors
dupsolid {0 0 -60 rotateOpoint3d} solidtransform
drawsolid**
dup (.6 setgray) (jaune) inoutputcolors
dupsolid {0 0 60 rotateOpoint3d} solidtransform
drawsolid**
```

En résumé :

solid **videsolid** – ⟶ ajoute au solide *solid* ses faces internes

solid **creusesolid** – ⟶ enlève les faces d'indice 0 et 1 du solide *solid*, puis ajoute toutes les faces internes

solid i **solidrmface** – ⟶ enlève la face d'indice *i* du solide *solid*

solid array **solidrmfaces** – ⟶ enlève du solide *solid* toutes les faces définies par le tableau d'indices *array*

2.11 - Opérations sur les solides

La commande **solidtransform** à qui l'on passe en argument une transformation $\mathbb{R}^3 \rightarrow \mathbb{R}^3$, applique cette tranformation à chacun des sommets du solide considéré. Ceci permet notamment, par le biais des commandes **translatepoint3d** et **rotate0point3d**, de positionner un solide dans l'espace, mais aussi, par l'intermédiaire de **scaleOpoint3d**, de changer ses dimensions.

solid $\{f\}$ **solidtransform** *solid* ⟶ applique la transformation f au solide *solid*, f étant une application $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$

Pour les cubes, on dispose de la commande **tronque_cube**, qui permet d'obtenir un nouveau solide en coupant les 8 coins du cube d'origine suivant une proportion que l'on transmet en argument (2 ⇒ on coupe au milieu de l'arête, 3 ⇒ au tiers, 4 ⇒ au quart, etc...)

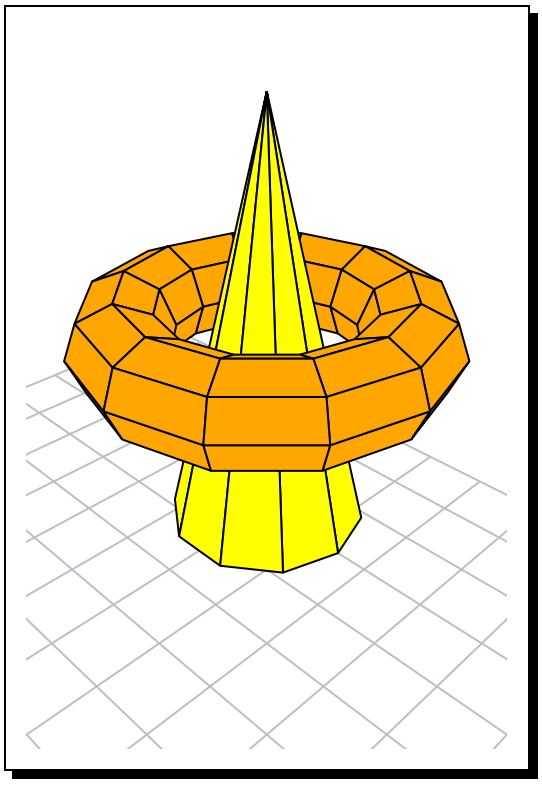
$solid_1$ n **tronque_cube** $solid_2$ ⟶ $solid_2$ est le solide obtenu en coupant chaque coin du parallélépipède $solid_1$. n est un réel supérieur à 2 indiquant la proportion à respecter pour la tronquature

solid $string_0$ $string_1$ **inoutputcolors** – ⟶ affecte la couleur définie par $string_0$ aux faces internes du solide *solid*, et la couleur définie par $string_1$ aux faces externes

solid string **outputcolors** – ⟶ affecte la couleur définie par *string* à toutes les faces du solide *solid*

2.12 - Fusionner 2 solides

Il est également possible de « fusionner » deux solides pour n'en obtenir qu'un. Ceci permet d'appliquer l'algorithme du peintre sur l'ensemble. Par exemple :



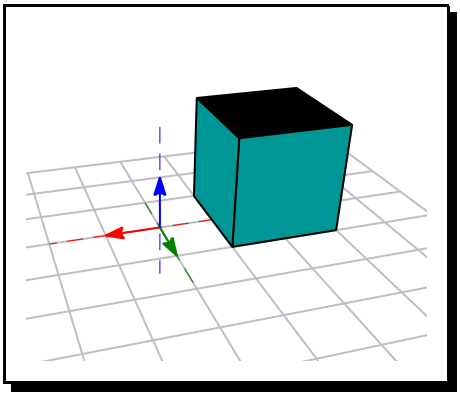
source jps

```
-3 3 setxrange
-3 6 setyrange
30 setxunit
-7 7 8 SetCamPos
0 0 0 SetCamView
2 setlinejoin
qplanxy

0 1.5 6 newcone
dup (jaune) outputcolors

.7 2 newtore
{0 0 3 translatepoint3d} solidtransform
dup (orange) outputcolors

solidfuz
drawsolid**
```

solid$_1$ *solid*$_2$ **`solidfuz`** *solid* $\longrightarrow$ dépose sur la pile le solide obtenu à partir de la fusion des solides *solid*$_1$ et *solid*$_2$

2.13 - Éclairage par une source lumineuse ponctuelle

Il est possible d'éclairer la scène par une source ponctuelle d'une couleur et d'une intensité donnée. Dans ce cas, seule cette dernière est prise en compte pour le coloriage des faces des solides présents. L'intensité de la couleur d'une face dépend de l'inclinaison de la face et de sa distance par rapport à la source lumineuse.



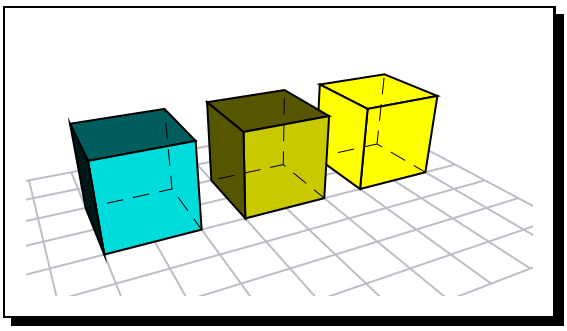
source jps

```
-2 4 setxrange
-2 3 setyrange
25 setxunit
3 10 6 SetCamPos
0 0 0 SetCamView

2 setlinejoin
qplanxy
-1 2 1 axesRVB

{cyan} setlight
2 4 2 setlightsrc

/aretescachees false def
2 newcube
{-2 0 1 translatepoint3d} solidtransform
drawsolid*
```

La source lumineuse est définie par sa position *lightsrc*, sa couleur *lightcolor* et son intensité *lightintensity*. On modifie ces paramètres en utilisant les commandes **`setlightsrc`**, **`setlight`** et **`setlightintensity`**. Seule *lightintensity* est définie par défaut. Si *lightsrc* n'est pas définie, alors le solide est colorié avec des teintes uniformes. Si *lightsrc* est défini mais pas *lightcolor*, alors on considère que la lumière est blanche, et l'intensité de la teinte de chaque facette dépend de l'orientation par rapport à *lightsrc*. Enfin, si en plus la variable *lightcolor* est définie, alors

les solides sont considérés blanc et la teinte de chaque facette dépend de l'orientation de la source lumineuse et de sa couleur.

Par exemple, voici un cube jaune avec 3 modes d'éclairage distincts : (1) pas de source lumineuse ponctuelle, (2) source ponctuelle blanche, (3) source ponctuelle cyan :



source jps

```
-4.5 5 setxrange
-2 3 setyrange
20 setxunit
5 10 6 SetCamPos 0 0 0 SetCamView
qplanxy
2 newcube           %% un nouveau cube
dup (jaune) outputcolors
dupsolid
{-3 0 1 translatepoint3d} solidtransform
drawsolid*
5 10 6 setlightsrc  %% avec lumiere blanche
dupsolid
{0 0 1 translatepoint3d} solidtransform
drawsolid*
{cyan} setlight     %% avec lumiere cyan
{3 0 1 translatepoint3d} solidtransform
drawsolid*
```

array **setlight** – ⟶ Affecte la couleur de la source lumineuse ponctuelle dans l'espace RGB ou CYMK, *array* étant un tableau de 3 ou 4 réels de l'intervalle [0; 1]

{*f*} **setlight** – ⟶ Éxécute la fonction *f* dans un **gsave .. grestore**, y récupère la définition de la couleur dans l'espace RGB, puis l'affecte à la couleur de la source lumineuse ponctuelle.

i **setlightintensity** – ⟶ Affecte l'intensité de la source lumineuse ponctuelle

x y z **setlightsrc** – ⟶ Affecte la position de la source lumineuse ponctuelle

- *lightintensity* : Intensité de la source lumineuse ponctuelle. **valeur par défaut :** 1

2.14 - Boîte à outils

Ci-dessous une liste des autres fonctions et procédures disponibles concernant le type *solid* :

– **newsolid** *solid* ⟶ dépose le solide nul sur la pile

any **issolid** *bool* ⟶ *bool* vaut *true* si *any* est de type *solid*, *false* sinon

solid i **solidgetsommetsface** *array* ⟶ *array* est le tableau des sommets de la face d'indice *i* du solide *solid*

solid i j **solidgetsommetface** *S* ⟶ *S* est le sommet d'indice *i* de la face d'indice *j* du solide *solid*

solid i **solidgetsommet** *S* ⟶ *S* est le sommet d'indice *i* du solide *solid*

solid **solidgetpointstable** *array* ⟶ *array* est le tableau des sommets du solide

solid **solidgetfaces** *array* ⟶ *array* est le tableau des faces du solide

solid i **solidgetface** *array* ⟶ *array* est le tableau décrivant la face *i* du solide (tableau d'indices de sommets)

solid array **solidputpointstable** – ⟶ Remplace le tableau des sommets du solide *solid* par le tableau *array*

solid array **solidputfaces** – ⟶ Remplace le tableau des faces du solide *solid* par le tableau *array*

solid i **solidfacevisible?** *bool* ⟶ *bool* vaut *true* si la face d'indice *i* du solide *solid* est visible

solid i **solidnormaleface** $\vec{u}$ ⟶ $\vec{u}$ est un vecteur normal à la face d'indice *i* du solide *solid*. Le vecteur est orientaté vers l'extérieur du solide

solid i **solidcentreface** *G* ⟶ le point *G* est le centre de la face d'indice *i* du solide *solid*

solid **solidnombresommets** *n* ⟶ nombres de sommets du solide *solid*

solid **solidnombrefaces** *n* ⟶ nombres de faces du solide *solid*

solid **dupsolid** *solid solid* ⟶ crée une nouvelle instance du solide déposé sur la pile

3. Projections

3.1 - Projeté d'un chemin

Il s'agit de créer un dessin sur le plan *xOy* et de le projeter sur un plan quelconque de la scène 3d représentée.

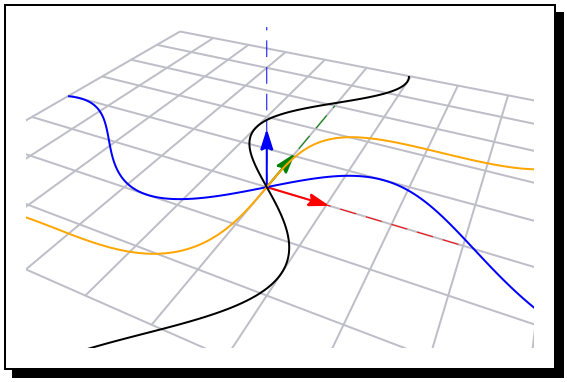
On commence par créer un chemin sur le plan *xOy*, puis on le projette avec la commande **`projpath`**, puis on l'encre (avec la commande **`stroke`**).

L'espace euclidien 3d étant rapporté à une base orthonormale $(O, \vec{\imath}, \vec{\jmath}, \vec{k})$, il nous suffit, pour définir un plan de projection, de connaître l'image de O, de $\vec{\imath}$ et $\vec{k}$ par cette projection.

Pour alléger la syntaxe, on peut utiliser seulement l'image de l'origine et du vecteur $\vec{k}$. Une orientation du plan de projection est alors proposée, et l'utilisiteur peut alors modifier cette orientation : soit en indiquant l'angle d'une rotation du repère proposée autour de la normale du plan de projection (portée par l'image de $\vec{k}$), soit en indiquant également l'image du vecteur $\vec{\imath}$.

Plus précisément, si (x_0, y_0, z_0) est l'image de O, et si (i_1, i_2, i_3) et (k_1, k_2, k_3) sont les images repctives des vecteut $\vec{\imath}$ et $\vec{k}$, alors la commande **`projpath`** admet les syntaxes suivantes :

> $x_0\ y_0\ z_0$ **[** $k_1\ k_2\ k_3$ **]** *bool* **`projpath`** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.
>
> $x_0\ y_0\ z_0$ **[** $k_1\ k_2\ k_3\ \alpha$ **]** *bool* **`projpath`** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$, après avoir fait subir une rotation d'angle α degrés autour de la normale par rapport à l'orientation originellement proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.
>
> $x_0\ y_0\ z_0$ **[** $i_1\ i_2\ i_3\ k_1\ k_2\ k_3$ **]** *bool* **`projpath`** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est donnée par $\vec{\imath}'(i_1, i_2, i_3)$, image du vecteur $(1, 0, 0)$ ar cette projection. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.



source jps

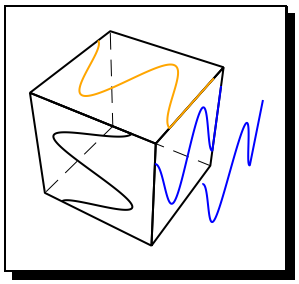
```
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
3 -6 5 SetCamPos
0 0 0 SetCamView
qplanxy
2 setlinejoin
0 3 1 axesRVB

%% on fait le chemin
newpath
-5 dup Sin smoveto
-5 5 {Sin} Courbe_

%% projete, avec vecteur normal
gsave
   bleu
   0 0 0 [0 0 1] projpath
stroke
grestore
%% projete, avec vecteur normal + rotation
gsave
orange
   0 0 0 [0 0 1 45] projpath
stroke
grestore
%% projete, avec images de I et K
0 0 0 [0 1 0 0 0 1] projpath
stroke
```

On peut également spécifier le plan de projection par rapport à l'une des faces d'un solide déjà construit.

Par défaut, il suffit de spécifier la face du solide. Le programme choisit alors le centre de la face comme image de l'origine, et propose une orientation du plan de projection. On peut spécifier un angle de rotation autour de la normale si l'orientation proposée ne convient pas. On peut également changer l'image proposée pour l'origine.

> *solid n* α *str bool* **`projpath`** - ⟶ Projette le chemin courant sur le plan affine défini par la face d'indice *n* du solide *solid*. Le booléen optionnel *bool* sert à préciser si on doit ou non tenir compte de la visibilité (*true* par défaut). La chaîne de caractères optionnelle *str* permet de préciser l'origine du plan de projection

(centre de la face de projection par défaut). L'angle optionnel α permet de préciser la rotation autour de la normale souhaitée.



source jps

```
autocrop
-4.5 5 setxrange
-3 3 setyrange
20 setxunit
5 -8 8 SetCamPos
0 0 0 SetCamView
2 setlinejoin

%% creation du cube
3 newcube /A exch def
A drawsolid

%% creation du chemin et des projetes
newpath
   -1.5 dup Sin smoveto
   -1.5 1.5 {3 mul Sin} Courbe_
   gsave
      orange A 0 projpath
      stroke
   grestore
   gsave
      bleu A 4 projpath
      stroke
   grestore
   gsave
      bleu A 4 (2.5 0 0) projpath
      stroke
   grestore
   A 3 projpath
stroke
```

3.2 - Projeté de texte

De façon analogue au projeté de chemin, il est possible de projeter un chemin défini par une chaîne de caractères. Là encore il faut définir l'origine du plan de projection, sa normale, et son orientation.

Lorsque tout cela est défini, la commande **`cctextp3d`** va projeter et encrer le chemin défini par la chaîne de caractère transmise en argument. Le positionnement sera tel que le point **`cc`** de la chaîne de caractères sera exactement positionné en l'origine du plan de projection. Plus précisément, le chemin sera crée par la commande **`0 0 cctext`** avant d'être projeté sur le plan défini de la scéne 3d.

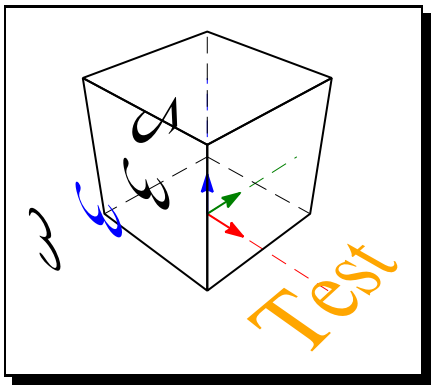
Cette commande **`cctextp3d`** supporte donc plusieurs syntaxes :

str x_0 y_0 z_0 **[** k_1 k_2 k_3 **]** *bool* **`cctextp3d`** - ⟶ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

str x_0 y_0 z_0 **[** k_1 k_2 k_3 α **]** *bool* **`cctextp3d`** - ⟶ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$, après avoir fait subir une rotation d'angle α degrés autour de la normale par rapport à l'orientation originellement proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

str x_0 y_0 z_0 **[** i_1 i_2 i_3 k_1 k_2 k_3 **]** *bool* **`cctextp3d`** - ⟶ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est donnée par $\vec{i}'(i_1, i_2, i_3)$, image du vecteur $(1, 0, 0)$ ar cette projection. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

str solid n α *str bool* **`cctextp3d`** - ⟶ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine défini par la face d'indice *n* du solide *solid*. Le booléen optionnel *bool* sert à préciser si on doit ou non tenir compte de la visibilité (*true* par défaut). La chaîne de caractères optionnelle *str* permet de préciser l'origine du plan de projection (centre de la face de projection par défaut). L'angle optionnel α permet de préciser la rotation autour de la normale souhaitée.



source jps

```
autocrop
-4.5 5 setxrange
-3 5 setyrange
20 setxunit
6 -6 7 SetCamPos
0 0 0 SetCamView
2 setlinejoin

35 setfontsize
setTimes

0 3 1 axesRVB

%% projete sur un plan
orange
(Test)
3 0 0 [0 0 1 90] cctextp3d

%% projete sur un solide
noir
3 newcube /A exch def
A {0 0 1.5 translatepoint3d} solidtransform
A drawsolid
%solidnumfaces
(2) A 2 false cctextp3d
(3) A 3 cctextp3d
bleu
(3) A 3 (0 -2.5 1.5) cctextp3d
noir
(3) A 3 90 (0 -3.5 1.5) cctextp3d
```

Comme à l'accoutumée, il y a 16 déclinaisons de la commande **`cctextp3d`**, qui ne diffèrent que par leurs 16 préfixes : **`ul`**, **`cl`**, **`bl`**, **`dl`**, **`uc`**, **`cc`**, **`bc`**, **`dc`**, **`ub`**, **`cb`**, **`bb`**, **`db`**, **`ur`**, **`cr`**, **`br`**, **`dr`**.

VI - Nœuds, arbres

1. Gestion des nœuds et de leurs connexions

La partie exposée dans ce paragraphe est une tentative de transcription conforme (du point de vue utilisateur) de la gestion par PSTricks des « Nodes » et de leurs connexions. Ces macros, qui utilisent directement l'environnement 'picture', permettent d'afficher divers labels ou étiquettes et les relier entre eux.

1.1 - Les nœuds

Lorsque l'on déclare un nœud, on déclare en même temps son nom (sous forme d'une chaîne alphanumérique) et son type (circulaire, circulaire de rayon fixe, rectangulaire, rectangulaire encadré avec tracé du cadre et rectangulaire encadré sans tracé du cadre). Après cette déclaration, le prochain objet affiché par l'environnement 'picture' sera considéré comme un nœud et associé au nom déclaré.

Plus précisément, on dispose de sept commandes **node**, **bnode**, **Rnode**, **cnode**, **Cnode**, **dianode** et **ovalnode** ayant la syntaxe suivante :

string **node** – ⟶ Déclare un nœud rectangulaire dont le nom est défini par *string*

string **bnode** – ⟶ Déclare un nœud rectangulaire encadré dont le nom est défini par *string*

string **Rnode** – ⟶ Déclare un nœud rectangulaire avec un encadrement non dessiné et dont le nom est défini par *string*

string **cnode** – ⟶ Déclare un nœud circulaire dont le nom est défini par *string*

string **Cnode** – ⟶ Déclare un nœud circulaire de rayon fixe *Circleradius* dont le nom est défini par *string*

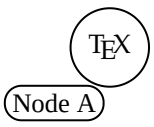
string **dianode** – ⟶ Déclare un nœud losange (diamond) dont le nom est défini par *string*

string **ovalnode** – ⟶ Déclare un nœud ovale (elliptique) dont le nom est défini par *string*

On dispose également de la commande **enode**, avec une syntaxe légèrement différente, qui permet de créer un nœud vide en un point donné :

x y string **enode** – ⟶ Déclare un nœud vide (*emptynode*) au point de coordonées (x, y) et dont le nom est défini par *string*

Voici un exemple où l'on déclare un nœud de chacun des deux types **bnode** et **cnode** :



source

```
(A) bnode           %% declare un node rectangulaire encadre (A)
3 setdboxit         %% reglage bordure de l'encadrement
/linearc .3 def     %% pour arrondir les angles
setTimes
(Node A) 0 0 cctext
circleit            %% encerclement du prochain label
(B) cnode           %% declaration d'un node circulaire (B)
#tex# \TeX
1 1 cctexlabel
```

1.2 - Les connexions de nœuds

Toutes les commandes de connexion de nœuds commencent par le préfixe **nc**, et elles ont toutes la même syntaxe :

$$string_1 \quad string_2 \quad option \quad \texttt{ncline}$$

où les chaînes $string_1$ et $string_2$ définissent les noms des nœuds à relier, et où *option* est une chaîne de caractère optionnelle définissant le type des terminaisons de ligne (du type **<->** ou ***-]**, etc. . .).

Ci-dessous, lorsque l'on fait référence aux nœuds *A* et *B*, on fait seulement référence aux nœds dans l'ordre dans lequel leurs noms ont été donnés comme argument aux commandes de tracés de connexions.

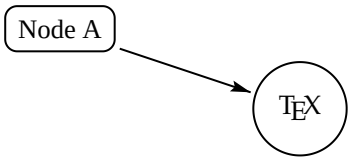
Ces commandes de tracés utilisent les paramètres suivants :

- *nodesep* : espace en picas séparant le point de connexion et l'extrémité du trait de connexion. **valeur par défaut :** 3
- *armA* : certaines connexions commencent avec un bras de longueur *armA* (en picas). **valeur par défaut :** 10
- *armB* : certaines connexions commencent avec un bras de longueur *armB* (en picas). **valeur par défaut :** 10

- *angleA* : certaines connexions vous permettent de spécifier avec quel angle vous souhaitez la connexion au nœud. **valeur par défaut :** 0
- *angleB* : certaines connexions vous permettent de spécifier avec quel angle vous souhaitez la connexion au nœud. **valeur par défaut :** 0
- *arcangleA* : ce paramètre ne sert qu'avec les commandes **`ncarc`** et **`pcarc`**. **valeur par défaut :** 10
- *arcangleB* : ce paramètre ne sert qu'avec les commandes **`ncarc`** et **`pcarc`**. **valeur par défaut :** 10

Pour les connexions proprement dites, on dispose de 8 commandes :

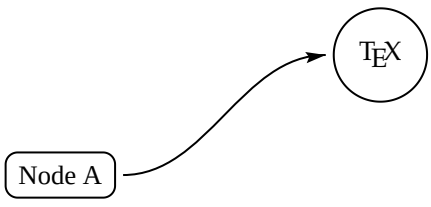
string$_1$ *string*$_2$ *option* **`ncline`** $--\longrightarrow$ Trace une simple ligne entre les nodes *A* et *B*. Seul le paramètre *nodesep* est utilisé.



ncline

```
5 setdboxit
setTimes
/linearc .3 store   %% pour arrondir les angles
boxit               %% encadrement du prochain label

(A) node
(Node A) -3 3 cctext
circleit
(B) cnode
#tex# \TeX
3 1 cctexlabel

(A) (B) (->) ncline
```

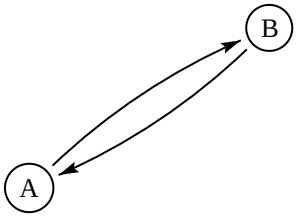
string$_1$ *string*$_2$ *option* **`nccurve`** $--\longrightarrow$ Trace une courbe de Bézier entre les nodes *A* et *B*. Les paramètres *angleA* et *angleB* sont utilisés.



nccurve

```
/angleB 180 def
5 setdboxit
setTimes
/linearc .3 store
boxit
(A) node
(Node A) -5 -1 cctext
circleit
(B) cnode
#tex# \TeX
3 2 cctexlabel

%% on ajuste la rotation de la fleche
/arrowangle 7.5 def
(A) (B) (->) nccurve
```

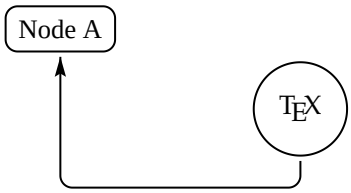
string$_1$ *string*$_2$ *option* **`ncarc`** $--\longrightarrow$ Connecte les nodes avec une courbe de Bézier, en utilisant le paramètre *nodesep*. La courbe se connecte en *A* avec un angle *arcangleA* par rapport à la droite (*AB*) et se connecte en *B* avec un angle $-$*arcangleB* par rapport à la droite (*AB*).



ncarc

```
setTimes
circleit
(A) cnode
(A) -3 -2 cctext
circleit
(B) cnode
(B) 3 2 cctext

(B) (A) (->) ncarc
(A) (B) (->) ncarc
```

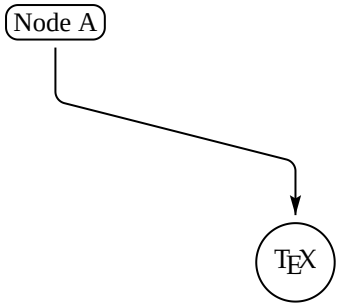
string$_1$ *string*$_2$ *option* **ncbar** – ⟶ Trace d'abord les bras de longueurs respectives *armA* et *armB* à un angle *angleA*. Ensuite, l'un des bras est étendu puis connecté, de telle façon que la ligne finale soit composée de 3 segments à angle droit.



ncbar

```
/angleA -90 def
5 setdboxit
setTimes
/linearc .3 store
boxit

(A) node
(Node A) -3 3 cctext
circleit
(B) cnode
#tex# \TeX
3 1 cctexlabel

(B) (A) (->) ncbar
```

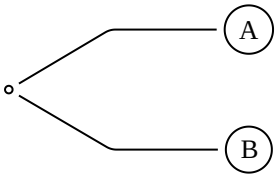
string$_1$ *string*$_2$ *option* **ncdiag** – ⟶ Trace d'abord les bras de longueurs respectives *armA* et *armB* à des angles respectifs *angleA* et *angleB*. Ensuite, ces bras sont connecté par une ligne droite. Le paramètre *linearc* est utilisé pour arrondir les angles.



ncdiag

```
/armA 20 def        /angleA -90 def
/armB 20 def        /angleB 90 def
setTimes
/linearc .3 store
boxit
(A) node
(Node A) -3 3 cctext
circleit
(B) cnode
#tex# \TeX
3 -3 cctexlabel

(A) (B) (->) ncdiag
```

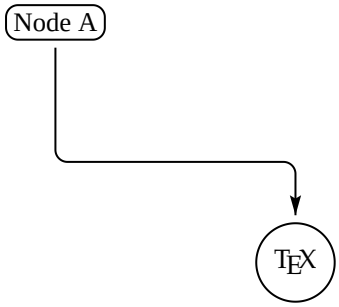
string$_1$ *string*$_2$ *option* **ncdiagg** – ⟶ Trace d'abord, en *A* et à l'angle *angleA*, le bras de longueur *armA*. Ensuite, ce bras est directement connecté au point *B*. Le paramètre *linearc* est utilisé pour arrondir les angles.



ncdiagg

```
/armA 40 def
/angleA 180 def
/linearc .5 def
/A {3 3} def /B {3 0} def /C {-3 1.5} def

setTimes
circleit
(A) cnode
(A) A cctext
circleit
(B) cnode
(B) B cctext

1 setdboxit
circleit
(C) cnode
() C cctext         %% node vide

(A) (C) (-) ncdiagg
(B) (C) (-) ncdiagg
```

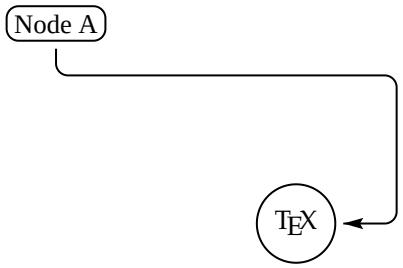
string$_1$ *string*$_2$ *option* **ncangle** – ⟶ Trace en *B*, et suivant l'angle *angleB* un bras de longueur *armB*, puis elle connecte ce bras en *A*, suivant l'angle *angleA* par un double segment à angle droit.



ncangle

```
/armB 20 def
/angleA -90 def
/angleB 90 def
/linearc .5 def
setTimes
/linearc .3 store
boxit

(A) node
(Node A) -3 3 cctext
circleit
(B) cnode
#tex# \TeX
3 -3 cctexlabel

(A) (B) (->) ncangle
```

string$_1$ *string*$_2$ *option* **ncangles** – ⟶ Trace en *A* (resp. *B*), et suivant l'angle *angleA* (resp. *angleB*) un bras de longueur *armA* (resp. *armB*), puis elle connecte les 2 bras par un double segment à angle droit (en partant de *B*).



ncangles

```
/armB 20 def
/angleA -90 def

setTimes
/linearc .3 store
boxit

(A) node
(Node A) -3 3 cctext
circleit
(B) cnode
#tex# \TeX
3 -2 cctexlabel

(A) (B) (->) ncangles
```

1.3 - Les connexions de points

Pour connexter des points entre eux, on dispose de 8 commandes analogues aux commandes de connections de nodes :

A B option **pcline** – ⟶ Comme **ncline**
A B option **pccurve** – ⟶ Comme **nccurve**
A B option **pcarc** – ⟶ Comme **ncarc**
A B option **pcbar** – ⟶ Comme **ncbar**
A B option **pcdiag** – ⟶ Comme **ncdiag**
A B option **pcdiagg** – ⟶ Comme **ncdiagg**
A B option **pcangle** – ⟶ Comme **ncangle**
A B option **pcangles** – ⟶ Comme **ncangles**

1.4 - Pour aller plus loin

Chaque fois qu'un nœud est déclaré, un dictionnaire spécifique lui est associé. Ce dictionnaire contient un certain nombre de données spécifiques au nœud considéré. En particulier, chaque dictionnaire contient l'ensemble des points définis par l'environnement 'picture' au moment de l'affichage (et donc au minimum les 16 points de référence *ul*, *ub*, etc. . .).

Pour les nœuds de type circulaire, le dictionnaire associé contient également l'entrée *noderayon*.

La commande **nodepoint** est prévue pour accéder à ces données. Sa syntaxe est :

string <*option*> **nodepoint**

où *string* désigne le nom du nœud considéré et où *option* peut soit être absente, soit être un nombre, soit être un littéral.

Sans l'option, **nodepoint** renvoie le point central du nœud (associé à l'entrée *cc*).

Avec une option numérique, **nodepoint** considère ce nombre comme un angle α, et renvoie le point de la bordure du nœud correspondant à l'angle α par rapport au centre du nœud.

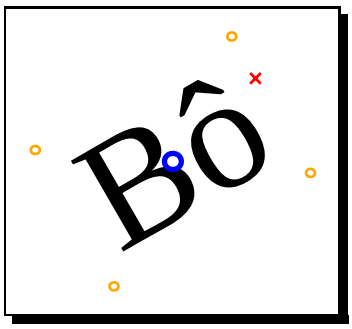
Avec une option littérale, **nodepoint** renvoie la valeur associée au littéral considéré dans le dictionnaire associé au nœud.

On peut également utiliser une facilité proposée par le format : par défaut lors du dessin d'un nœud, tous les points du dictionnaire *Pictdict* sont transférés dans le dictionnaire courant, suffixés par le nom du nœud. La commande **Abb** désigne ainsi le point *bb* du nœud *A*.

Les commandes **loadnodedictOn** et **loadnodedictOff** servent à activer ou désactiver ce transfert automatique vers le dictionnaire courant.

Dans l'exemple ci-dessous, on affiche la chaîne de caractères *Bô* après lui avoir fait subir agrandissement et rotation, puis on récupère les coordonnées des quatres sommets du cadre associé (affichés avec des cercles oranges), ainsi que celles du centre (affiché avec un cercle bleu), ainsi que celles du point de la bordure définissant un angle de 45° par

rapport au centre du nœud (affiché avec une crois rouge) :



Utilisation des points spéciaux

```
20 setfontsize
3 setdboxit
setTimes

(B) Rnode                        %% declaration du noeud
(Bô) 0 0 [3 dup] {30} cctext %% affichage du contenu
                                 %% du noeud
bleu
(B) nodepoint circ2              %% le point central

orange                           %% les sommets du cadre
(B) /ul nodepoint circ
Bur circ
Bdr circ
Bdl circ

rouge                            %% point a 45°
(B) 45 nodepoint times2
```

2. Gestion des arbres

La partie exposée dans ce paragraphe est une tentative de transcription conforme (du point de vue utilisateur) de la gestion des arbres par PSTricks. Je suis encore très loin du compte, ce n'est qu'un début. . .

2.1 - Les nœuds d'arbre

2.1.1 - Construction

Un nœud d'arbre est un objet complexe comprenant plusieurs champs : son type (rectangulaire, rectangulaire encadré, circulaire ou circulaire de rayon fixe), son contenu (vide, chaîne de caractères, label TEX, objet de l'environnement picture), son type de positionnement (par rapport à la *baseline* du contenu ou par rapport au centre de la *Bounding Box* du contenu), des paramètres éventuels, des indicateurs d'état (relié au père ou non), etc. . .

Les macros de construction de nœuds d'arbre sont les suivantes :

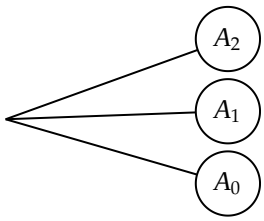
Tr	construit un nœud rectangulaire
TR	construit un nœud rectangulaire avec un encadrement non dessiné
Tb	construit un nœud rectangulaire encadré
Tc	construit un nœud circulaire
TC	construit un nœud de rayon fixe *Circleradius*
Tdia	construit un nœud losange
Toval	construit un nœud ovale
Tf	construit un nœud fantôme
Tdot	construit un nœud point en utilisant la commande **dot**

Pour ces dernières, le contenu est positionné par rapport à sa *baseline*. Les macros suivantes positionnent le contenu par rapport au centre de sa *Bounding Box* :

Trc	construit un nœud rectangulaire
TRc	construit un nœud rectangulaire avec un encadrement non dessiné
Tbc	construit un nœud rectangulaire encadré
Tcc	construit un nœud circulaire
TCc	construit un nœud de rayon fixe *Circleradius*
Tdiac	construit un nœud losange
Tovalc	construit un nœud ovale

Les nœuds d'arbre peuvent avoir trois types de contenu : chaîne de caractères, label TEX, ou objet de l'environnement picture. Pour passer en argument un contenu de type chaîne de caractères, la syntaxe est assez naturelle : par exemple **(Hello !) Tr** transmettra la chaîne *(Hello !)* au constructeur de nœud. Pour les labels TEX, la syntaxe est moins naturelle : les labels TEX sont tous indexés selon leur ordre de compilation dans le source jps (en commençant par l'indice 0). Ainsi, le littéral **/label0** va-t-il désigner le premier label TEX rencontré dans le source jps, alors que **/label1** va désigner le deuxième, etc. . . En d'autres termes, les commandes de construction de nœud acceptent un littéral de la forme **/label***i* où *i* est l'indice du label TEX passé en argument. Enfin pour un objet de l'environnement

picture, on donne la chaîne de caractères commençant par un point d'exclamation et suivie par le nom de l'objet. Ainsi la commande **(!monobjet) Tr** transmettra l'objet *monobjet* au constructeur de nœud.



Labels TEX

```
#tex# $A_0$
#tex# $A_1$
#tex# $A_2$
/levelsep 4 def
0 0 Tr rootnode
[right /label0 Tc /label1 Tc /label2 Tc]
tree*
```

Plus précisément :

string/*lit* **Tr** *tnode* ⟶ Construit un nœud d'arbre rectangulaire dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **TR** *tnode* ⟶ Construit un nœud d'arbre rectangulaire avec un cadre non dessiné, et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tb** *tnode* ⟶ Construit un nœud d'arbre rectangulaire encadré dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tc** *tnode* ⟶ Construit un nœud d'arbre circulaire dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **TC** *tnode* ⟶ Construit un nœud d'arbre circulaire de rayon fixe *Circleradius* dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tdia** *tnode* ⟶ Construit un nœud d'arbre en forme de losange et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Toval** *tnode* ⟶ Construit un nœud d'arbre en forme d'ellipse et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

– **Tf** *tnode* ⟶ Construit un nœud d'arbre fantôme (contenu vide et non relié au nœud père)

– **Tdot** *tnode* ⟶ Construit un nœud d'arbre contenant un point (en utilisant la commande **dot**)

string/*lit* **Trc** *tnode* ⟶ Construit un nœud d'arbre rectangulaire dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **TRc** *tnode* ⟶ Construit un nœud d'arbre rectangulaire avec un cadre non dessiné, et dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tbc** *tnode* ⟶ Construit un nœud d'arbre rectangulaire encadré dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tcc** *tnode* ⟶ Construit un nœud d'arbre circulaire dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **TCc** *tnode* ⟶ Construit un nœud d'arbre circulaire de rayon fixe *Circleradius* dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tdiac** *tnode* ⟶ Construit un nœud d'arbre en forme de losange et dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

string/*lit* **Tovalc** *tnode* ⟶ Construit un nœud d'arbre en forme d'ellipse et dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

2.1.2 - Nœud racine

Chaque arbre a besoin d'un nœud racine. Ce qui le différencie des autres nœuds d'arbre, c'est que ses coordonnées sont fixées par l'utilisateur. On utilise pour cela la commande **rootnode** couplée avec une autre commande de construction de nœud. Plus précisément, la syntaxe est :

x y tnode **rootnode** *tnode* ⟶ Dépose les coordonnées (x, y) dans le champ adapté du nœud d'arbre *tnode*, et le nomme *A* si celui-ci n'a pas de nom.

Par exemple, **0 0 Tr rootnode** va déposer sur la pile un nœud racine rectangulaire de contenu vide et de coordonnées $(0, 0)$.

Par défaut, **rootnode** construit un nœud de nom *A*.

2.1.3 - Modification des paramètres d'un nœud

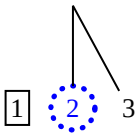
Lorsqu'un nœud a été construit avec les commandes précédentes, on peut modifier un certain nombre de ses champs : le nom, les paramètres et l'indicateur de liaison au père. Plus précisément, on dispose des trois commandes suivantes :

tnode **`non_relie`** *tnode* ⟶ Positionne à *false* le champ adapté du nœud d'arbre *tnode*

tnode string **`nomme_noeud`** *tnode* ⟶ Positionne à *string* le champ *nom* du nœud d'arbre *tnode*

tnode proc **`tnparametres`** *tnode* ⟶ Définit les paramètres du nœud d'arbre *tnode* par *proc*

Dans l'exemple ci-dessous, le premier nœud fils n'est pas relié, le second voit ses paramètres graphique changer, alors que le troisième sera dénommé *B*, et pourra être désigné comme tel par la suite.



modification des attributs

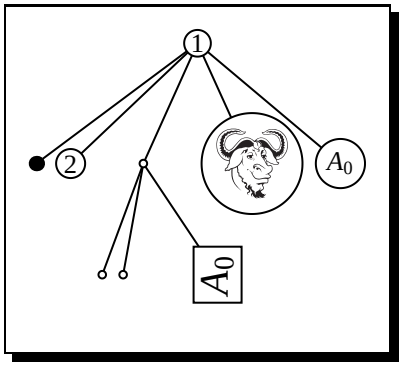
```
setTimes
0 5 Tr rootnode
[
   (1) Tb non_relie
   (2) Tc {2 setlinewidth bleu dotted} tnparametres
   (3) TR (B) nomme_noeud
]
tree
```

2.1.4 - Agrandissement, rotation ou décalage d'un nœud

Chaque nœud d'arbre peut subir un agrandissement/réduction, une rotation ou un décalage. Pour cela, on utilise la commande **`Toptaff`** (option d'affichage) qui prend en paramètre un tableau du type

[(*dx dy*) [$scale_x scale_y$] {α} **]**

où (*dx*, *dy*) est le déplacement en picas à rajouter au déplacement initial, ($scale_x$, $scale_y$) est le facteur d'agrandissement, et α l'angle de la rotation. Chacune des ces trois données est optionnelle.



source jps

```
1 setdboxit
autocrop
/dotscale {1.2 dup} def

/nodesep 0 def
/treenodesep 5 def

setTimes
#tex# $A_0$
-2 5 (1) Tc rootnode

[
   Tdot
   (2) Tcc
   Tc
   [
      Tc
      Tc
      /label0 Tbc [(0 -20) [1.5 dup] {90}] Toptaff
   ]
   (!gnuhead) Tcc [[.05 dup]] Toptaff
   /label0 Tcc
]
tree*
```

2.2 - Arbres et sous-arbres

Il y a deux façons de tracer un arbre : la plus simple qui consiste à demander un équilibrage automatique, et l'autre où l'utilisateur règle la distance séparant les centres des nœuds d'un même niveau. Cette dernière méthode permet d'obtenir des arbres d'une grande symétrie, ainsi que des arbres ayant des nœuds superposés.

Dans les deux cas le paramètre *levelsep* est utilisé. Il indique le nombre d'unités (dans le repère jps) séparant deux étages de l'arbre. Dans le cas de l'équilibrage automatique, l'autre paramètre utilisé est *nodetreesep*, indiquant la distance en picas séparant le bord de deux nœuds de niveau terminal. Pour l'équilibrage « manuel », c'est le paramètre *treesep* qui indique le nombre d'unités (dans le repère jps) séparant deux nœuds du même niveau.

Pour dessiner un arbre à équilibrage automatique, on utilise la commande **`tree*`**. Pour un équilibrage manuel, on utilise la commande **`tree`**. Si l'on souhaite un équilibrage automatique sur un sous-arbre seulement, on passe la commande **`autotreesepOn`** dans les paramètres du sous-arbre. *A contrario*, si l'on souhaite désactiver l'équilibrage automatique sur un sous-arbre, on passe la commande **`autotreesepOff`** dans les paramètres du sous-arbre.

Un arbre est constitué de deux éléments : un nœud racine et un sous-arbre.

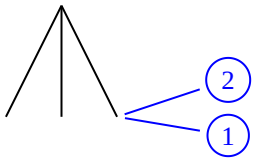
Un sous-arbre est un tableau contenant une direction (optionnelle, valant **`down`** par défaut), un exécutable (optionnel, le plus souvent des paramètres) ainsi que des nœuds (éventuellement aucun) et des sous-arbres (éventuellement aucun).

Pour les directions, on utilise les commandes **`up`**, **`down`**, **`left`**, **`right`** ou **`dir`**, la commande **`dir`** prenant en argument un angle en degrés.

arbre simple

```
0 5 Tr rootnode
[Tr Tr Tr]
tree
```

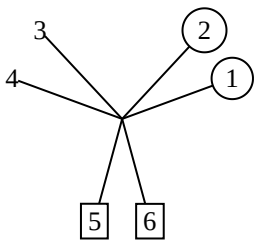
L'exécutable d'un sous-arbre permet de changer localement (c'est à dire dans l'ensemble du sous-arbre) les valeurs de certains paramètres. Dans l'exemple ci-dessous, on modifie un paramètre graphique (la couleur) ainsi qu'un parmètre défini dans un dictionnaire (le paramètre *nodesep* indiquant l'espace en picas séparant le nœude de l'extrémité de la connexion).



avec un sous-arbre

```
setTimes
0 5 Tr rootnode
[
   Tr Tr Tr
   [right {bleu /nodesep 3 def}
      (1) Tc (2) Tc
   ]
]
tree
```

Lorsque l'une ou l'autre des commandes **`tree`** rencontre un sous-arbre, elle le relie au nœud de niveau supérieur déjà tracé. Un même nœud peut donc être le père de plusieurs sous-arbres.



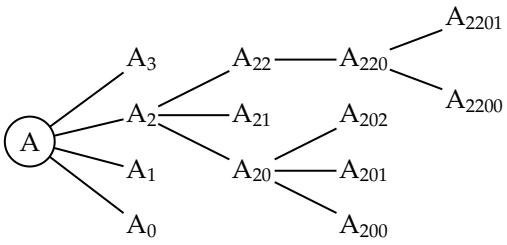
avec 3 sous-arbres

```
setTimes
0 0 Tr rootnode
[
   [30 dir (1) Tc (2) Tc]
   [150 dir (3) Tr (4) Tr]
   [down (5) Tb (6) Tb]
]
tree
```

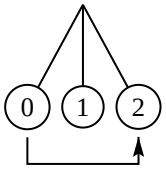
2.3 - Noms des nœuds

Lorsqu'un arbre est dessiné par l'une des commandes **`tree`**, chacun de ses nœuds a été nommé (y compris les nœuds fantômes), ce qui permet d'y faire référence par la suite. En particulier, en utilisant la commande **`nodepoint`** vu dans un précédent paragraphe, on peut récupérer les coordonnées de n'importe lequel des points de référence (soit au minimum 16) de n'importe quel nœud.

Par défaut, chaque nœud hérite du nom du père, suffixé par son indice dans le niveau (le premier nœud d'un niveau donné ayant pour indice 0). Ainsi, dans le dessin ci-dessous, le nœud racine a pour nom *A*, les 4 nœuds du premier niveau ont pour nom A_0, A_1, A_2 et A_3, etc. . .



On peut ainsi relier *a posteriori* certains nœuds :



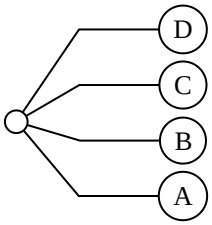
liaisons a posteriori

```
setTimes
0 0 Tr rootnode
[(0) Tc (1) Tc (2) Tc]
tree
/nodesep 3 store
/angleA -90 store
(A0) (A2) (->) ncbar
```

Néammoins, lorsque plusieurs sous-arbres sont connectés à un même nœud, on est confronté à un problème de surcharge des noms de nœuds. Par exemple, si le nœud père s'appelle *A* et que l'on a trois sous-arbres de chacun trois nœuds, alors seuls les nœuds du dernier sous-arbre seront accessibles par les noms A_0, A_1 et A_2. Si l'on a besoin d'accéder à un nœud d'un autre sous-arbre, il faut le nommer spécifiquement avec la commande **`nomme_noeud`**.

2.4 - Liaisons

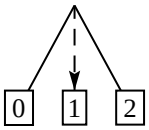
Pour relier deux nœuds d'arbres entre eux, la commande **`treeedge`**, initialisée à **`{(-) ncline}`** est utilisée. Une redéfinition de cette commande changera le type de connexion (on peut utiliser toutes les commandes disponibles pour les connexions de nœuds ; voir le paragraphe dédié).



changement du type de liaison

```
setTimes
/treeedge {exch (-) ncdiagg} def
/levelsep 3 store
/armA 30 store
/angleA 180 store
0 0 Tc rootnode
[right (A) Tc (B) Tc (C) Tc (D) Tc]
tree
```

Pour changer ponctuellement le type de liaison sur un niveau, on peut éviter de dessiner cette liaison (avec l'attribut **`non_relie`**) puis la dessiner *a posteriori.*

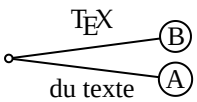


changement ponctuel de liaison

```
setTimes
0 0 Tr rootnode
[down (0) Tb (1) Tb non_relie (2) Tb]
tree
pointilles
(A) (A1) (->) ncline
```

2.5 - Labels sur les liaisons

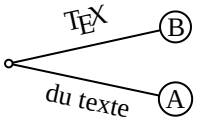
Il est possible d'ajouter des labels (chaines de caractères ou labels (La)TEX) sur les liaisons inter-nœuds. Pour de tels affichages, le point de référence est le milieu du segment joignant les deux nœuds considérés. Deux jeux de 16 commandes sont disponibles. Le premier utilise le suffixe **put** et les 16 préfixes habituels (**ul**, **ub**, **uc**, **ur**, etc. . .) et place la chaine de caractères ou le label TEX passé en argument à l'endroit spécifié par rapport au point de référence :



Les commandes **put**

```
1 setdboxit setTimes
/levelsep 3 def /treenodesep 4 def
#tex# \TeX
-5 0 Tc rootnode
[right
   (A) Tcc (du texte) dcput
   (B) Tcc /label0 ucput
]
tree*
```

Le second jeu utilise le suffixe **put*** et les 16 préfixes usuels. Il permet de placer le label selon l'angle défini par les deux nœ ds considérés :



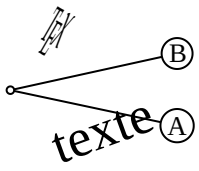
Les commandes **put***

```
1 setdboxit setTimes
/levelsep 3 def /treenodesep 15 def
#tex# \TeX
-5 0 Tc rootnode
[right
   (A) Tcc (du texte) dcput*
   (B) Tcc /label0 ucput*
]
tree*
```

Enfin, ces commandes supportent le passage d'options (translation, changements d'échelle et rotation pour la série **put**, translation et changements d'échelle pour la série étoilée). Elles se tranmettent sous la forme d'une chaine de caractère elle même encapsulée dans une accolade. Par exemple :

(la chaine){((*dx dy*) [$scale_x$ $scale_y$] {α} **)}ucput**

signifie « placer le texte **la chaine** au point de référence, après lui avoir fait subir la rotation d'angle α, le changement d'échelle ($scale_x$, $scale_y$) et la translation (*dx*, *dy*) », le vecteur translation étant exprimé en picas.



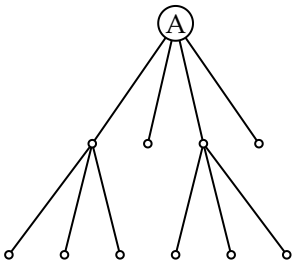
Option des commandes **put**

```
1 setdboxit setTimes
/levelsep 3 def /treenodesep 15 def
#tex# \TeX
-5 0 Tc rootnode
[right
   (A) Tcc (texte) {([2 2] {20})} dcput
   (B) Tcc /label0 {((-20 -10) [.5 2] {-30})}ucput
]
tree*
```

2.6 - Exemples

Quelques exemples pour conclure.

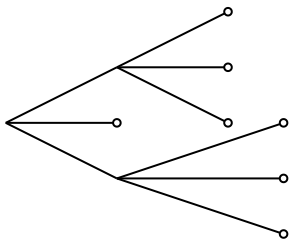
L'utilisation de nœuds fantômes (commande **`Tf`**, nœuds non représentés et non reliés) permet d'équilibrer différemment certains arbres :



nœuds fantômes

```
1 setdboxit
setPalatino
0 5 (A) Tc rootnode
[
   Tc [Tc Tc Tc Tf]
   Tc
   Tc [Tf Tc Tc Tc]
   Tc
]
tree
```

Les paramètres d'un sous-arbre affectent l'ensemble de ce sous-arbre. Cependant il existe une variante pour deux de ces paramètres afin d'effectuer des changements locaux, i.e qui n'affectent que le niveau courant. Ce sont les variables *thislevelsep* et *thistreesep* :

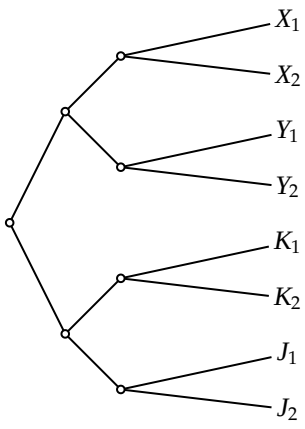


thislevelsep et thistreesep

```
1 setdboxit
-5 0 Tr rootnode
[right
   Tr [{/thislevelsep 3 def} Tc Tc Tc]
   Tc
   Tr [Tc Tc Tc]
]
tree
```

Parfois, il peut-être utile de définir un réglage pour l'ensemble d'un niveau à travers tout un arbre. On utilise à cet effet les commandes **`treehook`**roman(*n*) où *n* est est le niveau de profondeur où s'applique la commande (c'est une reprise de la syntaxe PSTricks, si je l'ai bien comprise. . .). Ainsi la commande **`treehooki`** sera exécutée sur le premier

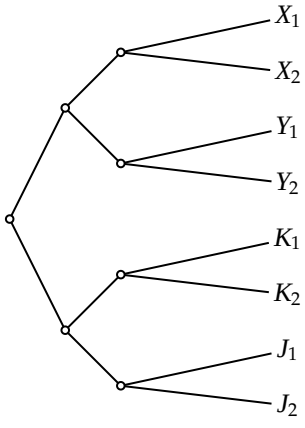
niveau, puis la commande **`treehookii`** sur le deuxième niveau, etc. . . Le premier niveau dessiné est d'indice 0.



les treehook

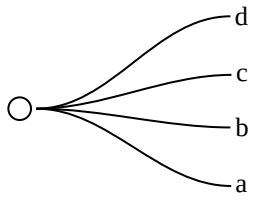
```
#tex# $J_2$
#tex# $J_1$
#tex# $K_2$
#tex# $K_1$
#tex# $Y_2$
#tex# $Y_1$
#tex# $X_2$
#tex# $X_1$
1 setdboxit setTimes
/levelsep 1 def /treesep 4 def
/treehooki {/thistreesep 2 def} def
/treehookii {
   /thislevelsep 3 def
   /thistreesep 1 def
} def

-5 0 Tc rootnode
[right
   Tc
   [
      Tc [/label0 TR /label1 TR]
      Tc [/label2 TR /label3 TR]
   ]
   Tc
   [
      Tc [/label4 TR /label5 TR]
      Tc [/label6 TR /label7 TR]
   ]
]
tree
```

Et voici le même arbre, mais avec un équilibrage automatique (commande **`tree*`**) et une présentation différente pour le code des labels TEX.



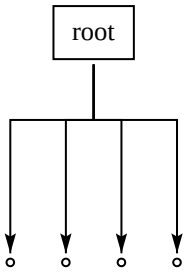
équilibrage automatique

```
1 setdboxit setTimes
/levelsep 1 def /treenodesep 4 def
/treehookii {
   /thislevelsep 3 def
} def

-5 0 Tc rootnode
[right
   Tc
   [
#tex# $J_2$
#tex# $J_1$
      Tc [/label0 TR /label1 TR]
#tex# $K_2$
#tex# $K_1$
      Tc [/label2 TR /label3 TR]
   ]
   Tc
   [
#tex# $Y_2$
#tex# $Y_1$
      Tc [/label4 TR /label5 TR]
#tex# $X_2$
#tex# $X_1$
      Tc [/label6 TR /label7 TR]
   ]
]
tree*
```



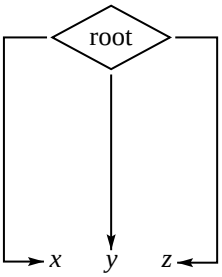
type de liaison

```
setTimes
/treeedge {(-) nccurve} def
/levelsep 4 store
/angleB 180 store
/nodesep 2 store
0 0 Tc rootnode
[right (a) Tr (b) Tr (c) Tr (d) Tr]
tree
```



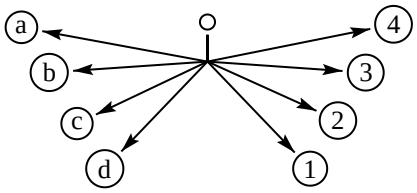
passage de paramètres

```
setTimes
/treeedge {(->) ncangle} def
/levelsep 4 store
/angleA -90 store
/angleB 90 store
/armB 50 store
/nodesep 2 store
7 setdboxit
0 0 (root) Tb rootnode
[{1 setdboxit} Tc Tc Tc Tc]
tree
```



sans titre

```
setTimes
/arrowscale {.8 dup} def
/nodesep 2 store
/levelsep 4 store
/treeedge {(->) ncline} def
3 setdiaboxit
0 0 (root) Tdia rootnode
[{setTimesItalic}
   (x) Tr non_relie
   (y) Tr
   (z) Tr non_relie
]
tree
/armA 15 store
(A) (A2) (->) ncbar
/angleA 180 store
/angleB 180 store
(A) (A0) (->) ncbar
```



avec **ncdiagg**

```
2 setdboxit
setTimes
/treeedge {(->) ncdiagg} def
/levelsep 3 def /nodesep 2 def
/angleB 90 def /angleA -90 def
0 5 Tc rootnode
[
   [-30 dir (1) Tc (2) Tc (3) Tc (4) Tc]
   [-150 dir (a) Tc (b) Tc (c) Tc (d) Tc]
]
tree
```

VII - Contributions – Bibliographie

Contributions

Jean-Michel Sarlat, Philippe Saadé, Guillaume Haberer, Manuel Luque Nicolas Roux, Dominique Petit, Jean-Louis Gallinari, Christophe Jorsen, Luís A. V. Ferreira, W. R. Lee, pour leur aide, leur code, ou leurs idées...

Références

[1]Donald Knuth. *The METAFONTbook*. Addison Wesley, 1986.

[2]*Manuel de référence du langage PostScript*, d'Adobe Systems Incorporation, Addison-Wesley, 1992.

[3]Bill Casselman. *Mathematical Illustrations*. , 1996. **www.math.ubc.ca/~cass/graphics/text/www/**

[4]Denis Girou, *Présentation de PSTricks*. Cahiers GUTenberg, n° 16, avril 1994

[5]Herbert Voss et W. R. Lee, *Prologue pst-grad.pro*,
www.perce.de/LaTeX/pstricks/dvips/pst-grad.pro

Annexe I : résumé des opérateurs Postscript

Nota : Les commandes ci-dessous sont recopiées du *Manuel de référence du langage PostScript*, d'Adobe Systems Incorporation, chez Addison-Wesley, 1992, conformément à l'autorisation Adobe publiée à la page 11 de cet ouvrage.

1. Opérateurs de manipulation de la pile d'opérandes

any **pop** $-\longrightarrow$ enlève l'élément supérieur

$any_1\ any_2$ **exch** $any_2\ any_1 \longrightarrow$ intervertit les deux éléments en haut de la pile

any **dup** *any any* $\longrightarrow$ duplique l'élément au sommet

$any_1 \ldots any_n\ n$ **copy** $any_1 \ldots any_n\ any_1 \ldots any_n \longrightarrow$ duplique n éléments au sommet

$any_n \ldots any_0\ n$ **index** $any_n \ldots any_0\ any_n \longrightarrow$ duplique un élément déterminé

$a_{n-1} \ldots a_0\ n\ j$ **roll** $a_{(j-1) \bmod n} \ldots a_0\ a_{n-1} \ldots a_{j \bmod n} \longrightarrow$ permute n éléments j fois

$\vdash any_1 \ldots any_n$ **clear** $\vdash \longrightarrow$ enlève tous les éléments

$\vdash any_1 \ldots any_n$ **count** $\vdash any_1 \ldots any_n\ n \longrightarrow$ compte les éléments dans la pile

$-$ **mark** *mark* $\longrightarrow$ place une marque sur la pile

marks $obj_1 \ldots obj_n$ **cleartomark** $- \longrightarrow$ enlève les éléments jusqu'à *mark*

marks $obj_1 \ldots obj_n$ **counttomark** *marks* $obj_1 \ldots obj_n\ n \longrightarrow$ compte les élémens jusqu'à *mark*

2. Opérateurs arihmétiques et mathématiques

$num_1\ num_2$ **add** *sum* $\longrightarrow num_1$ plus num_2

$num_1\ num_2$ **div** *quotient* $\longrightarrow num_1$ divisé par num_2

$int_1\ int_2$ **idiv** *quotient* $\longrightarrow$ division entière

$int_1\ int_2$ **mod** *remainder* $\longrightarrow int_1 \bmod int_2$

$num_1\ num_2$ **mul** *product* $\longrightarrow num_1$ multiplié par num_2

$num_1\ num_2$ **sub** *difference* $\longrightarrow num_1$ moins num_2

num_1 **abs** $num_2 \longrightarrow$ valeur absolue de num_1

num_1 **neg** $num_2 \longrightarrow$ opposé de num_1

num_1 **ceiling** $num_2 \longrightarrow$ plafond de num_1

num_1 **floor** $num_2 \longrightarrow$ plancher de num_1

num_1 **round** $num_2 \longrightarrow$ arrondit num_1 à l'entier le plus proche

num_1 **truncate** $num_2 \longrightarrow$ enlève la partie fractionnaire de num_1

num **sqrt** *real* $\longrightarrow$ racine carrée de *num*

num den **atan** *angle* $\longrightarrow$ arc tangente de num/den en degrés

angle **cos** *real* $\longrightarrow$ cosinus de *angle* (en degrés)

angle **sin** *real* $\longrightarrow$ sinus de *angle* (en degrés)

base exponent **exp** *real* $\longrightarrow$ élève *base* à la puissance *exponent*

num **ln** *real* $\longrightarrow$ logarithme naturel (base e)

num **log** *real* $\longrightarrow$ logarithme décimal (base 10)

$-$ **rand** *int* $\longrightarrow$ génère un entier au hasard

int **srand** $- \longrightarrow$ paramètre le nombre d'origine du générateur aléatoire

$-$ **rrand** *int* $\longrightarrow$ renvoie le nombre d'origine du générateur aléatoire

3. Opérateurs de tableau

int **array** *array* $\longrightarrow$ crée un tableau de longueur *int*

$-$ **[** *mark* $\longrightarrow$ commence la construction de tableau

mark $obj_0 \ldots obj_{n-1}$ **]** *array* $\longrightarrow$ termine la construction de tableau

array **length** *int* $\longrightarrow$ nombre d'éléments dans *array*

array index **get** *any* $\longrightarrow$ obtient l'élément du tableau indexé par *index*

array index any **put** $- \longrightarrow$ met *any* dans *array* à *index*

array index count **getinterval** *subarray* $\longrightarrow$ sous-tableau de *array* commençant à *index* et long de *count* éléments

array$_1$ *index array*$_2$ **putinterval** – ⟶ remplace le sous-tableau *array*$_1$ commençant à *index* par *array*$_2$

any$_0$ ... *any*$_{n-1}$ *array* **astore** *array* ⟶ pousse l'élément depuis la pile vers array

array **aload** $a_0 \ldots a_{n-1}$ *array* ⟶ pousse tous les éléments de *array* dans la pile

array$_1$ *array*$_2$ **copy** *subarray*$_2$ ⟶ copie les éléments de *array*$_1$ dans le sous-tableau initial de *array*$_2$

array proc **forall** – ⟶ éxécute *proc* pour chaque élément dans *array*

4. Opérateurs de dictionnaire

int **dict** *dict* ⟶ crée un dictionnaire ayant une contenance de *int* éléments

– **<<** *mark* ⟶ commence la construction de dictionnaire

mark key$_1$ *value*$_1$ ... *key*$_n$ *value*$_n$ **>>** *dict* ⟶ termine la construction de dictionnaire

dict **length** *int* ⟶ nombre de couples clé-valeur dans *dict*

dict **maxlength** *int* ⟶ capacité courante de *dict*

dict **begin** – ⟶ pousse *dict* dans la pile des dictionnaires

– **end** – ⟶ expulse la pile des dictionnaires

key value **def** – ⟶ associe *key* et *value* dans le dictionnaire courant

key **load** *value* ⟶ recherche *key* dans la pile des dictionnaires et renvoie le *vlue* qui y est associé

key value **store** – ⟶ remplace la définition la plus haute de *key*

dict key value **put** – ⟶ associe *key* à *value* dans *dict*

dict key **undef** – ⟶ enlève *key* et *value* dans *dict*

dict key **known** *bool* ⟶ test si *key* est dans *dict*

key **where** *dict true* ou bien *false* ⟶ trouve le dictionnaire dans lequel *key* est défini

dict proc **forall** – ⟶ exécute *proc* pour chaque élément de *dict*

– **currentdict** *dict* ⟶ pousse le dictionnaire courant dans la pile des opérandes

– **errordict** *dict* ⟶ dictionnaire des gestions d'erreur

– **$error** *dict* ⟶ dictionnaire de contrôle des erreurs et de statut

– **systemdict** *dict* ⟶ dictionnaire système

– **userdict** *dict* ⟶ dictionnaire en écriture en VM locale

– **globaldict** *dict* ⟶ dictionnaire en écriture en VM globale

– **statusdict** *dict* ⟶ dictionnaire dépendant du produit

– **countdictstack** *int* ⟶ compte les éléments dans la pile des dictionnaires

array **dictstack** *subarray* ⟶ copie la pile des dictionnaires dans *array*

– **cleardictstack** – ⟶ enlève tous les dictionnaires non permanents de la pile des dictionnaires

5. Opérateurs de chaînes

int **string** *string* ⟶ crée une chaîne de longueur *int*

string **length** *int* ⟶ nombre d'éléments dans *string*

string index **get** *int* ⟶ obtinet l'élement de *string* indexé par *index*

string index int **put** – ⟶ place *int* dans *string* à *index*

string index count **getinterval** *substring* ⟶ sous-chaîne de *string* commençant à *index* pour *count* éléments

string$_1$ *index string*$_2$ **putinterval** – ⟶ remplace la sous-chaîne de *string*$_1$ commençant à *index* par *string*$_2$

string$_1$ *string*$_2$ **copy** *substring*$_2$ ⟶ copie les élément de *string*$_1$ dans la sousè-chaîne initiale de *string*$_2$

string proc **forall** – ⟶ exécute *proc* pour chaque élément de *string*

string seek **anchorsearch** *post match true* ou bien *string false* ⟶ détermine si *seek* est une sous-chaîne de *string*

string seek **search** *post match pre true* ou bien *string false* ⟶ cherche *seek* dans *string*

string **token** *post token true* ou bien *string false* ⟶ lit le léxème à partir du début de *string*

6. Opérateurs relationnels, booléens et bit à bit

any$_1$ *any*$_2$ **eq** *bool* ⟶ test l'égalité

any$_1$ *any*$_2$ **ne** *bool* ⟶ test l'inégalité

num$_1$/*str*$_1$ *num*$_2$/*str*$_2$ **ge** *bool* ⟶ teste si supérieur ou égal

num$_1$/*str*$_1$ *num*$_2$/*str*$_2$ **gt** *bool* ⟶ teste si supérieur

num$_1$/*str*$_1$ *num*$_2$/*str*$_2$ **le** *bool* ⟶ test si inférieur

num$_1$/*str*$_1$ *num*$_2$/*str*$_2$ **lt** *bool* ⟶ teste si inférieur ou égal

bool$_1$/*int*$_1$ *bool*$_2$/*int*$_2$ **and** *bool*$_3$/*int*$_3$ ⟶ et logique bit à bit

bool$_1$/*int*$_1$ **not** *bool*$_2$/*int*$_2$ ⟶ non logique bit à bit

num$_1$/*string*$_1$ *num*$_2$/*string*$_2$ **or** *bool*$_3$/*int*$_3$ ⟶ ou inclusif logique bit à bit

num$_1$/*string*$_1$ *num*$_2$/*string*$_2$ **xor** *bool*$_3$/*int*$_3$ ⟶ ou exclusif logique bit à b it

– **true** *true* ⟶ pousse la valeur booléenne *true* (vrai)

– **false** *false* ⟶ pousse la valeur booléenne *false* (faux)

int$_1$ *shift* **bitshift** *int*$_2$ ⟶ décalement bit à bit de *int*$_1$ (positif à gauche)

7. Opérateurs de contrôle

any **exec** – ⟶ exécute un objet arbitraire

bool proc **if** – ⟶ exécute *proc* si *bool* est *true*

bool proc$_1$ *proc*$_2$ **ifelse** – ⟶ exécute *proc*$_1$ si *bool* est *true*, *proc*$_2$ sinon

init incr limit proc **for** – ⟶ exécute *proc* avec les valeurs à partie de *init* par pas de *inc* jusqu'à *limit*

int proc **repeat** – ⟶ exécute *int* fois *proc*

proc **loop** – ⟶ exécute *proc* un nombre indéfini de fois

– **exit** – ⟶ quitte la boucle active la plus interne

– **stop** – ⟶ termine le contexte **stopped**

any **stopped** *bool* ⟶ établit le contexte pour attraper **stop**

– **countexecstack** *int* ⟶ compte les éléments dans la pile d'éxécution

array **execstack** *subaray* ⟶ copie la pile d'exécution dans *array*

– **quit** – ⟶ quitte l'interpréteur

– **start** – ⟶ exécuté au lancement de l'interpréteur

8. Opérateurs de type, attribut et conversion

any **type** *name* ⟶ renvoie le nom identifiant le type de *any*

any **cvlit** *any* ⟶ convertit l'objet en littéral

any **cvx** *any* ⟶ convertit l'objet en exécutable

any **xcheck** *bool* ⟶ teste l'attribut d'exécutable

array/*packedarray*/*file*/*string* **executeonly** – *array*/*packedarray*/*file*/*string* ⟶ réduit l'accès à exécutablea seulement

array/*packedarray*/*dict*/*file*/*string* **noaccess** – *array*/*packedarray*/*dict*/*file*/*string* ⟶ interdit tout accès

array/*packedarray*/*dict*/*file*/*string* **readonly** – *array*/*packedarray*/*dict*/*file*/*string* ⟶ réduit l'accès à lecture seule

array/*packedarray*/*dict*/*file*/*string* **rcheck** *bool* ⟶ teste l'accès en lecture

array/*packedarray*/*dict*/*file*/*string* **wcheck** *bool* ⟶ teste l'accès en écriture

num/*string* **cvi** *int* ⟶ convertit en entier

string **cvn** *name* ⟶ convertit en nom

num/*string* **cvr** *real* ⟶ convertit en réel

num radix string **cvrs** *substring* ⟶ convertit en chaîne avec racine

any string **cvs** *substring* ⟶ convertit en chaîne

9. Opérateurs de fichier

string$_1$ *string*$_2$ **file** *file* ⟶ ouvre le fichier idéntifié par *string*$_1$ avec l'accès *string*$_2$

src/*tgt*/*param*$_1$ ... *param*$_n$ *name* **filter** *file* ⟶ établit le fichier filtré

file **closefile** – ⟶ ferme *file*

file **read** *int true* ou bien *false* ⟶ lit un caractère dans *file*

file int **write** – ⟶ écrit un caractère dans *file*

file string **readhexstring** *substring bool* ⟶ lit une chaîne hexadécimale depuis *file* vers *string*

file string **writehexstring** – ⟶ écrit *string* dans *file* en chaîne hexadécimale

file string **readstring** *substring bool* ⟶ lit une ligne depuis *file* dans *string*

file string **writestring** – ⟶ écrit *string* dans *file*

file string **readline** *substring bool* ⟶ lit une ligne depuis *file* dans *string*

file **token** *token true* ou bien *false* ⟶ lit un léxème depuis *file*

file **bytesavailable** *int* ⟶ nombre d'octets pouvant être lus

– **flush** – ⟶ renvoie les données du tampon dans le fichier de sortie standard

file **flushfile** – ⟶ renvoie les données du tampon ou les lit jusqu'à EOF

file **resetfile** – ⟶ ignore les caractères dans le tampon

file **status** *bool* ⟶ renvoie le statut de *file*

string **status** *pages bytes referenced created true* ou bien *false* ⟶ renvoie l'information à propos du fichier nommé

string **run** – ⟶ exécute le contenu du fichier nommé

– **currentfile** *file* ⟶ renvoie le fichier en cours d'exécution

any$_1$ ... *any*$_n$ *string* **deletefile** – ⟶ détruit le fichier nommé

string$_1$ *string*$_2$ **renamefile** – ⟶ renomme le fichier appelé *string*$_1$ en *string*$_2$

template proc scratch **filenameforall** – ⟶ exécute *proc* pour chaque nom de fichier correspondant à *template*

file int **setfileposition** – ⟶ met *file* à la position indiquée

file **fileposition** *int* ⟶ renvoie la position courant dans *file*

string **print** – ⟶ écrit *string* dans le fichier de sortie standard

any **=** – ⟶ écrit la représentation textuelle de *any* dans le fichier de sortie standard

any **==** – ⟶ écrit la représentation syntaxique de *any* dans le fichier de sortie standard

⊢ *any*$_1$ ... *any*$_n$ **stack** – ⊢ *any*$_1$ ... *any*$_n$ ⟶ imprime la pile de façon non destructive en utilisant **=**

⊢ *any*$_1$ ... *any*$_n$ **pstack** – ⊢ *any*$_1$ ... *any*$_n$ ⟶ imprime la pile de façon non destructive en utilisant **==**

obj int **printobject** – ⟶ écrit un objet binaire dans le fichier de sortie standard, en utilisant *int* en tant que drapeau

file obj int **writeobject** – ⟶ écrit un objet binaire dans *file* en utilisant *int* en tant que drapeau

int **setobjectformat** – ⟶ définit le format d'objet binaire (0 = désactivé, 1 = IEEE haut, 2 = bas, 3 = haut natif, 4 = bas)

– **currentobjectformat** *int* ⟶ renvoie le format d'objet binaire

10. Opérateurs de ressource

key instance category **defineresource** *instance* ⟶ enregistre la ressource nommée *instance* dans *category*

key category **undefinedresource** – ⟶ enlève l'enregistrement de ressource

key category **findresource** *instance* ⟶ renvoie la ressource *instance* identifiée par *key* dans *category*

key category **resourcestatus** *status size true* ou bien *false* ⟶ renvoie le *status* des instances de ressource

template proc scratch category **resourceforall** – ⟶ énumère les instances de ressources dans *category*

11. Opérateurs de mémoire virtuelle

– **save** *save* ⟶ crée une photo de la VM

save **restore** – ⟶ réinstalle la photo de la VM

bool **setglobal** – ⟶ définit le mode d'allocation en VM (*fale* = local, *true* = global)

– **currentglobal** *bool* ⟶ renvoie le mode courant d'allocation en VM

any **gcheck** *bool* ⟶ *true* si *any* est simple ou en VM globale, *false* si en VM locale

$bool_1$ *password* **startjob** $bool_2$ ⟶ commence une nouvelle tâche qui modifiera la VM initiale si *bool* est vrai

index any **defineuserobject** – ⟶ définit l'objet utilisateur associé à *index*

index **execuserobject** – ⟶ exécute l'objet utilisateur associé à *index*

index **undefineduserobject** – ⟶ enlève l'objet utilisateur associé à *index*

– **UserObject** *array* ⟶ tableau **UserObjects** courant défini dans **userdict**

12. Opérateurs divers

proc **bind** *proc* ⟶ remplace les noms d'opérateurs dans *proc* par les opérateurs

– **null** *null* ⟶ pousse *null* dans la pile d'opérandes

– **version** *string* ⟶ version de l'interpréteur

– **realtime** *int* ⟶ renvoie le temps réel en millisecondes

– **usertime** *int* ⟶ renvoie le temps d'exécution en millisecondes

– **languagelevel** *int* ⟶ niveau de fonction du langage

– **product** *string* ⟶ nom du produit

– **revision** *int* ⟶ numéro de révision du produit

– **serialnumber** *int* ⟶ numéro de série de la machine

– **executive** – ⟶ appelle l'exécuteur intéractif

bool **echo** – ⟶ alterne l'écho actif et inactif

– **prompt** – ⟶ exécuté quand prêt pour l'entrée intéractive

13. Opérateurs de de l'état graphique - Indépendants du périphérique

– **save** – ⟶ pousse l'état graphique

– **grestore** – ⟶ remet l'état graphique en place

– **grestorealll** – ⟶ remet l'état graphique le plus ancien en place

– **initgraphics** – ⟶ réinitialise les paramètres de l'état graphique

– **gstate** *gstate* ⟶ crée un objet état graphique

gstate **setgstate** – ⟶ établit l'état graphique à partir de *gstate*

gstate **currentgstate** *gstate* ⟶ copie l'état graphique courant dans *gstate*

num **setlinewidth** – ⟶ ajuste la largeur de ligne

– **currentlinewidth** *num* ⟶ renvoie la largeur de ligne courante

int **setlinecap** – ⟶ définit la forme des fins de ligne lors des tracés (0 = en ogive, 1 = rond, 2 = en biais)

– **currentlinecap** *int* ⟶ renvoie le recouvrement courant de la ligne

int **setlinejoin** – ⟶ définit la forme des coins lors du tracé (0 = en ogive, 1 = rond, 2 = en biais)

– **currentlinejoin** *num* ⟶ renvoie le type de jointure courant des lignes

num **setmiterlimit** – ⟶ définit la limite de longueur d'intersection

– **currentmiterlimit** *num* ⟶ renvoie la valeur courante du paramètre de limite d'extrémité de ligne

bool **setstrokeadjust** – ⟶ définit l'ajustement du tracé (*false* = désactivé, *true* = activé)

– **currentstrokeadjust** *bool* ⟶ renvoie l'ajustement de tracé courant

array offset **setdash** – ⟶ définit le pointillé pour le tracé

– **currentdash** *array offset* ⟶ renvoie le pointillé de tracé courant

array **setcolorspace** – ⟶ définit l'espace de couleur

– **currentcolorspace** *array* ⟶ renvoie l'espace de couleur courant

$comp_1 \ldots comp_n$ **setcolor** – ⟶ définit les composantes de couleur

– **currentcolor** $comp_1 \ldots comp_n$ ⟶ renvoie les composantes courantes de couleur

num **setgray** – ⟶ définit l'espace de couleur pour **DeviceGray** et la couleur de la valeur de gris spécifiée (0 = noir, 1 = blanc)

– **currentgray** *num* ⟶ renvoie la valeur courante en tant que valeur de gris

hue sat brt **sethsbcolor** – ⟶ définit **DeviceRGB** comme espace de couleur et la couleur à la teinte, saturation et luminosité indiquée

– **currenthsbcolor** *hue sat brt* ⟶ retourne la couleur courant en teinte, saturation et luminosité

red green blue **setrgbcolor** – ⟶ définit **DeviceRGB** comme espace de couleur et la couleur aux rouge, vert et bleu indiqués

– **currentrgbcolor** *red green blue* ⟶ renvoie la couleur courante selon ses composantes rouge, vert, bleu

cyan magenta yellow black **setcmykcolor** – ⟶ définit **DeviceCYMK** comme espace de couleur et la couleur selon le cyan, jaune, magenta et noir indiqués

– **currentcmykcolor** *cyan magenta yellow black* ⟶ renvoie la couleur courante selon les cyan, jaune, magenta et noir

14. Opérateurs de de l'état graphique - Dépendants du périphérique

dict **sethalftone** – ⟶ définit le dictionnaire de simili

– **currenthalftone** *dict* ⟶ renvoie le dictionnaire de simili courant

frequency angle proc **setscreen** – ⟶ définit l'écran de simili de gris

– **currentscreen** *frequency angle proc* ⟶ renvoie l'écran de simili de gris courant

redfreq redang redproc greenfreq greenang greenproc bluefreq blueang blueproc grayfreq grayang grayproc **setcolorscreen** – ⟶ définit les quatres écrans de simili

– **currentcolorscreen** *redfreq redang redproc greenfreq greenang greenproc bluefreq blueang blueproc grayfreq grayang grayproc* ⟶ renvoie les quatres écrans de simili

proc **settransfer** – ⟶ définit la fonction de transfert de gris

– **currenttransfer** *proc* ⟶ renvoie la fonction de transfert de gris

redproc greenproc blueproc grayproc **setcolortransfer** – ⟶ définit les quatre fonctions de transfert

– **currentcolortransfer** *redproc greenproc blueproc grayproc* ⟶ renvoie les fonctions de transfert courantes

proc **setblackgeneration** – ⟶ définit la fonction de génération de noir courante

– **currentblackgeneration** *proc* ⟶ renvoie la fonction de génération de noir courante

proc **setundercolorremoval** – ⟶ définit la fonction d'enlèvement de couleur sous-jacente

– **currentundercolorremoval** *proc* ⟶ renvoie la fonction d'enlèvement de couleur sous-jacente courante

dict **setcolorrenderring** – ⟶ définit le dictionnaire de rendu des couleurs basées sur le CIE

– **currentcolorrenderring** *dict* ⟶ renvoie le dictionnaire courant de rendu des couleurs basées sur le CIE

num **setflat** – ⟶ définit la tolérance de l'arrondi des courbes

– **currentflat** *num* ⟶ renvoie la tolérance de l'arrondi des courbes

bool **setoverprint** – ⟶ définit le paramètre de sur-impression

– **currentoverprint** *bool* ⟶ renvoie le paramètre de sur-impression courant

15. Opérateurs de système de coordonnées et de matrice

– **matrix** *matrix* ⟶ crée une matrice identité

– **initmatrix** – ⟶ définit la CTM pour le périphérique par défaut

matrix **identmatrix** *matrix* ⟶ remplit *matrix* avec la transformation identité

matrix **defaultmatrix** *matrix* ⟶ remplit *matrix* avec la matrice par défaut du périphérique

matrix **currentmatrix** *matrix* ⟶ remplit *matrix* avec la CTM

matrix **setmatrix** – ⟶ remplace la CTM ar *matrix*

t_x t_y **translate** – ⟶ déplace l'espace utilisateur par (t_x, t_y)

t_x t_y *matrix* **translate** *matrix* ⟶ définit le déplacement par (t_x, t_y)

S_x S_y **scale** – ⟶ met à l'échelle l'espace utilisateur par s_x et s_y

S_x S_y *matrix* **scale** *matrix* ⟶ définit la mise à l'échelle par s_x et s_y

angle **rotate** – ⟶ effectue une rotation de l'espace utilisateur de *angle* degrés

angle matrix **rotate** *matrix* ⟶ définit la rotation par *angle* degrés
matrix **concat** – ⟶ remplace la CTM par le produit *matrix* × CTM
$matrix_1$ $matrix_2$ $matrix_3$ **concatmatrix** $matrix_3$ ⟶ remplit $matrix_3$ avec le produit $matrix_1 \times matrix_2$
x y **transform** x' y' ⟶ transforme (x, y)*parCTM*
x y matrix **transform** x' y' ⟶ transform (x, y) par *matrix*
dx dy **dtransform** dx' dy' ⟶ transforme la distance (dx, dy) par CTM
dx dy matrix **dtransform** dx' dy' ⟶ transforme la distance (dx, dy) par *matrix*
x' y' **itransform** *x y* ⟶ inverse la transformation (x', y') par CTM
x' y' *matrix* **itransform** *x y* ⟶ inverse la transformation (x', y') par *matrix*
dx' dy' **idtransform** *dx dy* ⟶ inverse la transformation de la distance (dx', dy') par CTM
dx' dy' *matrix* **idtransform** *dx dy* ⟶ inverse la transformation de la distance (dx', dy') par *matrix*
$matrix_1$ $matrix_2$ **invertmatrix** $matrix_2$ ⟶ remplit $matrix_2$ avec l'inverse de $matrix_1$

16. Opérateurs de construction de chemin

– **newpath** – ⟶ initialise et vide le chemin courant
– **currentpoint** *x y* ⟶ renvoie les coordonnées du point courant
x y **moveto** – ⟶ définit le point courant à (x, y)
dx dy **rmoveto** – ⟶ **moveto** relatif
x y **lineto** – ⟶ ajoute une ligne droite jusqu'en (x, y)
dx dy **rlineto** – ⟶ **lineto** relatif
x y r ang_1 ang_2 **arc** – ⟶ ajoute un arc dans le sens contraire des aiguilles d'une montre
x y r ang_1 ang_2 **arcn** – ⟶ ajoute un arc dans le sens des aiguilles d'une montre
x_1 y_1 x_2 y_2 *r* **arct** *r* ⟶ ajoute un arc tangent
x_1 y_1 x_2 y_2 *r* **arcto** xt_1 yt_1 xt_2 yt_2 ⟶ ajoute un arc tangent
x_1 y_1 x_2 y_2 x_3 y_3 **curveto** – ⟶ ajoute une section cubique de Bézier
dx_1 dy_1 dx_2 dy_2 dx_3 dy_3 **rcurveto** – ⟶ **curveto** relatif
– **closepath** – ⟶ connecte le sous-chemin à son point de départ
– **flattenpath** – ⟶ convertit les courbes en suites de segments de droites
– **reversepath** – ⟶ renverse la direction du chemin courant
– **strokepath** – ⟶ calcul le contour du chemin tracé
userpath **ustrokepath** – ⟶ calcul le contour du *userpath* tracé
userpath matrix **ustrokepath** – ⟶ calcul le contour du *userpath* tracé
string bool **charpath** – ⟶ ajoute un contour de caractère au chemin courant
userpath **uappend** – ⟶ interprète *userpath* et l'ajoute au chemin courant
– **clippath** – ⟶ définit le chemin d'incrustation en tant que chemin courant
ll_x ll_y ur_x ur_y **setbbox** – ⟶ définit le cadre de limite pour le chemin courant
– **pathbbox** ll_x ll_y ur_x ur_y ⟶ renvoie le cadre de limite pour le chemin courant
move line curve close **pathforall** – ⟶ détaille le chemin courant
bool **upath** *userpath* ⟶ crée *userpath* pour le chemin courant ; inclut **ucache** si *bool* est *true*
– **initclip** – ⟶ définit le chemin d'incrustation au périphérique par défaut
– **clip** – ⟶ incruste en utilisant la règle du nombre sinueux différent de zéro
– **eoclip** – ⟶ incruste en utilisant la règle du pair-impair
x y width height **rectclip** – ⟶ incruste avec un chemin rectangulaire
numarray/*numstring* **rectclip** – ⟶ incruste avec des chemins rectangulaires
– **ucache** – ⟶ déclare que le chemin utilisateur doît être mis en cache

17. Opérateurs de dessin

– **erasepage** – ⟶ dessine la page courante en blanc
– **fill** – ⟶ remplit le chemin courant avec la couleur courante
– **eofill** – ⟶ remplit en utilisant la règle pair-impair

– **stroke** – ⟶ dessine la ligne le long du chemin courant

userpath **ufill** – ⟶ interprète et remplit *userpath*

userpath **ueofill** – ⟶ remplit *userpath* en utilisant la règle pair-impir

userpath **ustroke** – ⟶ interprète et trace *userpath*

userpath matrix **ustroke** – ⟶ interprète *userpath*, concatène *matrix* et trace

x y width height **rectfill** – ⟶ remplit le chemin rectangulaire

numarray/*numstring* **rectfill** – ⟶ remplit les chemins rectangulaires

x y width height **rectstroke** – ⟶ trace le chemin rectangulaire

numarray/*numstring* **rectstroke** – ⟶ trace les chemins rectangulaires

dict **image** – ⟶ dessine une image numérisée

widthheightbits/*sampmatrixdatasrc* **image** – ⟶ dessine une image numérisée monochrome

18. Opérateurs de test de position à l'intérieur du chemin

x y **infill** *bool* ⟶ teste si le point (x, y) est dessiné par **fill**

userpath **infill** *bool* ⟶ teste si les pixels dans *userpath* sont dessinés par **fill**

x y **ineofill** *bool* ⟶ teste si le point (x, y) est dessiné par **eofill**

userpath **ineofill** *bool* ⟶ teste si les pixels dans *userpath* sont dessinés par **eofill**

x y userpath **inufill** *bool* ⟶ teste si le point (x, y) est dessiné par **ufill** de *userpath*

$userpath_1$ $userpath_2$ **inufill** *bool* ⟶ teste si les pixels dans $userpath_1$ sont dessinés par **ufill** dans $userpath_2$

x y userpath **inueofill** *bool* ⟶ teste si le point (x, y) est dessiné par **eofill** de *userpath*

$userpath_1$ $userpath_2$ **inueofill** *bool* ⟶ teste si les pixels dans $userpath_1$ sont dessinés par **ueofill** dans $userpath_2$

x y **instroke** *bool* ⟶ teste si le point (x, y) est dessiné par **stroke**

x y userpath **instroke** *bool* ⟶ teste si le point (x, y) est dessiné par **ustroke** de *userpath*

x y userpath matrix **inustroke** *bool* ⟶ teste si le point (x, y) est dessiné par **ustroke** de *userpath*

$userpath_1$ $userpath_2$ **inustroke** *bool* ⟶ teste si les pixels dans $userpath_1$ sont dessinés par **ustroke** de $userpath_2$

$userpath_1$ $userpath_2$ *matrix* **inustroke** *bool* ⟶ teste si les pixels dans $userpath_1$ sont dessinés par **ustroke** de $userpath_2$

19. Opérateurs de formes et de motifs

pattern matrix **makepattern** *pattern* ⟶ crée une instance d'un motif à partir d'un prototype

$comp_1 \ldots comp_n$ *pattern* **setpattern** – ⟶ installe *pattern* en tant que couleur courante

$form$ **execform** – ⟶ dessin $form$

20. Opérateurs de configuration et de sortie du périphérique

– **showpage** – ⟶ transmet et réinitialise la page courante

– **copypage** – ⟶ transmet la page courante

dict **setpagedevice** – ⟶ installe le périphérique de sortie orienté page

– **currentpagedevice** *dict* ⟶ renvoie les paramètres du périphérique courant de sortie de page

– **nulldevice** – ⟶ installe un périphérique sans sortie

21. Opérateurs de caractères et de polices

key $font$ **definefont** $font$ ⟶ enregistre $font$ comme dictionnaire de police

key **undefinedfont** – ⟶ supprime l'identification de police

key **findfont** $font$ ⟶ renvoie le dictionnaire de police identifié par *key*

$font$ *scale* **scalefont** $font'$ ⟶ met à l'échelle $font$ par *sclae* pour produire la nouvelle $font'$

$font$ *matrix* **makefont** $font'$ ⟶ transforme $font$ par *matrix* pour produire la nouvelle $font'$

$font$ **setfont** – ⟶ définit le dictionnaire de police dans l'état graphique

– **currentfont** $font$ ⟶ renvoie le dictionnaire de police courant

– **rootfont** $font$ ⟶ renvoie le dictionnaire racine d'une police composite

key scale/*matrix* **selectfont** – ⟶ définit le dictionnaire de police par son nom et le transforme

string **show** – ⟶ déssine les caractères de *string* sur la page

a_x a_y *string* **ashow** – ⟶ ajoute (a_x, a_y) à la largeur de tout caractère tout en montrant *string*

c_x c_y *char string* **widthshow** – ⟶ ajoute (c_x, c_y) à la largeur de *char* tout en montrant *string*

c_x c_y *char* a_x a_y *string* **awidthshow** – ⟶ combine les effets de **ashow** et de **widthshow**

string numarray/*numstring* **xshow** – ⟶ dessine les caractères de *string* en utilisant les largeurs *x* dans *numarray*/*numstring*

string numarray/*numstring* **xyshow** – ⟶ dessine les caractères de *string* en utilisant les largeurs *x* et *y* dans *numarray*/*numstring*

string numarray/*numstring* **yshow** – ⟶ dessine les caractères de *string* en utilisant les largeurs *y* dans *numarray*/*numstring*

name **glypshow** – ⟶ dessine les caractères identifiés par *name*

string **stringwidth** w_x w_y ⟶ largeur de *string* dans la police courante

proc string **cshow** – ⟶ appelle l'algorithme d'affichage et appelle *proc*

proc string **kshow** – ⟶ exécute *proc* entre les caractères montrés depuis *string*

– **FontDirectory** *dict* ⟶ dictionnaire des dictionnaires de police

– **GlobalFontDirectory** *dict* ⟶ dictionnaire des dictionnaires de police dans la VM globale

– **StandardEncoding** *array* ⟶ vecteur d'encodage standard des polices Adobe

– **ISOLatin1Encoding** *array* ⟶ vecteur d'encodage international ISO Latin-1 des polices

key **findencoding** *array* ⟶ trouve le tableau d'encodage

w_x w_y ll_x ll_y ur_x ur_y **setcachedevice** – ⟶ déclare les mesures des caractères en cache

$w0_x$ $w0_y$ ll_x ll_y ur_x ur_y $w1_x$ $w1_y$ ll_x ll_y v_x v_y **setcachedevice2** – ⟶ déclare les mesures des caractères en cache

w_x w_y **setcharwidth** – ⟶ déclare les mesures des caractères non mis en cache

22. Opérateurs de paramétrage de l'interpréteur

dict **setsystemparams** – ⟶ définit les paramètres de système pour l'interpréteur

– **currentsystemparams** *dict* ⟶ renvoie les paramètres de système pour l'interpréteur

dict **setusersparams** – ⟶ établit les paramètres de l'interpréteur par contexte

– **currentusersparams** *dict* ⟶ renvoie les paramètres de l'interpréteur par contexte

string dict **setdevparams** – ⟶ définit les paramètres pour le périphérique d'entrée-sortie

string **currentdevarams** *dict* ⟶ renvoie les paramètres du périphérique

int **vmreclaim** – ⟶ contrôle le ramassage des poubelles

int **setvmhreshold** – ⟶ contrôle le ramassage des poubelles

– **vmstatus** *level used maximum* ⟶ rapport sur le statut de la VM

– **cachestatus** *bsize bmax msize mmax csize cmax blimit* ⟶ renvoie le statut du cache de police et ses paramètres

num **setcachelimit** – ⟶ définit le nombre maximum d'octets pour les caractères mis en cache

mark size lower upper **setcacheparams** – ⟶ modofie les paramètres du cache de police

– **currentcacheparams** *mark size lower upper* ⟶ renvoie les paramètres courant du cache de police

mark blimit **setucacheparams** – ⟶ définit les paramètres du cache de chemin utilisateur

– **ucachestatus** *mark bsize bmax rsize rmax blimit* ⟶ renvoie le statut du cache de chemin utilisateur et les paramètres

23. Opérateurs Display PostScript

– **currentcontext** *context* ⟶ renvoie l'identificateur du contexte courant

mark obj_1 ... obj_n *proc* **fork** *context* ⟶ crée un contexte exécutant *proc* avec obj_1 ... obj_n comme opérandes

context **join** *mark* obj_1 ... obj_n ⟶ attend la fin d'un contexte et renvoie son résultat

context **detach** – ⟶ permet à un contexte de se terminer immédiatement lorsqu'il est fini

– **lock** *lock* ⟶ crée un objet verrou

lock proc **monitor** – ⟶ exécute *proc* tout en gardant *lock*

– **condition** *condition* ⟶ crée un objet condition

lock condition **wait** – ⟶ relâche *lock*, attend *condition*, reprend *lock*

condition **notify** – ⟶ reprend le contexte en attendant *condition*

– **yield** – ⟶ met momentanément en attente dans le contexte courant

index name **defineusername** – ⟶ définit un index de nom encodé

– **viewclip** – ⟶ définit la vue d'incrustation depuis le chemin courant

– **eoviewclip** – ⟶ définit la vue d'incrustation en utilisant la règle pair-impair

x y width height **rectviewclip** – ⟶ définit un chemin rectangulaire de vue d'incrustation

numarray/numstring **rectviewclip** – ⟶ définit des chemins rectangulaires de vue d'incrustation

– **initviewclip** – ⟶ réinitialise la vue d'incrustation

– **viewclippath** – ⟶ définit le chemin courant à partir de la vue d'incrustation

– **deviceinfo** *dict* ⟶ renvoie le dictionnaire contenant les informations à propos du périphérique courant

– **wtranslation** *x y* ⟶ renvoie la traduction à partir de l'origine de la fenêtre vers l'origine de l'espace du périphérique

x y **sethalftonephase** – ⟶ définit la phase de simili

– **currenthalftonephase** *x y* ⟶ renvoie la phase de simili courante

24. Erreurs

configurationerror ⟶ une demande **setpagedevice** ne peut être satisfaite

dictfull ⟶ plus de place dans le dictionnaire

dictstackoverflow ⟶ trop de **begin**

dictstackunderflow ⟶ trop de **end**

execstackoverflow ⟶ imbrication trop nombreuse d'**exec**

handleerror ⟶ appelé pour donner un rapport sur les erreurs

interrupt ⟶ demande extérieure d'interruption

invalidaccess ⟶ essai de violation des attributs d'accès

invalidcontext ⟶ utilisation impropre de l'opération concernant le contexte

invalidexit ⟶ **exit** n'est pas dans la boucle

invalidfileaccess ⟶ chaîne d'accès onn correcte

invalidfont ⟶ nom de police ou de dictionnaire incorrect

invalid ⟶ identificateur invalide pour un objet externe

invalidrestore ⟶ **restore** incorrect

ioerror ⟶ erreur d'entrée/sortie

limitcheck ⟶ dépassement des limites de l'implémentation

nocurrentpoint ⟶ le point courant n'est pas défini

rangecheck ⟶ opérande en dehors des limites

stackoverflow ⟶ dépassement de capacité de la pile d'opérandes

stackunderflow ⟶ pas assez d'opérandes dans la pile

syntaxerror ⟶ erreur de syntaxe en PostScript

timeout ⟶ dépassement de la limite de temps

typecheck ⟶ opérande de mauvais type

undefined ⟶ nom inconnu

undefinedfilename ⟶ fichier non trouvé

undefinedsource ⟶ instance de resssource non trouvée

undefinedresult ⟶ résultat insuffisant, dépassant ou sans signification

undefinedmark ⟶ marque attendue absente de la pile

unregistered ⟶ erreur interne

VMerror ⟶ VM épuisée

Annexe II : Index des commandes jps

D

3dto2d

x y z **3dto2d** *X Y* ⟶ calcule les coordonnées du point projeté sur l'écran pour la représentation 3d. cette commande est synonyme de la commande **CamView**

3dto2d

x y z **3dto2d** *X Y* ⟶ calcule les coordonnées du point projeté sur l'écran pour la représentation 3d. cette commande est synonyme de la commande **CamView**

A

ABcercle

A B C **ABcercle** *cerc* ⟶ *cerc* est le cercle passant par les points *A*, *B* et *C*

ABCercle

I A B **ABCercle** – ⟶ trace l'arc du cercle $\mathcal{C}$ inscrit dans l'angle $\widehat{AIB}$ où $\mathcal{C}$ désigne le cercle de centre *I* passant par *A*

ABCercle*

I A B **ABCercle*** – ⟶ version étoilée de **ABCercle**

ABCercle_

I A B **ABCercle_** – ⟶ version underscore de **ABCercle**

ABpoint

α *A B* **ABpoint** *A′* ⟶ le point *A′* est l'image du point *B* par l'homothétie de centre *A*, de rapport α. Autrement dit $\overrightarrow{AA'} = \alpha\overrightarrow{AB}$

abs

a **abs** *c* ⟶ $c = |a|$

addc

z z′ **addc** *Z* ⟶ $Z = z + z'$ est la somme des complexes *z* et *z′*

add

a b **add** *c* ⟶ $c = a + c$

add

nombre$_1$ *nombre*$_2$ **add** *nombre*$_3$ ⟶ additionne *nombre*$_1$ et *nombre*$_2$

addm

A B **addm** *M* ⟶ additionne les matrices *A* et *B* et dépose le résultat sur la pile

addv3d

$\vec{u}\ \vec{v}$ **addv3d** $\vec{w}$ ⟶ $\vec{w} = \vec{u} + \vec{v}$

addv3d

$\vec{u}\ \vec{v}$ **addv3d** $\vec{w}$ ⟶ $\vec{w} = \vec{u} + \vec{v}$

addv

u u′ **addv** $\vec{U}$ ⟶ $\vec{U} = \vec{u} + \vec{u}'$ est la somme des vecteurs $\vec{u}$ et $\vec{u}'$

ahangle

ahangle ⟶ angle au sommet des flèches de fin de ligne. **valeur par défaut :** 30

ahcoeff

ahcoeff ⟶ coefficient entre 0 et 1 indiquant le niveau de décrochement sur la base de la flèche. **valeur par défaut :** 1

ahlength

ahlength ⟶ longueur en picas des flèches de fin de ligne. **valeur par défaut :** 6

angleA

angleA ⟶ certaines connexions vous permettent de spécifier avec quel angle vous souhaitez la connexion au nœud. **valeur par défaut :** 0

angleB

angleB ⟶ certaines connexions vous permettent de spécifier avec quel angle vous souhaitez la connexion au nœud. **valeur par défaut :** 0

angledroit

A B C **angledroit** – ⟶ dessine un angle droit en *B*

angle

A B **angle** $\alpha \longrightarrow \alpha$ est l'angle en dégré défini par le vecteur $\overrightarrow{AB}$ dans le repère **orthonormé** jps.

Anp

n p **Anp** $a \longrightarrow a = A_n^p = n \times (n-1) \times \cdots \times (n-p+1)$

apply

[$a_0 \ldots a_n$] f **apply** [$b_0 \ldots b_n$] ou $- \longrightarrow$ construit un nouveau tableau en répétant l'opération suivante : déposer l'élément a_i puis exécuter f, pour i variant de 0 à n. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

apply

[$a_0 \ldots a_n$] f **apply** [$b_0 \ldots b_n$] ou $- \longrightarrow$ construit un nouveau tableau en répétant, pour i variant de 0 à n, l'opération suivante : déposer l'élément a_i puis exécuter f. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

Apply

[$a_0 \ldots a_n$] f n_0 n_1 **Apply** [$b_0 \ldots b_n$] ou $- \longrightarrow$ L'éxécutable f prend n_1 arguments et on décale de n_0 à chaque itération. Ainsi le premier paramètre de la 2ème itération est x_{n_0+1}. Par exemple, les commandes **3 3 Apply** et **capply** sont équivalentes.

apply

[$a_0 \ldots a_n$] *string* **apply** [$b_0 \ldots b_n$] ou $- \longrightarrow$ Comme la précédente, mais l'exécutable est cette fois désigné par une chaîne de caractères.

arcangleA

arcangleA $\longrightarrow$ ce paramètre ne sert qu'avec les commandes **ncarc** et **pcarc**. **valeur par défaut :** 10

arcangleB

arcangleB $\longrightarrow$ ce paramètre ne sert qu'avec les commandes **ncarc** et **pcarc**. **valeur par défaut :** 10

arccos

a **arccos** $c \longrightarrow c = \text{Arccos}\, a$ (en degrés)

Arccos

a **Arccos** $c \longrightarrow c = \text{Arccos}\, a$ (en radians)

Arc

I A α **Arc** $- \longrightarrow$ trace au point *A* l'arc de cercle d'angle 2α centré en *A*

arc

x y r ang_1 ang_2 **arc** $- \longrightarrow$ ajoute un arc dans le sens contraire des aiguilles d'une montre

arcn

x y r ang_1 ang_2 **arcn** $- \longrightarrow$ ajoute un arc dans le sens des aiguilles d'une montre

arcnp

I A B **arcnp** $- \longrightarrow$ trace entre les points *A* et *B* l'arc du cercle de centre *I* et de rayon *IA* (sens inverse du sens trigonométrique)

arcp

I A B **arcp** $- \longrightarrow$ trace entre les points *A* et *B* l'arc du cercle de centre *I* et de rayon *IA* (sens trigonométrique)

arcsin

a **arcsin** $c \longrightarrow c = \text{Arcsin}\, a$ (en degrés)

Arcsin

a **Arcsin** $c \longrightarrow c = \text{Arcsin}\, a$ (en radians)

arcspherique

r θ_1 ϕ_1 r θ_2 ϕ_2 **arcspherique** $- \longrightarrow$ trace l'arc de cercle entre les points *A* et *B* de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

arcspherique

r θ_1 ϕ_1 r θ_2 ϕ_2 **arcspherique** $- \longrightarrow$ trace l'arc de cercle entre les points *A* et *B* de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

arctan

a **arctan** $c \longrightarrow c = \text{Arctan}\, a$ (en degrés)

Arctan

a **Arctan** $c \longrightarrow c = \text{Arctan}\, a$ (en radians)

arct

$x_1\ y_1\ x_2\ y_2\ r$ **arct** $r \longrightarrow$ ajoute un arc tangent

arcto

$x_1\ y_1\ x_2\ y_2\ r$ **arcto** $xt_1\ yt_1\ xt_2\ yt_2 \longrightarrow$ ajoute un arc tangent

aretescachees

aretescachees $\longrightarrow$ booléen indiquant à la procédure **drawsolid** si l'on doit ou non représenter les arêtes cachées. **valeur par défaut :** *true*

argcosh

a **argcosh** $c \longrightarrow c = \text{Arg ch}\, a$

arg

u **arg** $\theta \longrightarrow \theta \in]-180, 180]$ est l'angle que fait le vecteur $\vec{u}$ avec le vecteur unitaire de l'axe des abscisses

arg

z **arg** $\theta \longrightarrow \theta = \text{Arg}(z) \in]-180, 180]$

argsinh

a **argsinh** $c \longrightarrow c = \text{Arg sh}\, a$

argtanh

a **argtanh** $c \longrightarrow c = \text{Arg th}\, a$

armA

armA $\longrightarrow$ certaines connexions commencent avec un bras de longueur *armA* (en picas). **valeur par défaut :** 10

armB

armB $\longrightarrow$ certaines connexions commencent avec un bras de longueur *armB* (en picas). **valeur par défaut :** 10

arrowangle

arrowangle $\longrightarrow$ angle en degrés de la rotation que doit subir la flèche avant d'être tracée par la commande **arrow**. Ce paramètre permet d'ajuster l'angulation lorsque les calculs automatiques laissent à désirer. **valeur par défaut :** 0

arrowpath0

– **arrowpath0** *pathobj* $\longrightarrow$ chemin correspondant à l'axe d'une flèche potentielle au début du chemin courant lors du dernier appel à **arrowpaths**

arrowpath1

– **arrowpath1** *pathobj* $\longrightarrow$ chemin correspondant à l'axe d'une flèche potentielle à la fin du chemin courant lors du dernier appel à **arrowpaths**

arrowpaths

– **arrowpaths** $pathobj_1\ pathobj_2 \longrightarrow$ depose sur la pile les 2 sous chemins en provenance du chemin courant necessaires à **gere_arrowhead**

arrowscale

arrowscale $\longrightarrow$ exécutable donnant les facteurs d'échelles horizontale et verticale pour la tracé d'une flèche par **arrow**. **valeur par défaut :** { 1 1 }

axeB

zmin zmax ℓ **axeB** $- \longrightarrow [zmin; zmax]$ = étendue du pointille, ℓ = longueur du vecteur

axeOxarrow

– **axeOxarrow** $- \longrightarrow$ trace la flèche au bout de l'axe *Ox*

axeOyarrow

– **axeOyarrow** $- \longrightarrow$ trace la flèche au bout de l'axe *Oy*

axeR

xmin xmax ℓ **axeR** $- \longrightarrow [xmin; xmax]$ = étendue du pointille, ℓ = longueur du vecteur

axesarrow

– **axesarrow** $- \longrightarrow$ trace les flèches au bout des axes *Ox* et *Oy*

axesRVB

min max ℓ **axesRVB** $- \longrightarrow [min; max]$ = étendue des pointillés, ℓ = longueur des vecteurs

axesymcercle

cerc D **axesymcercle** *cerc′* $\longrightarrow$ le cercle *cerc′* est la symétrique du cercle *cerc* par rapport à la droite *D*

axesymcpath

cpathobj$_1$ *D* **axesymcpath** *cpathobj*$_2$ $\longrightarrow$ *cpathobj*$_2$ est le chemin continu image de *cpathobj*$_1$ par la symétrie axiale d'axe *D*

axesymdroite

d D **axesymdroite** *d′* $\longrightarrow$ la droite *d′* est la symétrique de la droite *d* par rapport à la droite *D*

axesymell

ell D **axesymell** *ell′* $\longrightarrow$ l'ellipse *ell′* est la symétrique de l'ellipse *ell* par rapport à la droite *D*

axesympath

pathobj$_1$ *D* **axesympath** *pathobj*$_2$ $\longrightarrow$ *pathobj*$_2$ est le chemin image de *pathobj*$_1$ par la symétrie axiale d'axe *D*

axesympoint

A D **axesympoint** *A′* $\longrightarrow$ le point *A′* est le symétrique du point *A* par rapport à la droite *D*

axesympol

pol D **axesympol** *pol′* $\longrightarrow$ le polygône *pol′* est la symétrique du polygône *pol* par rapport à la droite *D*

axeV

ymin ymax ℓ **axeV** $-$ $\longrightarrow$ [*ymin*; *ymax*] = étendue du pointille, ℓ = longueur du vecteur

B

barycentre3d

[*A a B b*] **barycentre3d** *G* $\longrightarrow$ le point *G* est le barycentre du système $[(A,a);(B,b)]$

barycentre3d

[*A a B b*] **barycentre3d** *G* $\longrightarrow$ le point *G* est le barycentre du système $[(A,a);(B,b)]$

baseeuler

a {*f*} x_0 y_0 *h* **baseeuler** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Plus précisement : $y_{i+1} = y_i + hy'_i$ où $y'_i = f(x_i,y_i)$, et $x_n = a$. Attention : on peut avoir $a < x0$, mais dans ce cas *h* doit être négatif

baseeuler

a {*f*} x_0 y_0 *n* **baseeuler** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ *n* étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas *h* est calculé en fonction de l'entier *n*.

baseeulermod

a {*f*} x_0 y_0 *h* **baseeulermod** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas *h* doit être négatif

baseeulermod

a {*f*} x_0 y_0 *n* **baseeulermod** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ *n* étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas *h* est calculé en fonction de l'entier *n*.

basemilne

a {*f*} x_0 y_0 *h* **basemilne** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas *h* doit être négatif

basemilne

a {*f*} x_0 y_0 *n* **basemilne** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ *n* étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le pas *h* est calculé en fonction de l'entier *n*.

baserungekutta

a {*f*} x_0 y_0 *h* **baserungekutta** x_0 y_0 x_1 y_1 ... x_n y_n $\longrightarrow$ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[x_0,a]$ de la fonction *s* solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Attention : on peut avoir $a < x0$, mais dans ce cas *h* doit être négatif

baserungekutta

$a\ \{f\}\ x_0\ y_0\ n$ **baserungekutta** $x_0\ y_0\ x_1\ y_1\ \ldots x_n\ y_n \longrightarrow n$ étant un entier, cette procédure dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[x_0, a]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le pas h est calculé en fonction de l'entier n.

baton

A **baton** $- \longrightarrow$ dessine un baton parallèle à l'axe Oy, dont une extrémité est le point A, et dont l'autre est sur l'axe Ox

bbpict

A [*xscale yscale*] $\{\alpha\}$ *string* **bbpict** $- \longrightarrow$ Se place au point A, puis affiche l'objet désigné par la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

bbtexlabel3d

bbtexlabel3d $- \longrightarrow$ Analogue 3d de la commande **bbtexlabel**

bbtexlabel

A [*xscale yscale*] $\{\alpha\}$ **bbtexlabel** $- \longrightarrow$ Se place au point A, puis dessine le label TEX au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

bbtext3d

bbtext3d $- \longrightarrow$ Analogue 3d de la commande **bbtext**

bbtext

string A [*xscale yscale*] $\{\alpha\}$ **bbtext** $- \longrightarrow$ Se place au point A, puis affiche la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

bcpict

A [*xscale yscale*] $\{\alpha\}$ *string* **bcpict** $- \longrightarrow$ Se place au point A, puis affiche l'objet désigné par la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

bctexlabel3d

bctexlabel3d $- \longrightarrow$ Analogue 3d de la commande **bctexlabel**

bctexlabel

A [*xscale yscale*] $\{\alpha\}$ **bctexlabel** $- \longrightarrow$ Se place au point A, puis dessine le label TEX au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

bctext3d

bctext3d $- \longrightarrow$ Analogue 3d de la commande **bctext**

bctext

string A [*xscale yscale*] $\{\alpha\}$ **bctext** $- \longrightarrow$ Se place au point A, puis affiche la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

Bessel_j0

x **Bessel_j0** $y \longrightarrow y$ est l'image de x par la fonction J_0 de Bessel

Bessel_j1

x **Bessel_j1** $y \longrightarrow y$ est l'image de x par la fonction J_1 de Bessel

Bessel_y0

x **Bessel_y0** $y \longrightarrow y$ est l'image de x par la fonction Y_0 de Bessel

Bessel_y1

x **Bessel_y1** $y \longrightarrow y$ est l'image de x par la fonction Y_1 de Bessel

bezier_curve_

array **bezier_curve_** $- \longrightarrow$ ajoute au chemin courant la courbe de Bézier spécifiée par le tableau de points *array*

bezier_curve

array **bezier_curve** $- \longrightarrow$ trace la courbe de Bézier spécifiée par le tableau de points *array*

binomiale

k n p **binomiale** $a \longrightarrow a = C_n^k p^k (1-p)^{n-k}$

bissectrice

A B C **bissectrice** *D* $\longrightarrow$ *D* est la bissectrice de l'angle $\widehat{ABC}$

bissectrice

A B C **bissectrice** *D* $\longrightarrow$ la droite *D* bissectrice de l'angle $\widehat{ABC}$

blanc

– **blanc** – $\longrightarrow$ sélectionne la couleur blanc

bleu

– **bleu** – $\longrightarrow$ sélectionne la couleur bleu

blpict

A [*xscale yscale*] {α} *string* **blpict** – $\longrightarrow$ Se place à aguche du point *A*, puis affiche l'objet désigné par la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

bltexlabel3d

bltexlabel3d – $\longrightarrow$ Analogue 3d de la commande **bltexlabel**

bltexlabel

A [*xscale yscale*] {α} **bltexlabel** – $\longrightarrow$ Se place à aguche du point *A*, puis dessine le label TEX au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

bltext3d

bltext3d – $\longrightarrow$ Analogue 3d de la commande **bltext**

bltext

string A [*xscale yscale*] {α} **bltext** – $\longrightarrow$ Se place à aguche du point *A*, puis affiche la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

bnode

string **bnode** – $\longrightarrow$ Déclare un nœud rectangulaire encadré dont le nom est défini par *string*

boxit_all

– **boxit_all** – $\longrightarrow$ sélectionne l'encadrement systématique des objets affichés par l'environnement 'picture'. En particulier, la gestion de l'affichage du texte ou des labels TEX est concernée

boxit

– **boxit** – $\longrightarrow$ sélectionne l'encadrement du prochain objet affiché par l'environnement 'picture'. En particulier, la gestion de l'affichage du texte ou des labels TEX est concernée

boxit_none

– **boxit_none** – $\longrightarrow$ désélectionne l'encadrement des prochains objets affichés par l'environnement 'picture'.

brpict

A [*xscale yscale*] {α} *string* **brpict** – $\longrightarrow$ Se place à droite du point *A*, puis affiche l'objet désigné par la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

brpict

A [*xscale yscale*] {α} *string* **brpict** – $\longrightarrow$ Se place en bas à droite du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

brtexlabel3d

brtexlabel3d – $\longrightarrow$ Analogue 3d de la commande **brtexlabel**

brtexlabel

A [*xscale yscale*] {α} **brtexlabel** – $\longrightarrow$ Se place à droite du point *A*, puis dessine le label TEX au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

brtexlabel

A [*xscale yscale*] {α} **brtexlabel** – $\longrightarrow$ Se place en bas à droite du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont

optionnels

brtext3d

brtext3d – ⟶ Analogue 3d de la commande **brtext**

brtext

string A [*xscale yscale*] {α} **brtext** – ⟶ Se place à droite du point *A*, puis affiche la chaîne *string* au niveau de la *baseline*, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

brtext

string A [*xscale yscale*] {α} **brtext** – ⟶ Se place en bas à droite du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

bubblesort

$array_1$ **bubblesort** $array_2$ ⟶ le tableau de réels $array_2$ est le résultat du tri à bulle sur le tableau de réels $array_1$.

C **CamView**

x y z **CamView** *X Y* ⟶ On projete le point 3d sur le plan de représentation de la caméra, selon le mode de représentation

capply

[$cerc_0 \ldots cerc_n$] *f* **capply** [$b_0 \ldots b_n$] ou – ⟶ construit un nouveau tableau en répétant, pour *i* variant de 0 à *n*, l'opération suivante : déposer le cercle C_i puis exécuter *f*. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

cbpict

A [*xscale yscale*] {α} *string* **cbpict** – ⟶ Se place au point *A*, puis affiche l'objet désigné par la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cbtexlabel3d

cbtexlabel3d – ⟶ Analogue 3d de la commande **cbtexlabel**

cbtexlabel

A [*xscale yscale*] {α} **cbtexlabel** – ⟶ Se place au point *A*, puis dessine le label TEX en cours centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cbtext3d

cbtext3d – ⟶ Analogue 3d de la commande **cbtext**

cbtext

string A [*xscale yscale*] {α} **cbtext** – ⟶ Se place au point *A*, puis affiche la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ccpict

A [*xscale yscale*] {α} *string* **ccpict** – ⟶ Se place au point *A*, puis affiche l'objet désigné par la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cctexlabel3d

cctexlabel3d – ⟶ Analogue 3d de la commande **cctexlabel**

cctexlabel

A [*xscale yscale*] {α} **cctexlabel** – ⟶ Se place au point *A*, puis dessine le label TEX en cours centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cctext3d

cctext3d – ⟶ Analogue 3d de la commande **cctext**

cctext

string A [*xscale yscale*] {α} **cctext** – ⟶ Se place au point *A*, puis affiche la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cctextp3d

str solid n α *str bool* **cctextp3d** - ⟶ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine défini par la face d'indice *n* du solide *solid*. Le booléen optionnel *bool* sert à préciser si on doit ou non tenir compte de la visibilité (*true* par défaut). La chaîne de caractères optionnelle *str* permet de préciser l'origine du

plan de projection (centre de la face de projection par défaut). L'angle optionnel α permet de préciser la rotation autour de la normale souhaitée.

cctextp3d

str x_0 y_0 z_0 **[** i_1 i_2 i_3 k_1 k_2 k_3 **]** *bool* **cctextp3d** - $\longrightarrow$ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est donnée par $\vec{\imath}'(i_1, i_2, i_3)$, image du vecteur $(1, 0, 0)$ ar cette projection. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

cctextp3d

str x_0 y_0 z_0 **[** k_1 k_2 k_3 α **]** *bool* **cctextp3d** - $\longrightarrow$ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$, après avoir fait subir une rotation d'angle α degrés autour de la normale par rapport à l'orientation originellement proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

cctextp3d

str x_0 y_0 z_0 **[** k_1 k_2 k_3 **]** *bool* **cctextp3d** - $\longrightarrow$ Projette puis encre le chemin défini par la chaîne *str* sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

ceiling

num_1 **ceiling** num_2 $\longrightarrow$ plafond de num_1

Cercle_

α β *cerc* **Cercle_** – $\longrightarrow$ ajoute au chemin courant la portion de cercle allant du point de paramètre α au point de paramètre β

Cercle

α β *cerc* **Cercle** – $\longrightarrow$ trace la portion de cercle spécifiée

cercle_

cerc **cercle_** – $\longrightarrow$ ajoute au chemin courant le cercle spécifié

cercle

cerc **cercle** – $\longrightarrow$ trace le cercle spécifié

cframe_

A L ℓ **cframe_** – $\longrightarrow$ ajoute au chemin courant trace le rectangle dont le point A est le centre, de dimension horizontale L et de dimension verticale ℓ

cframe

A L ℓ **cframe** – $\longrightarrow$ trace le rectangle dont le point A est le centre, de dimension horizontale L et de dimension verticale ℓ

champvecteur

f $step_1$ $step_2$ ℓ **champvecteur** – $\longrightarrow$ Trace les vecteurs de norme ℓ définis par $y' = f(x, y)$, en partant de $(xmin, ymin)$ et jusqu'à $(xmax, ymax)$ et en tenant compte des pas $step_1$ (sur Ox) et $step_2$ (sur O_y)

circleit_all

– **circleit_all** – $\longrightarrow$ sélectionne l'encerclement systématique des objets affichés par l'environnement 'picture'. En particulier, la gestion de l'affichage du texte ou des labels TEX est concernée

circleit

– **circleit** – $\longrightarrow$ sélectionne l'encerclement du prochain objet affiché par l'environnement 'picture'. En particulier, la gestion de l'affichage du texte ou des labels TEX est concernée

circleit_none

– **circleit_none** – $\longrightarrow$ désélectionne l'encerclement des prochains objets affichés par l'environnement 'picture'.

circ

A **circ** – $\longrightarrow$ dessine un point cerclé en A dans le repère jps

closepath

– **closepath** – $\longrightarrow$ connecte le sous-chemin à son point de départ

clpict

A [*xscale yscale*] $\{\alpha\}$ *string* **clpict** – $\longrightarrow$ Se place à aguche du point A, puis affiche l'objet désigné par la chaîne *string* centré, avec l'échelle $(xscale, yscale)$ et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

cltexlabel3d

cltexlabel3d – ⟶ Analogue 3d de la commande **cltexlabel**

cltexlabel

A [*xscale yscale*] {α} **cltexlabel** – ⟶ Se place à aguche du point *A*, puis dessine le label TEX en cours centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

cltext3d

cltext3d – ⟶ Analogue 3d de la commande **cltext**

cltext

string A [*xscale yscale*] {α} **cltext** – ⟶ Se place à aguche du point *A*, puis affiche la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

Cnode

string **Cnode** – ⟶ Déclare un nœud circulaire de rayon fixe *Circleradius* dont le nom est défini par *string*

cnode

string **cnode** – ⟶ Déclare un nœud circulaire dont le nom est défini par *string*

Cnp

n p **Cnp** $c \longrightarrow c = C_n^p = A_n^p / p!$

coeffdir

D **coeffdir** $a \longrightarrow a$ est le coefficient directeur de la droite *D* si celle-ci n'est pas verticale, erreur sinon

coeff_xmarks

array string coeff **coeff_xmarks** – ⟶ Numérote l'axe *Ox* en fractions de *coeff* en utilisant *string* pour l'affichage et les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

coeff_xticks

array coeff **coeff_xticks** – ⟶ Trace des tirets sur l'axe *Ox* en fractions de *coeff* en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

coeff_ymarks

array string coeff **coeff_ymarks** – ⟶ Numérote l'axe *Ox* en fractions de *coeff* en utilisant *string* pour l'affichage et les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

coeff_yticks

array coeff **coeff_yticks** – ⟶ Trace des tirets sur l'axe *Oy* en fractions de *coeff* en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

ComputeCamera

– **ComputeCamera** – ⟶ *Compute vectors usefull to CamView.*

conjugue

z **conjugue** $\overline{z} \longrightarrow \overline{z}$ est le conjugué du complexe *z*

continu

– **continu** – ⟶ sélectionne le tracé de type *continu*

correlation

$array_1$ $array_2$ **correlation** $r \longrightarrow$ le réel *r* est la coefficient de corrélation de la série double définie par les tableaux de réels $array_1$ et $array_2$

cosh

a **cosh** $c \longrightarrow c = \operatorname{ch} a$

cos

a **cos** $c \longrightarrow c = \cos a$ (*a* en degré)

Cos

a **Cos** $c \longrightarrow c = \cos a$ (*a* en radian)

cotanh

a **cotanh** $c \longrightarrow c = \coth a$

cotan

a **cotan** $c \longrightarrow c$ = cotan *a* (*a* en degré)

coTan

a **coTan** $c \longrightarrow c$ = cotan *a* (*a* en radian)

gradientfill

couleur$_1$ *couleur*$_2$ *m* **gradientfill** – $\longrightarrow$ remplit le domaine d'incrustation en cours par un gradient de couleur passant de *couleur*$_1$ à *couleur*$_2$. Le nombre *m* appartient à [0; 1]; il indique à quel moment doit atteindre la couleur *couleur*$_2$

Courbe_

a b {*f*} **Courbe_** – $\longrightarrow$ ajoute au chemin courant la courbe représentative de la fonction f pour x allant de *a* à *b*

Courbe

a b {*f*} **Courbe** – $\longrightarrow$ trace la courbe représentative de la fonction f sur l'intervalle $[a; b]$

Courbe_

a b proc **Courbe_** – $\longrightarrow$ ajoute au chemin courant la courbe représentative pour x allant de *a* à *b* de la fonction définie par l'exécutable *proc*

Courbe

a b proc **Courbe** – $\longrightarrow$ trace la courbe représentative sur l'intervalle $[A; B]$ de la fonction définie par l'exécutable *proc*

courbe_

{*f*} **courbe_** – $\longrightarrow$ ajoute au chemin courant la courbe représentative de la fonction f sur l'intervalle [*xmin*; *xmax*]

courbe

{*f*} **courbe** – $\longrightarrow$ trace la courbe représentative de la fonction f sur l'intervalle [*xmin*; *xmax*]

courbe_

proc **courbe_** – $\longrightarrow$ ajoute au chemin courant la courbe représentative sur l'intervalle [*xmin*; *xmax*] de la fonction définie par l'exécutable *proc*

courbe

proc **courbe** – $\longrightarrow$ trace la courbe représentative sur l'intervalle [*xmin*; *xmax*] de la fonction définie par l'exécutable *proc*

Courbeparam_

a b proc$_1$ *proc*$_2$ **Courbeparam_** – $\longrightarrow$ ajoute au chemin courant la courbe paramétrée définie, pour t variant de *a* à *b*, par les exécutables *proc*$_1$ et *proc*$_2$

Courbeparam

a b proc$_1$ *proc*$_2$ **Courbeparam** – $\longrightarrow$ trace sur l'intervalle $[a; b]$ la courbe paramétrée définie par les exécutables *proc*$_1$ et *proc*$_2$

Courbeparam_

a b {*X*} {*Y*} **Courbeparam_** – $\longrightarrow$ ajoute au chemin courant la courbe paramétrée $t \mapsto (X(t); Y(t))$ pour t variant de *a* à *b*

Courbeparam

a b {*X*} {*Y*} **Courbeparam** – $\longrightarrow$ trace la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle $[a; b]$

courbeparam_

proc$_1$ *proc*$_2$ **courbeparam_** – $\longrightarrow$ ajoute au chemin courant la courbe paramétrée définie sur l'intervalle [*tmin*; *tmax*] par les exécutables *proc*$_1$ et *proc*$_2$

courbeparam

proc$_1$ *proc*$_2$ **courbeparam** – $\longrightarrow$ trace sur l'intervalle [*tmin*; *tmax*] la courbe paramétrée définie par les exécutables *proc*$_1$ et *proc*$_2$

courbeparam_

{*X*} {*Y*} **courbeparam_** – $\longrightarrow$ ajoute au chemin courant la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle [*tmin*; *tmax*]

courbeparam

{*X*} {*Y*} **courbeparam** – $\longrightarrow$ trace la courbe paramétrée $t \mapsto (X(t); Y(t))$ sur l'intervalle [*tmin*; *tmax*]

Courbepolar

a b {ρ} **Courbepolar** – ⟶ trace la courbe paramétrée $\theta \mapsto (\rho(\theta)\cos\theta; \rho(\theta)\sin\theta)$ sur l'intervalle $[a; b]$

courbepolar

{ρ} **courbepolar** – ⟶ trace la courbe paramétrée $\theta \mapsto (\rho(\theta)\cos\theta; \rho(\theta)\sin\theta)$ sur l'intervalle $[tmin; tmax]$

covariance

$array_1$ $array_2$ **covariance** c ⟶ le réel c est la covariance de la série double définie par les tableaux de réels $array_1$ et $array_2$

cpathelongueur

t cpathobj **cpathelongueur** ℓ ⟶ longueur du sous-chemin continu $\widehat{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et B est le dernier point de *cpathobj*

cpathendpoint

cpathobj **cpathendpoint** B ⟶ B est le dernier point du chemin continu paramétré *cpathobj*

cpathlongueur

cpathobj **cpathlongueur** ℓ ⟶ longueur de chemin continu parmétré *cpathobj*

cpathlongueurs

t cpathobj **cpathlongueurs** ℓ_1 ℓ_2 ⟶ ℓ_1 et ℓ_2 sont les longueurs respectives des sous-chemins $\widehat{AM}$ et $\widehat{MB}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et où A et B sont respectivement les premier et dernier points de *cpathobj*

cpathparamtable

cpathobj **cpathparamtable** *array* ⟶ *array* est le tableau des paramètres du chemin continu paramétré *cpathobj*

cpathpoint

t cpathobj **cpathpoint** M ⟶ M est le point de paramètre t du chemin continu paramétré *cpathobj*

cpathpointstable

cpathobj **cpathpointstable** *array* ⟶ *array* est le tableau de points du chemin continu paramétré *cpathobj*

cpathslongueur

t cpathobj **cpathslongueur** ℓ ⟶ longueur du sous-chemin continu $\widehat{AM}$, où M est le point de paramètre t du chemin continu paramétré *cpathobj*, et A est le premier point de *cpathobj*

cpathstartpoint

cpathobj **cpathstartpoint** A ⟶ A est le premier point du chemin continu paramétré *cpathobj*

cpathtocppath

$cpathobj_1$ **cpathtocppath** $cpathobj_2$ ⟶ transforme le chemin continu paramétré $cpathobj_1$ exprimé dans le repère jps en le chemin continu paramétré $cpathobj_2$ exprimé dans le repère postscript

cpoint

α *cerc* **cpoint** M ⟶ dépose sur la pile les coordonnées du point M du cercle *cerc* correspondant à l'angle α

cppathtocpath

$cpathobj_1$ **cppathtocpath** $cpathobj_2$ ⟶ transforme le chemin continu paramétré $cpathobj_1$ exprimé dans le repère postscript en le chemin continu paramétré $cpathobj_2$ exprimé dans le repère jps

creusesolid

solid **creusesolid** – ⟶ enlève les faces d'indice 0 et 1 du solide *solid*, puis ajoute toutes les faces internes

crpict

A [*xscale yscale*] {α} *string* **crpict** – ⟶ Se place à droite du point *A*, puis affiche l'objet désigné par la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

crtexlabel3d

crtexlabel3d – ⟶ Analogue 3d de la commande **crtexlabel**

crtexlabel

A [*xscale yscale*] {α} **crtexlabel** – ⟶ Se place à droite du point *A*, puis dessine le label TEX en cours centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

crtext3d

crtext3d – ⟶ Analogue 3d de la commande **crtext**

crtext

string A [*xscale yscale*] {α} **crtext** – ⟶ Se place à droite du point *A*, puis affiche la chaîne *string* centré, avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

currentcpathobj

– **currentcpathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère jps)

currentcpathpointstable

– **currentcpathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère jps

currentcppathobj

– **currentcppathobj** *cpathobj* ⟶ chemin continu paramétré courant (dans le repère postscript)

currentcppathpointstable

– **currentcppathpointstable** *array* ⟶ tableau des points définissant le chemin continu courant dans le repère postscript

currentpoint

– **currentpoint** *x y* ⟶ renvoie les coordonnées du point courant

curveto

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ **curveto** – ⟶ ajoute une section cubique de Bézier

cyan

– **cyan** – ⟶ sélectionne la couleur cyan

D

dapply

[$d_0 \ldots d_n$] *f* **dapply** [$b_0 \ldots b_n$] ou – ⟶ construit un nouveau tableau en répétant, pour *i* variant de 0 à *n*, l'opération suivante : déposer la droite d_i puis exécuter *f*. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

dashpoint

point **dashpoint** – ⟶ dessine le point spécifié avec projection sur les axes en pointillé

dashpoints

[$point_1 \ldots point_n$ **]** **dashpoints** – ⟶ dessine les points spécifiés avec projection sur les axes en pointillé

dbpict

A [*xscale yscale*] {α} *string* **dbpict** – ⟶ Se place en bas du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dbtexlabel3d

dbtexlabel3d – ⟶ Analogue 3d de la commande **dbtexlabel**

dbtexlabel

A [*xscale yscale*] {α} **dbtexlabel** – ⟶ Se place en bas du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dbtext3d

dbtext3d – ⟶ Analogue 3d de la commande **dbtext**

dbtext

string A [*xscale yscale*] {α} **dbtext** – ⟶ Se place en bas du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dcpict

A [*xscale yscale*] {α} *string* **dcpict** – ⟶ Se place en bas du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dctexlabel3d

dctexlabel3d – ⟶ Analogue 3d de la commande **dctexlabel**

dctexlabel

A [*xscale yscale*] {α} **dctexlabel** – ⟶ Se place en bas du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dctext3d

dctext3d $-\longrightarrow$ Analogue 3d de la commande **dctext**

dctext

string A [*xscale yscale*] $\{\alpha\}$ **dctext** $-\longrightarrow$ Se place en bas du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

defaultsolidmode

defaultsolidmode $\longrightarrow$ mode de résolution par défaut pour les solides. **valeur par défaut :** 2

defcercle

cerc name **defcercle** $-\longrightarrow$ associe le cercle *cerc* au nom *name*

defdroite

d name **defdroite** $-\longrightarrow$ associe la droite *d* au nom *name*

defpoint3d

x y z lit **defpoint3d** $-\longrightarrow$ Associe le littéral *lit* au point (x, y, z)

defpoint3d

x y z lit **defpoint3d** $-\longrightarrow$ Associe le littéral *lit* au point (x, y, z)

defpoint

x y name **defpoint** $-\longrightarrow$ affecte le nom *name* au couple de nombres (x, y)

demidroite

demidroite *A B option* $\longrightarrow -$: Trace la demi-droite [*AB*). *option* est une chaine de caractères optionnelle indiquant le type de terminaison de ligne en *A*

df+

x $\{f\}$ **df+** *a* $\longrightarrow$ Calcule une valeur approchée du nombre dérivé à droite de la fonction f en *x*

df-

x $\{f\}$ **df-** *a* $\longrightarrow$ Calcule une valeur approchée du nombre dérivé à gauche de la fonction f en *x*

diamcercle

A B **diamcercle** *cerc* $\longrightarrow$ *cerc* est le cercle de diamètre [*AB*]

diamond

A **diamond** $-\longrightarrow$ dessine un losange au point *A* dans le repère *jps*

dianode

string **dianode** $-\longrightarrow$ Déclare un nœud losange (diamond) dont le nom est défini par *string*

dich_solve

a b ε $\{f\}$ **dich_solve** *x* $\longrightarrow$ $\{f\}$ est un exécutable, le produit $f(a) \times proc(b)$ est négatif, et ε est un réel strictement positif. Alors *x* est un réel tel que $f(x+\varepsilon) \times f(x-\varepsilon) < 0$

dimmatrix

M **dimmatrix** *m n* $\longrightarrow$ dépose sur la pile les dimensions de la matrice *M* (*m* lignes, *n* colonnes)

dir

α **dir** $\vec{u} \longrightarrow \vec{u}$ est un vecteur correspondant à l'angle α (en degrés)

dir

α **dir** $\vec{v} \longrightarrow \vec{v}$ est le vecteur de norme 1 d'angle $(\widehat{\vec{u}, \vec{v}}) = \alpha$ où $\vec{u}$ désigne le vecteur unitaire de l'axe des abscisses

distance3d

A B **distance3d** *d* $\longrightarrow$ calcule la distance $d = AB$

distance3d

A B **distance3d** *d* $\longrightarrow$ calcule la distance $d = AB$

distance

A B **distance** $\ell \longrightarrow$ le nombre réel ℓ est la distance séparant les points *A* et *B*

divc

z z' **divc** *Z* $\longrightarrow Z = z/z'$ est le quotient des complexes *z* et *z'*

div

a b **div** *c* $\longrightarrow c = a/b$

dlpict

A [*xscale yscale*] {α} *string* **dlpict** – ⟶ Se place en bas à gauche du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dltexlabel3d

dltexlabel3d – ⟶ Analogue 3d de la commande **dltexlabel**

dltexlabel

A [*xscale yscale*] {α} **dltexlabel** – ⟶ Se place en bas à gauche du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dltext3d

dltext3d – ⟶ Analogue 3d de la commande **dltext**

dltext

string A [*xscale yscale*] {α} **dltext** – ⟶ Se place en bas à gauche du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

dotangle

dotangle ⟶ paramètre indiquant l'angle en degrés de la rotation à appliquer pour les dessins de type point. **valeur par défaut :** 0

dotscale

dotscale ⟶ exécutable indiquant les échelles horizontale et verticale à appliquer pour les dessins de type point. **valeur par défaut :** { 1 1 }

dotsize

dotsize ⟶ dimension en points postcript paramétrant la taille des dessins de type point. **valeur par défaut :** 4

dotted

– **dotted** – ⟶ sélectionne le tracé de type *pointilles*

doublebubblesort

$array_1$ **doublebubblesort** $array_2$ $array_3$ ⟶ $array_3$ est obtenu en triant $array_1$ par ordre croissant et $array_2$ correspond à la position des indices de départ dans le tableau d'arrivé, ie si $array_1 = [13, 12, 14, 11]$, alors $array_2 = [3, 1, 0, 2]$

down

– **down** $\vec{u}$ ⟶ $\vec{u}$ est le vecteur $(0, -1)$

draw_arrow

pathobj **draw_arrow** – ⟶ considère *pathobj* comme l'axe d'une flèche et dessine la flèche correspondante

drawcpath

cpathobj string **drawcpath** – ⟶ Dessine le chemin continu représenté par *cpathobj*. *string* est un paramètre optionnel indiquant le type de terminaison de ligne

drawsolid

solid **drawsolid** – ⟶ dessine le solide *solid*

drawsolid*

solid **drawsolid*** – ⟶ dessine le solide *solid* avec coloriage des faces visibles

drawsolid**

solid **drawsolid**** – ⟶ dessine le solide *solid* avec coloriage des faces visibles et en utilisant l'algorithme du peintre

droite

d **droite** – ⟶ trace la droite *d*

dupc

cerc **dupc** *cerc cerc* ⟶ duplique le cercle au dessus de la pile

dupd

D **dupd** *D D* ⟶ duplique la droite au dessus de la pile

dupmatrix

M **dupmatrix** *M M′* ⟶ dépose une nouvelle instance de M sur la pile

dupp3d

A **dupp3d** *A A* ⟶ Dupplique le point 3d au dessus de la pile

dupp3d

A **dupp3d** *A A* ⟶ Dupplique le point 3d au dessus de la pile

dupp

A **dupp** *A A* ⟶ duplique le point au dessus de la pile

dupsolid

solid **dupsolid** *solid solid* ⟶ crée une nouvelle instance du solide déposé sur la pile

dupv3d

u **dupv3d** *u u* ⟶ Dupplique le vecteur *u* au dessus de la pile

dupv3d

u **dupv3d** *u u* ⟶ Dupplique le vecteur *u* au dessus de la pile

dx_boxit

dx_boxit ⟶ taille, en points postscript, de l'espace horizontal entre un objet et son cadre placé par l'environnement 'picture' . **valeur par défaut :** 0

dy_boxit

dy_boxit ⟶ taille, en points postscript, de l'espace vertical entre un objet et son cadre placé par l'environnement 'picture' . **valeur par défaut :** 0

E

ecarttype

[$a_0 \ldots a_n$] **ecarttype** σ ⟶ le réel σ est l'écart-type de la série des a_i.

ell2pol

ell **ell2pol** *pol* ⟶ le polygône *pol* est constitué des 4 sommets de l'ellipse

ella

ell **ella** *a* ⟶ *a* est la longueur du demi-axe horizontal de l'ellipse *ell*

ellangle

ell **ellangle** α ⟶ α est l'angle de l'ellipse *ell*

ellb

ell **ellb** *b* ⟶ *b* est la longueur du demi-axe vertical de l'ellipse *ell*

ellcentre

ell **ellcentre** *A* ⟶ le points *A* est le centre de l'ellipse *ell*

ellipseangle

ellipseangle ⟶ angle que fait le premier axe d'une ellipse avec l'horizontale lorsque l'on trace cette ellipse avec la syntaxe allégée des commandes **ellipse** et **Ellipse**. **valeur par défaut :** 0

Ellipse_

α β *ell* **Ellipse_** – ⟶ ajoute au chemin courant la portion de l'ellipse entre les points de paramètres respectifs α et β (dans ce sens).

Ellipse

α β *ell* **Ellipse** – ⟶ trace la portion de l'ellipse entre les points de paramètres respectifs α et β.

Ellipse

α β *I a b* **Ellipse** – ⟶ trace la portion spécifiée de l'ellipse de centre *I* et de demi-axes *a* et *b*. L'angle que fait le premier axe avec l'horizontale est déterminé par le paramètre *ellipseangle*

ellipse_

ell **ellipse_** – ⟶ ajoute au chemin courant le chemin de l'ellipse spécifiée

ellipse

ell **ellipse** – ⟶ trace l'ellipse spécifiée

ellipse

I a b **ellipse** – ⟶ trace l'ellipse de centre *I* et de demi-axes *a* et *b*. L'angle que fait le premier axe avec l'horizontale est déterminé par le paramètre *ellipseangle*

e

– **e** 2,718 ⟶ le nombre *e*

enode

x y string **enode** $-\longrightarrow$ Déclare un nœud vide (*emptynode*) au point de coordonées (x,y) et dont le nom est défini par *string*

Epoint

α *ell* **Epoint** $A \longrightarrow A$ est le point de l'ellipse *ell* tel que $(\vec{u}, \overrightarrow{\Omega A}) = \alpha$, où Ω désigne le centre de l'ellipse, et $\vec{u}$ le vecteur de base de l'axe Ox.

epoint

t ell **epoint** $A \longrightarrow A$ est le point de l'ellipse *ell* de paramètre t (avec le paramétrage $(a\cos t, b\sin t)$)

eqc

$z\ z'$ **eqc** *bool* $\longrightarrow$ le booléen *bool* vaut **true** si les complexes z et z' sont égaux, **false** sinon.

eqp

$A\ B$ **eqp** *bool* $\longrightarrow$ le booléen *bool* vaut **true** si les points A et B sont confondus, **false** sinon

Euler

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Euler** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

Euler

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Euler** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

euler

$\{f\}\ x_0\ y_0\ h$ **euler** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

euler

$\{f\}\ x_0\ y_0\ n$ **euler** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

Eulermod

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Eulermod** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

Eulermod

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Eulermod** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[a,b]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

eulermod

$\{f\}\ x_0\ y_0\ h$ **eulermod** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

eulermod

$\{f\}\ x_0\ y_0\ n$ **eulermod** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode d'Euler modifiée, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x,y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

exch_l

$M\ i\ j$ **exch_l** $-\longrightarrow$ échange les lignes d'indice i et d'indice j dans la matrice M

exec

any **exec** $-\longrightarrow$ exécute un objet arbitraire

Exp

a **Exp** $c \longrightarrow c = \exp(a) = e^a$

exp

$a\ n$ **exp** $c \longrightarrow c = a^n$

exposant

string **exposant** – ⟶ affiche la chaîne *string* dans la police courante, après une réduction à 70% de la taille courante et un déplacement vertical (40% de *fontsize*) par rapport au point courant

F

factorielle

n **factorielle** *b* ⟶ $b = a!$

Fillcourbe

a b {*f*} **Fillcourbe** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de *f*, et les droites verticales $x = a$ et $x = b$

Fillcourbe

a b proc **Fillcourbe** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = a$ et $x = b$

fillcourbe

{*f*} **fillcourbe** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de *f*, et les droites verticales $x = xmin$ et $x = xmax$

fillcourbe

proc **fillcourbe** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = xmin$ et $x = xmax$

Fillcourbes

a b {*f*} {*g*} **Fillcourbes** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par les courbes représentatives des fonctions *f* et *g*, et les droites verticales $x = a$ et $x = b$

Fillcourbes

a b $proc_1$ $proc_2$ **Fillcourbes** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = a$ et $x = b$

fillcourbes

{*f*} {*g*} **fillcourbes** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par les courbes représentatives des fonctions *f* et *g*, et les droites verticales $x = xmin$ et $x = xmax$

fillcourbes

$proc_1$ $proc_2$ **fillcourbes** – ⟶ remplit, suivant les indications de *fillstyle*, le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = xmin$ et $x = xmax$

flattenpath

– **flattenpath** – ⟶ convertit les courbes en suites de segments de droites

floor

num_1 **floor** num_2 ⟶ plancher de num_1

fontsize

fontsize ⟶ taille, en points postscript, de la prochaine fonte chargée. **valeur par défaut :** 12, 5

frameangle

frameangle ⟶ angle que fait avec l'horizontale le côté « bas » d'un rectangle que l'on trace avec la syntaxe allégée d'une commande **frame** ou dérivée. **valeur par défaut :** 0

frame_

A B **frame_** – ⟶ ajoute au chemin courant le rectangle dont les points *A* et *B* sont repectivement les coins inférieur droit et supérieur gauche

frame

A B **frame** – ⟶ trace le rectangle dont les points *A* et *B* sont repectivement les coins inférieur droit et supérieur gauche

fresnelC

x **fresnelC** *y* ⟶ *y* est l'image de *x* par la fonction de Fresnel $x \mapsto \int_0^x \cos\left(\frac{\pi t^2}{2}\right) dt$

fresnelS

x **fresnelS** *y* ⟶ *y* est l'image de *x* par la fonction de Fresnel $x \mapsto \int_0^x \sin\left(\frac{\pi t^2}{2}\right) dt$

fuz

[$a_0 \ldots a_n$] [$b_0 \ldots b_n$] **fuz** [$a_0\ b_0 \ldots a_n\ b_n$] ⟶ fusionne les 2 tableaux de même tailles donnés en entrée

G

Gammaln

x **Gammaln** *y* $\longrightarrow$ *y* est l'image de *x* par la fonction $\ln \circ \Gamma$

Gauss

x m σ **Gauss** *y* $\longrightarrow$ *y* est l'image de *x* par la loi normale de paramètres *m* et σ. Soit $y = e^{-\left((x-m)/\sigma\right)^2/2}/(\sigma\sqrt{2\pi})$

generesolid

$array_1$ $array_2$ **generesolid** *solid* $\longrightarrow$ construit un solide dont le tableaux des sommets est $array_1$ et le tableau des faces est $array_2$

genMi

/*M n* **genMi** *M*1 *M*2 . . . *Mn* $\longrightarrow$ dépose les valeurs des noms /*M*1, /*M*2, . . ., /*Mn* sur la pile

genMiname

/*M n* **genMiname** *M*1 *M*2 . . . *Mn* $\longrightarrow$ dépose les noms /*M*1, /*M*2, . . ., /*Mn* sur la pile

genpolyreg

I r n α **genpolyreg** *array* $\longrightarrow$ *array* est le tableau des *n* points sommets du polygône régulier de centre *I* tel que l'angle entre le vecteur unité sur *Ox* et le vecteur IA soit de α degrés (où *A* est le « premier » sommet du polygône)

geodesique_sphere

r θ_1 ϕ_1 *r* θ_2 ϕ_2 **geodesique_sphere** – $\longrightarrow$ trace le cercle passant par les points *A* et *B* de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

geodesique_sphere

r θ_1 ϕ_1 *r* θ_2 ϕ_2 **geodesique_sphere** – $\longrightarrow$ trace le cercle passant par les points *A* et *B* de coordonnées respectives (r, θ_1, ϕ_1) et (r, θ_2, ϕ_2)

GetCamPos

– **GetCamPos** *x y z* $\longrightarrow$ Dépose sur la pile les coordonnées de la caméra

GetCamUp

– **GetCamUp** U_x U_y U_z $\longrightarrow$ *Set Camera Up vector.*

GetCamVec

– **GetCamVec** V_x V_y V_z $\longrightarrow$ *Get Camera Looking vector.*

get_Ci

M i **get_Ci** *array* $\longrightarrow$ le tableau *array* représenta la colonne d'indice *i* de la matrice *M*

get_ij

M i j **get_ij** *a* $\longrightarrow$ *a* est le coefficient d'indice (*i*, *j*) de la matrice *M*

get_Li

M i **get_Li** *array* $\longrightarrow$ *array* représente la ligne d'indice *i* de la matrice *M*

getp3d

getp3d $\longrightarrow$

getp

[A_0 . . . A_n] *i* **getp** A_i $\longrightarrow$ donne le point d'indice *i* du tableau de points donné en entrée.

gradangle

gradangle $\longrightarrow$ Angle utilisé pour les gradients de couleurs. **valeur par défaut :** 0

gris

– **gris** – $\longrightarrow$ sélectionne la couleur gris

H

Hachcourbe

a b {*f*} **Hachcourbe** – $\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de *f*, et les droites verticales *x* = *a* et *x* = *b*

Hachcourbe

a b proc **Hachcourbe** – $\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales *x* = *a* et *x* = *b*

hachcourbe

{*f*} **hachcourbe** – $\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de *f*, et les droites verticales *x* = *xmin* et *x* = *xmax*

hachcourbe

proc **hachcourbe** $-\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, la courbe représentative de la fonction numérique définie par *proc*, et les droites verticales $x = xmin$ et $x = xmax$

Hachcourbes

$a\ b\ \{f\}\ \{g\}$ **Hachcourbes** $-\longrightarrow$ hachure le domaine plan délimité par les courbes représentatives des fonctions f et g, et les droites verticales $x = a$ et $x = b$

Hachcourbes

$a\ b\ proc_1\ proc_2$ **Hachcourbes** $-\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = a$ et $x = b$

hachcourbes

$\{f\}\ \{g\}$ **hachcourbes** $-\longrightarrow$ hachure le domaine plan délimité par les courbes représentatives des fonctions f et g, et les droites verticales $x = xmin$ et $x = xmax$

hachcourbes

$proc_1\ proc_2$ **hachcourbes** $-\longrightarrow$ hachure le domaine plan délimité par l'axe *Ox*, les courbes représentatives des fonctions numérique définies par $proc_1$ et $proc_2$, et les droites verticales $x = xmin$ et $x = xmax$

hachure

$-$ **hachure** $-\longrightarrow$ hachure l'ensemble de la fenêtre courante

had just

had just $\longrightarrow$ taille, en points postscript, du décalage horizontal appliqué, s'il y a lieu par les commandes de positionnement de l'environnement 'picture' . **valeur par défaut :** $3,75$

hangle

hangle $\longrightarrow$ l'angle en degrés que font les hachures avec l'horizontale. **valeur par défaut :** -45

hcolor

hcolor $\longrightarrow$ couleur d'un hachure. **valeur par défaut :** $\{\}$

homcercle

cerc $I\ \alpha$ **homcercle** *cerc'* $\longrightarrow$ le cercle *cerc'* est l'image du cercle *cerc* par l'homothétie de centre I, de rapport α.

homcpath

$cpathobj_1\ I\ k$ **homcpath** $cpathobj_2 \longrightarrow cpathobj_2$ est le chemin continu image de $cpathobj_1$ par l'homothétie de centre I et de rapport k

homell

ell $I\ \alpha$ **homell** *ell'* $\longrightarrow$ l'ellipse *ell'* est l'image de l'ellipse *ell* par l'homothétie de centre I, de rapport α.

hompath

$pathobj_1\ I\ k$ **hompath** $pathobj_2 \longrightarrow pathobj_2$ est le chemin image de $pathobj_1$ par l'homothétie de centre I et de rapport k

hompoint3d

$M\ A\ \alpha$ **hompoint3d** $M' \longrightarrow M'$ est l'image de M par l'homothétie de centre A et de rapport α

hompoint3d

$M\ A\ \alpha$ **hompoint3d** $M' \longrightarrow M'$ est l'image de M par l'homothétie de centre A et de rapport α

hompoint

$A\ I\ \alpha$ **hompoint** $A' \longrightarrow$ le point A' est l'image du point A par l'homothétie de centre I, de rapport α. Autrement dit $\overrightarrow{IA'} = \alpha\overrightarrow{IA}$

hompol

pol $I\ \alpha$ **hompol** *pol'* $\longrightarrow$ le polygône *pol'* est l'image du polygône *pol* par l'homothétie de centre I, de rapport α.

horizontale

b **horizontale** $D \longrightarrow$ dépose sur la pile la droite horizontale D d'équation $y = b$

hstep

hstep $\longrightarrow$ l'espace en points postscript séparant 2 hachures. **valeur par défaut :** 7

hwidth

hwidth $\longrightarrow$ l'épaisseur du trait en points postscript pour une hachure. **valeur par défaut :** $0,8$

I

IAcercle

I A **IAcercle** *cerc* $\longrightarrow$ *cerc* est le cercle de centre *I* passant par *A*

idiv

a b **idiv** *q* $\longrightarrow$ *q* est le quotient de la division euclidienne de *a* par *b*

idmatrix

m n **idmatrix** *M* $\longrightarrow$ dépose une nouvelle matrice identité (m, n) sur la pile

ifelse

bool $proc_1$ $proc_2$ **ifelse** $-$ $\longrightarrow$ exécute $proc_1$ si *bool* est $true$, $proc_2$ si *bool* est $false$

if

bool proc **if** $-$ $\longrightarrow$ exécute *proc* si *bool* est $true$

indice

string **indice** $-$ $\longrightarrow$ affiche la chaîne *string* dans la police courante, après une réduction à 70% de la taille courante et un déplacement vertical (20% de $fontsize$) par rapport au point courant

in

any string/*array* **in** *i bool* $\longrightarrow$ cherche si l'élément *any* est dans *string*/*array*. renvoie $false$ dans la négative, ou *i* et $true$ dans l'affirmative, *i* étant l'indice de *any* dans *string*/*array*

inoutputcolors

solid str_1 str_2 **inoutputcolors** $-$ $\longrightarrow$ affecte la couleur définie par str_2 aux faces externes du solide *solid*, et la couleurs définie par str_1 aux faces internes

inoutputcolors

solid $string_0$ $string_1$ **inoutputcolors** $-$ $\longrightarrow$ affecte la couleur définie par $string_0$ aux faces internes du solide *solid*, et la couleur définie par $string_1$ aux faces externes

intercercle

$cerc_1$ $cerc_2$ **intercercle** $A\,A'$ $\longrightarrow$ les points *A* et A' sont les points d'intersection du cercle $cerc_1$ avec le cercle $cerc_2$, triés par la fonction *ordonnepoints*. Comme d'habitude, l'appel de cette fonction provoque une erreur si ces cercles n'ont pas de point commun.

interdroitecercle

D cerc **interdroitecercle** $A\,A'$ $\longrightarrow$ les points *A* et A' sont les points d'intersection de la droite *D* avec le cercle *cerc*, triés par la fonction *ordonnepoints*

interdroiteell

D ell **interdroiteell** $A\,A'$ $\longrightarrow$ les points *A* et A' sont les points d'intersection de la droite *D* avec l'ellipse *ell*, triés par la fonction *ordonnepoints*

interdroite

D D' **interdroite** *A* $\longrightarrow$ si les droites *D* et D' sont sécantes, alors *A* est leur point d'intersection. Erreur sinon

interpolparamfunct

interpolparamfunct $\longrightarrow$

isobarycentre3d

[$A_0 \ldots A_n$] **isobarycentre3d** *G* $\longrightarrow$ le point *G* est le barycentre du système $[(A_0, 1); \ldots; (A_n, 1)]$

isobarycentre3d

[$A_0 \ldots A_n$] **isobarycentre3d** *G* $\longrightarrow$ le point *G* est le barycentre du système $[(A_0, 1); \ldots; (A_n, 1)]$

issolid

any **issolid** *bool* $\longrightarrow$ *bool* vaut $true$ si *any* est de type *solid*, $false$ sinon

J

jaune

$-$ **jaune** $-$ $\longrightarrow$ sélectionne la couleur jaune

jtoppoint

X Y **jtoppoint** *x y* $\longrightarrow$ Reçoit les coordonnées (X, Y) dans le repère jps et renvoie les coordonnées (x, y) dans le repère postscript

L

lambdav3d

$\lambda\,\vec{u}$ **lambdav3d** $\vec{v}$ $\longrightarrow$ Le vecteur $\vec{v}$ vérifie $\vec{v} = \lambda\vec{u}$

lambdav3d

$\lambda\ \vec{u}$ **lambdav3d** $\vec{v} \longrightarrow$ Le vecteur $\vec{v}$ vérifie $\vec{v} = \lambda\vec{u}$

lastcpath

– **lastcpath** *cpathobj* $\longrightarrow$ l'objet cpath, coordonnées dans le repère jps, associé au dernier chemin continu dessiné

lastcppath

– **lastcppath** *cpathobj* $\longrightarrow$ l'objet cpath, coordonnées dans le repère postcript, associé au dernier chemin continu dessiné

left

– **left** $\vec{u} \longrightarrow \vec{u}$ est le vecteur $(-1, 0)$

lightintensity

lightintensity $\longrightarrow$ Intensité de la source lumineuse ponctuelle. **valeur par défaut :** 1

ligne3d

array **ligne3d** – $\longrightarrow$ Analogue 3d de la commande **ligne**

ligne3d

array **ligne3d** – $\longrightarrow$ Analogue 3d de la commande **ligne**

ligne_

array **ligne_** – $\longrightarrow$ ajoute au chemin courant la ligne définie par le tableau de points *array*

ligne

array string **ligne** – $\longrightarrow$ trace la ligne définie par le tableau de points *array*. Les extrémités de la ligne sont décrites par la chaîne optionnelle *string*

linearc

linearc $\longrightarrow$ indique au format le rayon (dans le repère jps) de l'arc de cercle reliant deux segments de droites pour les commandes **ligne**, **polygone**, ainsi que les **frame** et dérivés. . **valeur par défaut :** 0

line

A B string **line** – $\longrightarrow$ trace le segment de droite $[AB]$. Les extrémités du segment sont décrites par la chaîne optionnelle *string*

lineto

x y **lineto** – $\longrightarrow$ ajoute une ligne droite jusqu'en (x, y)

ln

a **ln** $c \longrightarrow c = \ln a$, logarithme népérien de *a*

log_bande

i **log_bande** – $\longrightarrow$ *i* est un entier. trace 10 traits horizontaux et 10 traits verticaux sur $\left]10^{i-1}; 10^{i}\right]$ en utilisant les commandes **hrule** et **vrule**

logicNInput

logicNInput $\longrightarrow$ Nombre d'entrées de la cellule. **valeur par défaut :** 2

logicUnit

logicUnit $\longrightarrow$ Échelle pour le dessin du symbole. **valeur par défaut :** $0,5$

logicWireLength

logicWireLength $\longrightarrow$ Longueur des pattes de raccordement (unité jps). **valeur par défaut :** $0,5$

log

a **log** $c \longrightarrow c = \log a$, logarithme décimal de *a*

log_seq

a b **log_seq** $10^a\ 2.10^a\ 3.10^a \ldots 9.10^a\ 10^{a+1}\ 2.10^{a+1} \ldots 9.10^{a+1} \ldots 9.10^b \longrightarrow$ *a* et *b* sont des entiers. Génère une séquence pour une échelle logarithmique de graduations entre 10^a et 10^b

log_xbande

i **log_xbande** – $\longrightarrow$ *i* est un entier. trace 10 traits verticaux $\left]10^{i-1}; 10^{i}\right]$ en utilisant la commande **vrule**

log_xmark

n **log_xmark** – $\longrightarrow$ *n* est un entier. affiche la numérotation correspondant à 10^n sur l'axe *Ox*

log_ybande

i **log_ybande** – $\longrightarrow$ *i* est un entier. trace 10 traits horizontaux $\left]10^{i-1}; 10^{i}\right]$ en utilisant la commande **hrule**

log_ymark

n **log_ymark** $-\longrightarrow$ *n* est un entier. affiche la numérotation correspondant à 10^n sur l'axe *Oy*

loop

proc **loop** $-\longrightarrow$ exécute *proc* un nombre indéfini de fois

M

magenta

$-$ **magenta** $-\longrightarrow$ sélectionne la couleur magenta

mapnc

[$lit_0\ lit_1 \ldots lit_n$] [$C_0\ C_1 \ldots C_n$] **mapnc** $-\longrightarrow$ associe, pour $0 \leqslant i \leqslant n$, le littéral lit_i et le cercle C_i dans le dictionnaire courant

mapnp

[$lit_0\ lit_1 \ldots lit_n$] [$A_0\ A_1 \ldots A_n$] **mapnp** $-\longrightarrow$ associe, pour $0 \leqslant i \leqslant n$, le littéral lit_i et le point A_i dans le dictionnaire courant

mapnu

[$lit_0\ lit_1 \ldots lit_n$] [$a_0\ a_1 \ldots a_n$] **mapnu** $-\longrightarrow$ associe, pour $0 \leqslant i \leqslant n$, le littéral lit_i et la valeur unaire a_i dans le dictionnaire courant

marked

A B n **marked** $-\longrightarrow$ marque le segment [*AB*] avec *n* traits inclinés

marks

$-$ **marks** $-\longrightarrow$ inscrit toute la numérotation des axes *Ox* et *Oy*

marks

$-$ **marks** $-\longrightarrow$ numérote les graduations sur les axes *Ox* et *Oy*

{masque-_

$-$ **masque-_** $-\longrightarrow$ ajoute au chemin courant le rectangle de coin inférieur gauche le point (*xmin*, *ymin*) et de coin supérieur droit (*xmax*, *ymax*). Le chemin est parcouru dans le sens des aiguilles d'une montre

{masque_

$-$ **masque_** $-\longrightarrow$ ajoute au chemin courant le rectangle de coin inférieur gauche le point (*xmin*, *ymin*) et de coin supérieur droit (*xmax*, *ymax*). Le chemin est parcouru dans le sens inverse des aiguilles d'une montre

max

a b **max** *c* $\longrightarrow$ *c* est le plus grand des deux nombres *a* et *b*

max

a b **max** *c* $\longrightarrow$ *c* est le plus grand des deux nombres *a* et *b*

Mayer

$array_1$ $array_2$ **Mayer** *d* $\longrightarrow$ la droite *d* est la droite de Mayer définie par les tableaux de réels $array_1$ et $array_2$ définissant respectivement les abscisses et les ordonnées d'un nuage de points

mediane

[$a_0 \ldots a_n$] **mediane** *m* $\longrightarrow$ le réel *m* est la médiane de la série des a_i.

mediatrice

A B **mediatrice** *D* $\longrightarrow$ *D* est la médiatrice du segment [*AB*]

methodetrapeze

a b {*f*} *n* **methodetrapeze** *real* $\longrightarrow$ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode des trapèzes pour $n+1$ points (*n* est un entier)

mframe_

A L ℓ **mframe_** $-\longrightarrow$ ajoute au chemin courant le rectangle dont le point *A* est le milieu du côté inférieur, de dimension horizontale *L* et de dimension verticale ℓ

mframe

A L ℓ **mframe** $-\longrightarrow$ trace le rectangle dont le point *A* est le milieu du côté inférieur, de dimension horizontale *L* et de dimension verticale ℓ

milieu3d

A B **milieu3d** *I* $\longrightarrow$ *I* est le milieu de [*AB*]

milieu3d

A B **milieu3d** *I* $\longrightarrow$ *I* est le milieu de [*AB*]

milieu

$A\ B$ **milieu** $I \longrightarrow$ le point I est le milieu du segment $[AB]$

Milne

$a\ b\ \{f\}\ x_0\ y_0\ h$ **Milne** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[a, b]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

Milne

$a\ b\ \{f\}\ x_0\ y_0\ n$ **Milne** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[a, b]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

milne

$\{f\}\ x_0\ y_0\ h$ **milne** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ dépose les points, calculés par la méthode de Milne, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est un réel positif, il détermine le pas entre chaque point calculé

milne

$\{f\}\ x_0\ y_0\ n$ **milne** $x_{-n}\ y_{-n} \ldots x_{-1}\ y_{-1}\ x_0\ y_0\ x_1\ y_1 \ldots x_n\ y_n \longrightarrow$ n étant un entier, cette procédure dépose les points, calculés par la méthode de Milne, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

min

$a\ b$ **min** $c \longrightarrow c$ est le plus petit des deux nombres a et b

min

$a\ b$ **min** $c \longrightarrow c$ est le plus petit des deux nombres a et b

mixte

$-$ **mixte** $- \longrightarrow$ sélectionne le tracé de type trait mixte

mod

$a\ b$ **mod** $r \longrightarrow r$ est reste de la division euclidienne de a par b

module

z **module** $r \longrightarrow$ le réel $r = |z|$

moveto

$x\ y$ **moveto** $- \longrightarrow$ définit le point courant à (x, y)

moyenne

$[\ a_0\ a_1 \ldots a_n\ ]$ **moyenne** $m \longrightarrow m$ est la moyenne arithmétique des nombres $a_0, a_1, \ldots, a_n$

moyenne

$[\ a_0 \ldots a_n\ ]$ **moyenne** $m \longrightarrow$ le réel m est la moyenne arithmétique de la série des a_i. $m = \left(\sum_{i=0}^{n} a_i\right) / (n + 1)$.

mulc

$z\ z'$ **mulc** $Z \longrightarrow Z = zz'$ est le produit des complexes z et z'

mul

$a\ b$ **mul** $c \longrightarrow c = a \times b$

mulm

$A\ B$ **mulm** $M \longrightarrow$ multiplie les matrices A et B et dépose le résultat sur la pile

mulv3d

$\vec{u}\ \lambda$ **mulv3d** $\vec{v} \longrightarrow \vec{v} = \lambda\vec{u}$

mulv3d

$\vec{u}\ \lambda$ **mulv3d** $\vec{v} \longrightarrow \vec{v} = \lambda\vec{u}$

mulv

$u\ a$ **mulv** $\vec{U} \longrightarrow \vec{U} = a\vec{u}$ où a est un nombre réel

N

ncangle

$string_1\ string_2\ option$ **ncangle** $- \longrightarrow$ Trace en B, et suivant l'angle *angleB* un bras de longueur *armB*, puis elle connecte ce bras en A, suivant l'angle *angleA* par un double segment à angle droit.

ncangles

$string_1\ string_2\ option$ **ncangles** $- \longrightarrow$ Trace en A (resp. B), et suivant l'angle *angleA* (resp. *angleB*) un bras

de longueur *armA* (resp. *armB*), puis elle connecte les 2 bras par un double segment à angle droit (en partant de *B*).

ncarc

string$_1$ *string*$_2$ *option* **ncarc** – ⟶ Connecte les nodes avec une courbe de Bézier, en utilisant le paramètre *nodesep*. La courbe se connecte en *A* avec un angle *arcangleA* par rapport à la droite (*AB*) et se connecte en *B* avec un angle $-$*arcangleB* par rapport à la droite (*AB*).

ncbar

string$_1$ *string*$_2$ *option* **ncbar** – ⟶ Trace d'abord les bras de longueurs respectives *armA* et *armB* à un angle *angleA*. Ensuite, l'un des bras est étendu puis connecté, de telle façon que la ligne finale soit composée de 3 segments à angle droit.

nccurve

string$_1$ *string*$_2$ *option* **nccurve** – ⟶ Trace une courbe de Bézier entre les nodes *A* et *B*. Les paramètres *angleA* et *angleB* sont utilisés.

ncdiagg

string$_1$ *string*$_2$ *option* **ncdiagg** – ⟶ Trace d'abord, en *A* et à l'angle *angleA*, le bras de longueur *armA*. Ensuite, ce bras est directement connecté au point *B*. Le paramètre *linearc* est utilisé pour arrondir les angles.

ncdiag

string$_1$ *string*$_2$ *option* **ncdiag** – ⟶ Trace d'abord les bras de longueurs respectives *armA* et *armB* à des angles respectifs *angleA* et *angleB*. Ensuite, ces bras sont connecté par une ligne droite. Le paramètre *linearc* est utilisé pour arrondir les angles.

ncline

string$_1$ *string*$_2$ *option* **ncline** – ⟶ Trace une simple ligne entre les nodes *A* et *B*. Seul le paramètre *nodesep* est utilisé.

neg

a **neg** *c* ⟶ $c = -a$

newanneau

array n **newanneau** *solid* ⟶ crée un nouvel anneau, de type *solid*, dont la section est définie par *array*, un tableau de points (x_i, z_i). Le maillage est défini par *n*. *n* peut être un entier représentant le nombre de mailles, ou un exécutable représentant un mode..

newbiface

array **newbiface** *solid* ⟶ construit un solide biface, dans le plan *xOy*, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique ⇒ face de normale *Oz*, dans le sens des *z* croissants)

newcalottespherecreuse

r ϕ θ **newcalottespherecreuse** *solid* ⟶ crée une nouvelle calotte sphérique creuse, de rayon r et d'angles d'ouverture ϕ et θ. *option* indique le maillage ou le mode.

newcalottesphere

r ϕ θ **newcalottesphere** *solid* ⟶ crée une nouvelle calotte sphérique, de rayon r et d'angles d'ouverture ϕ et θ. *option* indique le maillage ou le mode.

newcone

z_0 r z_1 **newcone** *solid* ⟶ crée un nouveau cône, de type *solid*, d'axe *Oz*, de rayon r, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

newcone

z_0 r z_1 *option* **newcone** *solid* ⟶ crée un nouveau cône, de type *solid*, d'axe *Oz*, de rayon r, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

newcube

a option **newcube** *solid* ⟶ crée un nouveau cube, de type *solid*, de centre *O*, d'arête *a*. *option* indique le maillage ou le mode.

newcylindrecreux

z_0 z_1 r_1 *option* **newcylindrecreux** *solid* ⟶ crée un nouveau cylindre creux, de type *solid*, d'axe *Oz*, de rayon r, allant du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

newcylindre

z_0 r z_1 *option* **newcylindre** *solid* ⟶ crée un nouveau cylindre, de type *solid*, d'axe *Oz*, de rayon r, allant

du plan $z = z_0$ jusqu'au plan $z = z_1$. *option* indique le maillage ou le mode.

newdodecaedre

r **newdodecaedre** *solid* $\longrightarrow$ crée un nouveau dodécaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

newgrille

xmin xmax ymin ymax option **newgrille** *solid* $\longrightarrow$ crée une nouvelle grille, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou un exécutable et représenter un mode.

newicosaedre

r **newicosaedre** *solid* $\longrightarrow$ crée un nouvel isocaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

newmatrix

m n **newmatrix** *M* $\longrightarrow$ dépose une nouvelle matrice nulle (m, n) sur la pile

newmonoface

array **newmonoface** *solid* $\longrightarrow$ construit un solide monoface, dans le plan *xOy*, à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale *Oz*, dans le sens des *z* croissants)

newoctaedre

r **newoctaedre** *solid* $\longrightarrow$ crée un nouvel octaèdre régulier, de type *solid*, , inscrit dans la sphère de centre *O* et de rayon *r*

newpath

– **newpath** – $\longrightarrow$ initialise et vide le chemin courant

newprismedroit

array z_0 z_1 **newprismedroit** *solid* $\longrightarrow$ construit un prisme droit d'axe *Oz* à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale *Oz*, dans le sens des *z* croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$

newprisme

array z_0 z_1 $\vec{u}$ **newprisme** *solid* $\longrightarrow$ construit un prisme oblique d'axe $(O, \vec{u})$ à partir du tableau de points *array*. Les points sont en 2d, et rangés de manière à décrire l'orientation de la face (sens trigonométrique $\Rightarrow$ face de normale *Oz*, dans le sens des *z* croissants). La base du prisme est sur le plan $z = z_0$ et sa face supérieure sur le plan $z = z_1$

newruban

array z option **newruban** *solid* $\longrightarrow$ *array* est un tableau de points en 2d indiquant les sommets du ruban sur le plan *xOy*, et *z* est un nombre indiquant la hauteur du ruban. L'argument optionnel *option* permet de spécifier un mode ou un maillage.

newsolid

– **newsolid** *solid* $\longrightarrow$ dépose le solide nul sur la pile

newsphere

r option **newsphere** *solid* $\longrightarrow$ crée une nouvelle sphère, de type *solid*, de centre *O*, de rayon *r*. Le paramètre $mode \in \{0, 1, 2, 3, 4\}$ est optionnel; il indique le niveau de résolution souhaité (0 = mini, 4 = maxi). *option* indique le maillage ou le mode.

newsurface

xmin xmax ymin ymax option $\{f\}$ **newsurface** *solid* $\longrightarrow$ crée une nouvelle surface, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou un exécutable et représenter un mode. L'exécutable f est une fonction de $[xmin, xmax] \times [ymin, ymax]$ dans $\mathbb{R}$.

newsurfaceparametree

xmin xmax ymin ymax option $\{f\}$ **newsurfaceparametree** *solid* $\longrightarrow$ crée une nouvelle surface paramétrée, de type *solid*, dont le maillage est éventuellement défini par *option*. *option* peut être un tableau et représenter un maillage, ou un exécutable et représenter un mode. L'exécutable f est une fonction de $[xmin, xmax] \times [ymin, ymax]$ dans $\mathbb{R}^3$.

newtetraedre

r **newtetraedre** *solid* $\longrightarrow$ crée un nouveau tétraèdre régulier, de type *solid*, inscrit dans la sphère de centre *O* et de rayon *r*

newton_solve

x_0 ε ε $\{f\}$ $\{f'\}$ **newton_solve** $x \longrightarrow$ $\{f\}$ et $\{f'\}$ sont des exécutables, et f' désigne la fonction dérivée de f. Le réel x est obtenu par la méthode des tangentes de Newton, avec la valeur initiale x_0 et la tolérance ε

newtore

r R option **newtore** *solid* $\longrightarrow$ crée un nouveau tore, de type *solid*, de centre *O*, de rayon principal *R*, et de rayon du tube *r*. *option* indique le maillage ou le mode.

newtroncconecreux

z_0 r_0 z_1 r_1 *option* **newtroncconecreux** *solid* $\longrightarrow$ crée un nouveau tronc de cône creux, de type *solid*, d'axe *Oz*, de rayon de base r_0 (sur le plan $z = z_0$) et de rayon au sommet r_1 (sur le plan $z = z_1$). *option* indique le maillage ou le mode.

newtronccone

z_0 r_0 z_1 r_1 *option* **newtronccone** *solid* $\longrightarrow$ crée un nouveau tronc de cône, de type *solid*, d'axe *Oz*, de rayon de base r_0 (sur le plan $z = z_0$) et de rayon au sommet r_1 (sur le plan $z = z_1$). *option* indique le maillage ou le mode.

newvecteur

x y z **newvecteur** *solid* $\longrightarrow$ crée un nouveau vecteur, de coordonnées (x, y, z).

node

string **node** – $\longrightarrow$ Déclare un nœud rectangulaire dont le nom est défini par *string*

nodesep

nodesep $\longrightarrow$ espace en picas séparant le point de connexion et l'extrémité du trait de connexion. **valeur par défaut :** 3

noir

– **noir** – $\longrightarrow$ sélectionne la couleur noir

nomme_noeud

tnode string **nomme_noeud** *tnode* $\longrightarrow$ Positionne à *string* le champ *nom* du nœud d'arbre *tnode*

non_relie

tnode **non_relie** *tnode* $\longrightarrow$ Positionne à *false* le champ adapté du nœud d'arbre *tnode*

normalize3d

$\vec{u}$ **normalize3d** $\vec{v} \longrightarrow$ Sinonyme de **unitaire3d** : si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

normalize3d

$\vec{u}$ **normalize3d** $\vec{v} \longrightarrow$ Sinonyme de **unitaire3d** : si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

normalizecpath

$cpathobj_1$ **normalizecpath** $cpathobj_2 \longrightarrow$ Les chemins décrits par $cpathobj_1$ et $cpathobj_2$ sont confondus, mais le tableau des paramètres de $cpathobj_2$ a été modifié, de telle facon que ceux-ci soient dans l'intervalle [0; 100], et qu'ils soient répartis de façon proportionnelle à la longueur du chemin

normalize

u **normalize** *v* $\longrightarrow$ *v* est le vecteur de norme 1 obtenu à partir de *u*, ie. $\vec{v} = \frac{1}{\|u\|}\vec{u}$

normal

u **normal** *v* $\longrightarrow$ le vecteur *v* vérifie $\vec{u} \cdot \vec{v} = 0$. Plus présisément, si $\vec{u}(a, b)$ alors $\vec{v}(-b, a)$.

norme3d

$\vec{u}$ **norme3d** *r* $\longrightarrow$ *r* est la norme du vecteur $\vec{u}$

norme3d

$\vec{u}$ **norme3d** *r* $\longrightarrow$ *r* est la norme du vecteur $\vec{u}$

norme

u **norme** *r* $\longrightarrow$ le réel $r = \|\vec{u}\|$

nullc

z **nullc** *bool* $\longrightarrow$ le booléen *bool* vaut **true** si le complexe *z* est nul, **false** sinon.

O

O

– **O** O_x O_y $\longrightarrow$ dépose sur la pile les coordonnées de l'origine du repère *jps*

orange

– **orange** – $\longrightarrow$ sélectionne la couleur orange

ordonnepoints

A B **ordonnepoints** *A′ B′* ⟶ range les points *A* et *B* par ordre d'ordonnée décroissante si possible, par ordre d'abscisse décroissante sinon

ordorig

D **ordorig** *b* ⟶ *b* est l'ordonnée à l'origine de la droite *D* si celle-ci n'est pas verticale, erreur sinon

orthoprojcpath

$cpathobj_1$ *D* **orthoprojcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur la droite *D*

orthoproj

A D **orthoproj** *A′* ⟶ le point *A′* est le projeté orthogonal du point *A* sur la droite *D*

orthoprojpath

$pathobj_1$ *D* **orthoprojpath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur la droite *D*

orthoprojplane3d

M A $\vec{v}$ **orthoprojplane3d** *M′* ⟶ Le point *M′* est le projeté du point *M* sur le plan *P* défini par le point *A* et le vecteur $\vec{v}$, normal à *P*.

orthoprojplane3d

M A $\vec{v}$ **orthoprojplane3d** *M′* ⟶ Le point *M′* est le projeté du point *M* sur le plan *P* défini par le point *A* et le vecteur $\vec{v}$, normal à *P*.

outputcolors

solid string **outputcolors** – ⟶ affecte la couleur définie par *string* à toutes les faces du solide *solid*

outputcolors

solid str **outputcolors** – ⟶ affecte la couleur définie par *str* à toutes les faces du solide *solid*

ovalnode

string **ovalnode** – ⟶ Déclare un nœud ovale (elliptique) dont le nom est défini par *string*

OX

OX *D* ⟶ dépose la droite *D* = *Ox* sur la pile

OY

OY *D* ⟶ dépose la droite *D* = *Oy* sur la pile

P

pangle

A B **pangle** α ⟶ α est l'angle en dégré défini par le vecteur $\overrightarrow{AB}$ dans le repère postscript

papply3d

[$A_0 \ldots A_n$] *f* **papply3d** [$b_0 \ldots b_n$] ou – ⟶ construit un nouveau tableau en répétant, pour *i* variant de 0 à *n*, l'opération suivante : déposer le point A_i puis exécuter *f*. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

papply3d

[$A_0 \ldots A_n$] *f* **papply3d** [$b_0 \ldots b_n$] ou – ⟶ construit un nouveau tableau en répétant, pour *i* variant de 0 à *n*, l'opération suivante : déposer le point A_i puis exécuter *f*. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

papply

[$A_0 \ldots A_n$] *f* **papply** [$b_0 \ldots b_n$] ou – ⟶ construit un nouveau tableau en répétant, pour *i* variant de 0 à *n*, l'opération suivante : déposer le point A_i puis exécuter *f*. Si à la fin de cette opération le tableau est vide, alors il est enlevé de la pile.

parallelopoint

A B C **parallelopoint** *D* ⟶ le point *D* tel que *ABCD* soit un parallélogramme

paral

D A **paral** *D′* ⟶ *D′* est la droite parallèle à la droite *D* passant par le point *A*

pcangle

A B option **pcangle** – ⟶ Comme **ncangle**

pcangles

A B option **pcangles** – ⟶ Comme **ncangles**

pcarc

A B option **pcarc** $-\longrightarrow$ Comme **ncarc**

pcbar

A B option **pcbar** $-\longrightarrow$ Comme **ncbar**

pccurve

A B option **pccurve** $-\longrightarrow$ Comme **nccurve**

pcdiagg

A B option **pcdiagg** $-\longrightarrow$ Comme **ncdiagg**

pcdiag

A B option **pcdiag** $-\longrightarrow$ Comme **ncdiag**

pcline

A B option **pcline** $-\longrightarrow$ Comme **ncline**

perp

D A **perp** $D' \longrightarrow D'$ est la droite perpendiculaire à la droite D passant par le point A

pictdict

name **pictdict** $x\,y \longrightarrow$ dépose sur la pile les coordonnées associées au nom *name* dans le dictionnaire *Pictdict*

pi

$-$ **pi** $3,141\,59 \longrightarrow$ le nombre π

plot

[$A_0\ A_1 \ldots A_n$] **plot** $-\longrightarrow$ affiche les points A_i du tableau en utilisant la commande *dotstyle*

plot

[$A_0\ A_1 \ldots A_n$] *proc* **plot** $-\longrightarrow$ affiche les points A_i du tableau en utilisant la procédure *proc*

plus3d

A **plus3d** $-\longrightarrow$ Analogue 3d de la commande **plus**

plus3d

A **plus3d** $-\longrightarrow$ Analogue 3d de la commande **plus**

plus

A **plus** $-\longrightarrow$ dessine une croix + au point A dans le repère *jps*

point3d

A **point3d** $-\longrightarrow$ Analogue 3d de la commande **point**

point3d

A **point3d** $-\longrightarrow$ Analogue 3d de la commande **point**

pointilles

$-$ **pointilles** $-\longrightarrow$ sélectionne le tracé de type *tiret court*

point

A **point** $-\longrightarrow$ dessine un point en A dans le repère *jps*

point

point **point** $-\longrightarrow$ dessine le point spécifié

points3d

array **points3d** $-\longrightarrow$ Analogue 3d de la commande **points**

points3d

array **points3d** $-\longrightarrow$ Analogue 3d de la commande **points**

points

[$point_1 \ldots point_n$ **]** **points** $-\longrightarrow$ dessine les points spécifiés

Poisson

$x\ \lambda$ **Poisson** $y \longrightarrow y$ est l'image de x par la loi de Poisson de paramètre λ

pol2ell

pol **pol2ell** *ell* $\longrightarrow$ le polygône *pol* est constitué des 4 sommets de l'ellipse *ell*

polygone*3d

array **polygone*3d** $-\longrightarrow$ Analogue 3d de la commande **polygone***

polygone*3d

array **polygone*3d** – ⟶ Analogue 3d de la commande **polygone***

polygone3d

array **polygone3d** – ⟶ Analogue 3d de la commande **polygone**

polygone3d

array **polygone3d** – ⟶ Analogue 3d de la commande **polygone**

polygone_

array **polygone_** – ⟶ ajoute au chemin courant le polygône défini par le tableau de points *array*

polygone

array **polygone** – ⟶ trace le polygône défini par le tableau de points *array*

popc

cerc **popc** – ⟶ enlève le cercle au sommet de la pile

popd

D **popd** – ⟶ enlève la droite au sommet de la pile

popp

A **popp** – ⟶ enlève le point au sommet de la pile

printmatrix

A x y M **printmatrix** – ⟶ Affiche la matrice M. Le coefficient a_{00} est affiché en A, et on utilise les décalages $(x, 0)$ et $0, y$ pour les autres coefficients

projpath

solid n α *str bool* **projpath** - ⟶ Projette le chemin courant sur le plan affine défini par la face d'indice *n* du solide *solid*. Le booléen optionnel *bool* sert à préciser si on doit ou non tenir compte de la visibilité (*true* par défaut). La chaîne de caractères optionnelle *str* permet de préciser l'origine du plan de projection (centre de la face de projection par défaut). L'angle optionnel α permet de préciser la rotation autour de la normale souhaitée.

projpath

$x_0\ y_0\ z_0$ **[** $i_1\ i_2\ i_3\ k_1\ k_2\ k_3$ **]** *bool* **projpath** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est donnée par $\vec{\imath}'(i_1, i_2, i_3)$, image du vecteur $(1, 0, 0)$ ar cette projection. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

projpath

$x_0\ y_0\ z_0$ **[** $k_1\ k_2\ k_3\ \alpha$ **]** *bool* **projpath** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$, après avoir fait subir une rotation d'angle α degrés autour de la normale par rapport à l'orientation originellement proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

projpath

$x_0\ y_0\ z_0$ **[** $k_1\ k_2\ k_3$ **]** *bool* **projpath** - ⟶ Projette le chemin courant sur le plan affine d'origine (x_0, y_0, z_0) et de normale le vecteur $\vec{k}'(k_1, k_2, k_3)$. L'orientation est proposée par le programme. Le booléen optionnel *bool* permet de spécifier s'il faut ou non tenir compte de la visibilité du plan de projection.

projxcpath

$cpathobj_1$ **projxcpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe Ox

projx

A **projx** A' ⟶ le point A' est le projeté orthogonal du point A sur l'axe Ox

projxpath

$pathobj_1$ **projxpath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur l'axe Ox

projycpath

$cpathobj_1$ **projycpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu projeté orthogonal de $cpathobj_1$ sur l'axe Oy

projy

A **projy** A' ⟶ le point A' est le projeté orthogonal du point A sur l'axe Oy

projypath

$pathobj_1$ **projypath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin projeté orthogonal de $pathobj_1$ sur l'axe Oy

ptangente+

t {*X*} {*Y*} **ptangente+** – ⟶ trace la demi-tangente à droite pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre *t*.

ptangente-

t {*X*} {*Y*} **ptangente-** – ⟶ trace la demi-tangente à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre *t*.

ptangente

t {*X*} {*Y*} **ptangente** – ⟶ trace les demi-tangentes à droite et à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre *t*.

ptangente

t (*X*) (*Y*) **ptangente** – ⟶ trace les demi-tangentes à droite et à gauche pour la courbe de la fonction $t \mapsto \big(X(t), Y(t)\big)$ au point de paramètre *t* en utilisant les fonctions X' et Y' pour le calcul du vecteur dérivé.

ptojpoint

x y **ptojpoint** *X Y* ⟶ Reçoit les coordonnées (x, y) dans le repère postscript et renvoie les coordonnées (X, Y) dans le repère *jps*

put_ij

M i j any **put_ij** – ⟶ affecte le coefficient a_{ij} de la matrice *M* à *any*

put_Li

M i L **put_Li** – ⟶ remplace dans la matrice *M* la ligne d'indice *i* par *L*

Q

qplanxy

– **qplanxy** – ⟶ Trace un quadrillage du plan XY

qplanxz

– **qplanxz** – ⟶ effectue un quadrillage du plan *xOz*

qplanyz

– **qplanyz** – ⟶ effectue un quadrillage du plan *yOz*

quadrillagegray

quadrillagegray ⟶ niveau de gris pour le quadrillage tracé avec la commande **quadrillage**. **valeur par défaut :** 0, 4

quadrillage

– **quadrillage** – ⟶ Trace un quadrillage simple. Les paramètres sont *xstquadrillage*, *ystquadrillage*, *quadrillagegray* et *quadrillagewd*

Quadrillage

xs$_0$ *ys*$_0$ { *color* } **Quadrillage** – ⟶ Trace un quadrillage simple, de couleur *color*, avec les pas sur *x* et *y* définis par le couple (xs_0, ys_0). L'argument $\{color\}$ est optionnel. Avec cette syntaxe, l'épaisseur du trait est l'épaisseur courante.

Quadrillage

[*xs*$_0$ *ys*$_0$ *xs*$_1$ *ys*$_1$ *xs*$_2$ *ys*$_2$] { *color* } **Quadrillage** – ⟶ Trace un triple quadrillage, de couleur *color*, avec les pas sur *x* et *y* définis par les xs_i et ys_i. L'argument $\{color\}$ est optionnel, de même que les couples (xs_2, ys_2) et (xs_1, ys_1). Les épaisseurs de trait sont relevées dans le tableau *quadrillagewidth*

Quadrillage

[*xs*$_0$ *ys*$_0$ *xs*$_1$ *ys*$_1$ *xs*$_2$ *ys*$_2$] { *color* } **Quadrillage** – ⟶ Trace un triple quadrillage, de couleur *color*, avec les pas sur *x* et *y* définis par les xs_i et ys_i. L'argument $\{color\}$ est optionnel, de même que les couples (xs_2, ys_2) et (xs_1, ys_1). Les épaisseurs de trait sont relevées dans le tableau *quadrillagewidth*

quadrillagewd

quadrillagewd ⟶ épaisseur du trait pour le quadrillage tracé avec la commande **quadrillage**. **valeur par défaut :** 0, 25

Quadrillagewidth

Quadrillagewidth ⟶ tableau définissant les 3 épaisseurs de traits pour le quadrillage triple tracé par la commande **Quadrillage**. **valeur par défaut :** [0, 7 0, 4 0, 2]

quadrilleXYZ

xmin xmax ymin ymax zmin zmax **quadrilleXYZ** – ⟶ Effectue un quadrillage d'unité 1 sur le produit $[xmin; xmax] \times [ymin; ymax] \times [zmin; zmax]$

R

rand

– **rand** *int* ⟶ génère un entier au hasard

rcurveto

$dx_1\ dy_1\ dx_2\ dy_2\ dx_3\ dy_3$ **rcurveto** – ⟶ **curveto** relatif

rect

A **rect** – ⟶ colorie en niveau de gris un rectangle dont une base est porté par l'axe *Ox*, et tel que le point *A* soit le milieu du côté opposé

rect

A **rect** – ⟶ dessine un rectangle dont une base est porté par l'axe *Ox*, et tel que le point *A* soit le milieu du côté opposé

regxy

$array_1\ array_2$ **regxy** *d* ⟶ *d* est la droite de régression des *x* en *y* de la série double définie par les tableaux de réels $array_1$ et $array_2$

regyx

$array_1\ array_2$ **regyx** *d* ⟶ *d* est la droite de régression des *y* en *x* de la série double définie par les tableaux de réels $array_1$ et $array_2$

repeat

int proc **repeat** – ⟶ exécute *proc int* fois

representationtype

representationtype ⟶ Chaîne de caractère spécifiant le type de perspective : (perspective) ou (ortho). **valeur par défaut :** (perspective)

reversecpathobj

$cpathobj_1$ **reversecpathobj** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu obtenu à partir de $cpathobj_1$ en inversant le sens de parcours

reversepath

– **reversepath** – ⟶ renverse la direction du chemin courant

rframe_

A L ℓ **rframe_** – ⟶ ajoute au chemin courant le rectangle dont le point *A* est le coin inférieur droit, de dimension horizontale *L* et de dimension verticale ℓ

rframe

A L ℓ **rframe** – ⟶ trace le rectangle dont le point *A* est le coin inférieur droit, de dimension horizontale *L* et de dimension verticale ℓ

right

– **right** $\vec{u}$ ⟶ $\vec{u}$ est le vecteur $(1, 0)$

rlineto

dx dy **rlineto** – ⟶ **lineto** relatif

rmoveto

dx dy **rmoveto** – ⟶ **moveto** relatif

Rnode

string **Rnode** – ⟶ Déclare un nœud rectangulaire avec un encadrement non dessiné et dont le nom est défini par *string*

rollp

n p **rollp** – ⟶ considère la pile comme une file circulaire de *n* points, et la tourne de *p* crans

romberg

a b $\{f\}$ ε **romberg** *real* ⟶ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode de Romberg pour une valeur de convergence fixée à ε

rootnode

x y tnode **rootnode** *tnode* ⟶ Dépose les coordonnées (x, y) dans le champ adapté du nœud d'arbre *tnode*, et le nomme *A* si celui-ci n'a pas de nom.

rose

– **rose** – ⟶ sélectionne la couleur rose

rotatecercle

cerc *I* α **rotatecercle** *cerc′* ⟶ le cercle *cerc′* est l'image du cercle *cerc* par la rotation de centre *I* et d'angle α

rotatecpath

$cpathobj_1$ *I* α **rotatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la rotation de centr(e *I* et d'angle α

rotatedroite

D *I* α **rotatedroite** *D′* ⟶ la droite *D′* est l'image de la droite *D* par la rotation de centre *I* et d'angle α

rotateell

ell *I* α **rotateell** *ell′* ⟶ l'ellipse *ell′* est l'image de l'ellipse *ell* par la rotation de centre *I* et d'angle α

rotateOpoint3d

M α_x α_y α_z **rotateOpoint3d** *M′* ⟶ *M′* est l'image de *M* par la rotation de centre *O* et d'angles respectifs α_x α_y α_z sur les axes *Ox*, *Oy*, *Oz*

rotateOpoint3d

M α_x α_y α_z **rotateOpoint3d** *M′* ⟶ *M′* est l'image de *M* par la rotation de centre *O* et d'angles respectifs α_x α_y α_z sur les axes *Ox*, *Oy*, *Oz*

rotatepath

$pathobj_1$ *I* α **rotatepath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin image de $pathobj_1$ par la rotation de centr(e *I* et d'angle α

rotatepoint

A *I* α **rotatepoint** *A′* ⟶ le point *A′* est l'image du point *A* par la rotation de centre *I* et d'angle α

rotatepol

pol *I* α **rotatepol** *pol′* ⟶ le polygône *pol′* est l'image du polygône *pol* par la rotation de centre *I* et d'angle α

rouge

– **rouge** – ⟶ sélectionne la couleur rouge

round

num_1 **round** num_2 ⟶ arrondit num_1 à l'entier le plus proche

#rpn#

#rpn# $expr_1$ ⟶ $expr_2$: $expr_2$ est l'écriture en notation polonaise inverse de l'expression en notation cartésienne $expr_1$

rptojpoint

x *y* **rptojpoint** *X* *Y* ⟶ Reçoit les coordonnées (x, y) dans le repère BB (*real postscript*) et renvoie les coordonnées (X, Y) dans le repère *jps*

rtp2xyz

r θ ϕ **rtp2xyz** *x* *y* *z* ⟶ Passage des coordonnées sphériques vers les coordonnées cartésiennes

rtp2xyz

r θ ϕ **rtp2xyz** *x* *y* *z* ⟶ Passage des coordonnées sphériques vers les coordonnées cartésiennes

Rungekutta

a *b* {*f*} x_0 y_0 *h* **Rungekutta** x_{-n} y_{-n} ... x_{-1} y_{-1} x_0 y_0 x_1 y_1 ... x_n y_n ⟶ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[a, b]$ de la fonction *s* solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre *h* est un réel positif, il détermine le pas entre chaque point calculé

Rungekutta

a *b* {*f*} x_0 y_0 *n* **Rungekutta** x_{-n} y_{-n} ... x_{-1} y_{-1} x_0 y_0 x_1 y_1 ... x_n y_n ⟶ *n* étant un entier, cette procédure dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[a, b]$ de la fonction *s* solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre *h* est calculé en fonction de *n*.

rungekutta

{*f*} x_0 y_0 *h* **rungekutta** x_{-n} y_{-n} ... x_{-1} y_{-1} x_0 y_0 x_1 y_1 ... x_n y_n ⟶ dépose les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[xmin, xmax]$ de la fonction *s* solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre *h* est un réel positif, il détermine le pas entre chaque point calculé

rungekutta

{*f*} x_0 y_0 *n* **rungekutta** x_{-n} y_{-n} ... x_{-1} y_{-1} x_0 y_0 x_1 y_1 ... x_n y_n ⟶ *n* étant un entier, cette procédure dépose

les points, calculés par la méthode de Runge-Kutta, de la courbe sur $[xmin, xmax]$ de la fonction s solution de l'équation différentielle $y' = f(x, y)$ vérifiant $s(x_0) = y_0$. Le nombre h est calculé en fonction de n.

S

sarc

$x\ y\ r\ ang_1\ ang_2$ **sarc** $-\longrightarrow$ ajoute un arc dans le sens contraire des aiguilles d'une montre (coordonnées dans le repère jps)

sarcn

$x\ y\ r\ ang_1\ ang_2$ **sarcn** $-\longrightarrow$ ajoute un arc dans le sens des aiguilles d'une montre (coordonnées dans le repère jps)

scaleOpoint3d

$x\ y\ z\ k_1\ k_2\ k_3$ **scaleOpoint3d** $k_1x\ k_2y\ k_3z \longrightarrow$ opère une « dilatation » des coordonnées du point $M(x, y, z)$ sur les axes Ox, Oy et Oz suivant les facteurs k_1, k_2 et k_3

scaleOpoint3d

$x\ y\ z\ k_1\ k_2\ k_3$ **scaleOpoint3d** $k_1x\ k_2y\ k_3z \longrightarrow$ opère une « dilatation » des coordonnées du point $M(x, y, z)$ sur les axes Ox, Oy et Oz suivant les facteurs k_1, k_2 et k_3

scalprod3d

$\vec{u}\ \vec{v}$ **scalprod3d** $s \longrightarrow$ Produit scalaire : $s = \vec{u} \cdot \vec{v}$

scalprod3d

$\vec{u}\ \vec{v}$ **scalprod3d** $s \longrightarrow$ Produit scalaire : $s = \vec{u} \cdot \vec{v}$

scalprod

$\vec{u}\ \vec{v}$ **scalprod** $\vec{u} \cdot \vec{v} \longrightarrow$ Le produit scalaire de $\vec{u}$ par $\vec{v}$

ScreenDist

ScreenDist $\longrightarrow$ Distance par rapport à l'écran. **valeur par défaut :** 0.1

scurveto

$x_1\ y_1\ x_2\ y_2\ x_3\ y_3$ **scurveto** $-\longrightarrow$ ajoute une section cubique de Bézier (coordonnées dans le repère jps)

setangle_repere

α **setangle_repere** $-\longrightarrow$ Instruction destinée au script $jps2ps$. Elle indique que l'angle entre les axes Ox et Oy est de α degrés. Attention, α doit être lisible en clair par le script

setborder

b **setborder** $-\longrightarrow$ Indique, en points postripts, a taille de la bordure entourant l'image. Cette instruction est interprétée par le script $jps2ps$ pour déterminer la BoundingBox.

SetCamPos

$x\ y\ z$ **SetCamPos** $-\longrightarrow$ Positionne la caméra au point (x, y, z)

SetCamUp

$U_x\ U_y\ U_z$ **SetCamUp** $-\longrightarrow$ *Set Camera Up vector.*

SetCamVec

$V_x\ V_y\ V_z$ **SetCamVec** $-\longrightarrow$ *Set Camera Looking vector.*

SetCamView

M **SetCamView** $-\longrightarrow$ Oriente la visée de la caméra vers le point $M(x, y, z)$ et recalcule tous les vecteurs nécessaires

setcmykcolor

cyan magenta yellow black **setcmykcolor** $-\longrightarrow$ définit la couleur CMYK. Les nombres *red*, *green* et *blue* sont des réels compris entre 0 et 1

setCourierBoldItalic

– **setCourierBoldItalic** $-\longrightarrow$ sélectionne la police CourierBoldItalic

setCourierBold

– **setCourierBold** $-\longrightarrow$ sélectionne la police CourierBold

setCourierItalic

– **setCourierItalic** $-\longrightarrow$ sélectionne la police CourierItalic

setCourier

– **setCourier** $-\longrightarrow$ sélectionne la police Courier

setellipseangle

α **setellipseangle** $-\longrightarrow$ affecte la valeur α au paramètre *ellipseangle*

setfontsize

n **setfontsize** $-\longrightarrow$ affecte la valeur n au paramètre $fontsize$

setformat

k **setformat** $-\longrightarrow$ Instruction destinée au script $jps2ps$. Elle indique que k est le rapport entre largeur et hauteur de l'image. Attention, k doit être lisible en clair par le script

setframeangle

α **setframeangle** $-\longrightarrow$ affecte la valeur α au paramètre $frameangle$

setheight

ℓ **setheight** $-\longrightarrow$ Affecte la valeurs ℓ à la variable *height* (taille verticale, en points poscripts, de la fenêtre de dessin). La première occurence de cette instruction dans le fichier jps est interprétée par le script pour déterminer la BoundingBox.

setlightintensity

i **setlightintensity** $-\longrightarrow$ Affecte l'intensité de la source lumineuse ponctuelle

setlight

array **setlight** $-\longrightarrow$ Affecte la couleur de la source lumineuse ponctuelle dans l'espace RGB ou CYMK, *array* étant un tableau de 3 ou 4 réels de l'intervalle $[0;1]$

setlight

$\{f\}$ **setlight** $-\longrightarrow$ Éxécute la fonction f dans un **gsave .. grestore**, y récupère la définition de la couleur dans l'espace RGB, puis l'affecte à la couleur de la source lumineuse ponctuelle.

setlightsrc

$x\ y\ z$ **setlightsrc** $-\longrightarrow$ Affecte la position de la source lumineuse ponctuelle

setmkstep

$t_1\ t_2$ **setmkstep** $-\longrightarrow$ définit respectivement les pas t_1 et t_2 pour la numérotation sur les axes Ox et Oy

setorigine

$x\ y$ **setorigine** $-\longrightarrow$ affecte les coordonnées (x,y) au point origine du repère jps

setPalatinoBoldItalic

– **setPalatinoBoldItalic** $-\longrightarrow$ sélectionne la police PalatinoBoldItalic

setPalatinoBold

– **setPalatinoBold** $-\longrightarrow$ sélectionne la police PalatinoBold

setPalatinoItalic

– **setPalatinoItalic** $-\longrightarrow$ sélectionne la police PalatinoItalic

setPalatino

– **setPalatino** $-\longrightarrow$ sélectionne la police Palatino

setquadrillagegray

g **setquadrillagegray** $-\longrightarrow$ affecte la valeur g au paramètre *quadrillagegray*. On doit avoir $0 \leqslant g \leqslant 1$

setresolution

n **setresolution** $-\longrightarrow$ affecte l'entier n à la variable *resolution* qui contrôle le nombre de points de calculs pour la représentation d'une courbe de fonction numérique

setrgbcolor

red green blue **setrgbcolor** $-\longrightarrow$ définit la couleur RGB. Les nombres *red*, *green* et *blue* sont des réels compris entre 0 et 1

setrotate

α **setrotate** $-\longrightarrow$ Instruction destinée au script $jps2ps$. Elle indique que l'image finale devra subir une rotation d'angle α (en degrés). Le calcul de la BoundingBox tient compte de cette rotation. Attention, α doit être lisible en clair par le script

setsubtkstep

$t_1\ t_2$ **setsubtkstep** $-\longrightarrow$ définit respectivement les pas t_1 et t_2 pour les sous-tirets sur les axes Ox et Oy

setSymbolBoldItalic

– **setSymbolBoldItalic** $-\longrightarrow$ sélectionne la police SymbolBoldItalic

setSymbolBold

– **setSymbolBold** – ⟶ sélectionne la police SymbolBold

setSymbolItalic

– **setSymbolItalic** – ⟶ sélectionne la police SymbolItalic

setSymbol

– **setSymbol** – ⟶ sélectionne la police Symbol

settailletangente

x **settailletangente** – ⟶ affecte la valeur réelle *a* à la variable *tailletangente* qui gère la taille des tangentes tracées par **tangente**. La taille est exprimée en unités de l'axe *Ox*

setTimesBoldItalic

– **setTimesBoldItalic** – ⟶ sélectionne la police TimesBoldItalic

setTimesBold

– **setTimesBold** – ⟶ sélectionne la police TimesBold

setTimesItalic

– **setTimesItalic** – ⟶ sélectionne la police TimesItalic

setTimes

– **setTimes** – ⟶ sélectionne la police Times

settkstep

t_1 t_2 **settkstep** – ⟶ définit respectivement les pas t_1 et t_2 pour les tirets sur les axes *Ox* et *Oy*

settrange

a b **settrange** – ⟶ affecte respectivement les valeurs réelles *a* et *b* aux variables *tmin* et *tmax*

settvar

a **settvar** – ⟶ affecte la valeur réelle *a* à la variable *t*

setwidth

L **setwidth** – ⟶ Affecte la valeurs *L* à la variable *width* (taille horizontale, en points postscript, de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

setxmkstep

t **setxmkstep** – ⟶ définit le pas pour la numérotation sur l'axe *Ox*

setxrange3d

xmin xmax **setxrange3d** – ⟶ affecte les variables *xmin*3*d* et *xmax*3*d* définissant l'intervalle de travail sur l'axe *Ox* en 3*d*

setxrange

x_1 x_2 **setxrange** – ⟶ Affecte respectivement les valeurs x_1 et x_2 à *xmin* et *xmax* (amplitude horizontale de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

setxstquadrillage

p **setxstquadrillage** – ⟶ affecte la valeur *p* au paramètre *xstquadrillage*

setxsubtkstep

t **setxsubtkstep** – ⟶ définit le pas pour les sous-tirets sur l'axe *Ox*

setxtkstep

t **setxtkstep** – ⟶ définit le pas pour les tirets sur l'axe *Ox*

setxunit

u **setxunit** – ⟶ Spécifie, en nombre de points postcripts par unité, l'échelle sur l'axe *Ox*. La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

setxvar

a **setxvar** – ⟶ affecte la valeur réelle *a* à la variable *x*

setxyrapport

α **setxyrapport** – ⟶ Instruction destinée au script *jps*2*ps*. Elle indique que le rapport entre l'unité sur *Ox* et l'unité sur *Oy* est α. Attention, α doit être lisible en clair par le script

setymkstep

t **setymkstep** – ⟶ définit le pas pour la numérotation sur l'axe *Oy*

setyrange3d

ymin ymax **setyrange3d** $-\longrightarrow$ affecte les variables *ymin*3*d* et *ymax*3*d* définissant l'intervalle de travail sur l'axe *Oy* en 3*d*

setyrange

$y_1\ y_2$ **setyrange** $-\longrightarrow$ Affecte respectivement les valeurs y_1 et y_2 à *ymin* et *ymax* (amplitude verticale de la fenêtre de dessin). La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

setystquadrillage

p **setystquadrillage** $-\longrightarrow$ affecte la valeur *p* au paramètre *ystquadrillage*

setysubtkstep

t **setysubtkstep** $-\longrightarrow$ définit le pas pour les sous-tirets sur l'axe *Oy*

setytkstep

t **setytkstep** $-\longrightarrow$ définit le pas pour les tirets sur l'axe *Oy*

setyunit

v **setyunit** $-\longrightarrow$ Spécifie, en nombre de points postcripts par unité, l'échelle sur l'axe *Oy*. La première occurence de cette instruction dans le fichier *jps* est interprétée par le script pour déterminer la BoundingBox.

setzrange3d

zmin zmax **setzrange3d** $-\longrightarrow$ affecte les variables *zmin*3*d* et *zmax*3*d* définissant l'intervalle de travail sur l'axe *Oz* en 3*d*

simpson

a b {*f*} *n* **simpson** *real* $\longrightarrow$ *real* est une approximation de l'intégrale de $f(x)$ entre *a* et *b*, calculée avec la méthode de Simpson pour $n+1$ points (n est un entier pair)

sinh

a **sinh** $c \longrightarrow c = \mathrm{sh}\, a$

sin

a **sin** $c \longrightarrow c = \sin a$ (*a* en degré)

Sin

a **Sin** $c \longrightarrow c = \sin a$ (*a* en radian)

slineto

x y **slineto** $-\longrightarrow$ ajoute une ligne droite jusqu'en (x, y) dans le repère *jps*

smoveto

x y **smoveto** $-\longrightarrow$ définit le point courant à (x, y) dans le repère *jps*

smulm

M α **smulm** $M' \longrightarrow M'$ est la matrice produit de la matrice *M* par le scalaire α

solidcentreface

solid i **solidcentreface** $G \longrightarrow$ le point *G* est le centre de la face d'indice *i* du solide *solid*

solidfacevisible?

solid i **solidfacevisible?** *bool* $\longrightarrow$ *bool* vaut *true* si la face d'indice *i* du solide *solid* est visible

solidfuz

$solid_1\ solid_2$ **solidfuz** *solid* $\longrightarrow$ dépose sur la pile le solide obtenu à partir de la fusion des solides $solid_1$ et $solid_2$

solidgetface

solid i **solidgetface** *array* $\longrightarrow$ *array* est le tableau décrivant la face *i* du solide (tableau d'indices de sommets)

solidgetfaces

solid **solidgetfaces** *array* $\longrightarrow$ *array* est le tableau des faces du solide

solidgetfcolors

solid **solidgetfcolors** *array* $\longrightarrow$ le tableau *array* est le tableau des couleurs des faces pour le solide *solid*

solidgetpointstable

solid **solidgetpointstable** *array* $\longrightarrow$ *array* est le tableau des sommets du solide

solidgetsommetface

solid i j **solidgetsommetface** $S \longrightarrow S$ est le sommet d'indice *i* de la face d'indice *j* du solide *solid*

solidgetsommet

solid i **solidgetsommet** $S \longrightarrow S$ est le sommet d'indice *i* du solide *solid*

solidgetsommetsface

solid i **solidgetsommetsface** *array* $\longrightarrow$ *array* est le tableau des sommets de la face d'indice *i* du solide *solid*

solidnombrefaces

solid **solidnombrefaces** $n \longrightarrow$ nombres de faces du solide *solid*

solidnombresommets

solid **solidnombresommets** $n \longrightarrow$ nombres de sommets du solide *solid*

solidnormaleface

solid i **solidnormaleface** $\vec{u} \longrightarrow \vec{u}$ est un vecteur normal à la face d'indice *i* du solide *solid*. Le vecteur est orientaté vers l'extérieur du solide

solidnumfaces

solid array bool **solidnumfaces** $- \longrightarrow$ Si *array* est présent, numérote les faces désignées dans le tableau d'indices *array*. Numérote toutes les faces sinon. Le booléen optionnel *bool* permet de définir si on doit ou non tenir compte de la visibilité de la face considérée.

solidnumsommets

solid array **solidnumsommets** $- \longrightarrow$ Si *array* est présent, numérote les sommets désignés dans le tableau d'indices *array*. Numérote tous les sommets sinon.

solidputfaces

solid array **solidputfaces** $- \longrightarrow$ Remplace le tableau des faces du solide *solid* par le tableau *array*

solidputfcolors

solid array **solidputfcolors** $- \longrightarrow$ affecte au solide *solid* le tableau *array* en tant que tableau des couleurs des faces

solidputhuecolors

solid array **solidputhuecolors** $- \longrightarrow$ Affecte les couleurs des faces externes du solide *solid* selon le dégradé désigné par *array*

solidputinhuecolors

solid array **solidputinhuecolors** $- \longrightarrow$ Affecte les couleurs des faces internes du solide *solid* selon le dégradé désigné par *array*

solidputinouthuecolors

solid array **solidputinouthuecolors** $- \longrightarrow$ Affecte les couleurs de l'ensemble des faces externes et internes du solide *solid* selon le dégradé désigné par *array*

solidputpointstable

solid array **solidputpointstable** $- \longrightarrow$ Remplace le tableau des sommets du solide *solid* par le tableau *array*

solidrmface

solid i **solidrmface** $- \longrightarrow$ enlève la face d'indice *i* du solide *solid*

solidrmfaces

solid array **solidrmfaces** $- \longrightarrow$ enlève du solide *solid* toutes les faces définies par le tableau d'indices *array*

solidshowsommets

solid array **solidshowsommets** $- \longrightarrow$ Si *array* est présent, pointe les sommets désignés dans le tableau d'indices *array*. Pointe tous les sommets sinon.

solidtransform

solid $\{f\}$ **solidtransform** *solid* $\longrightarrow$ applique la transformation f au solide *solid*, f étant une application $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$

solve2nddegre

a b c **solve2nddegre** $x_1\ x_2 \longrightarrow$ Les réels x_1 et x_2 sont les racines réelles de l'équation $ax^2 + bx + c = 0$, où $a \neq 0$ et où $b^2 - 4ac \geqslant 0$

solve_syst

$B\,A$ **solve_syst** $X \longrightarrow A$ est une matrice carrée de déterminant non nul, B est un vecteur colonne, et X est le vecteur colonne solution de l'équation matricielle $AX = B$

solve_syst

$B\,A$ **solve_syst** $X \longrightarrow A$ est une matrice carrée de déterminant non nul et X est l'unique vecteur solution de l'équation $AX = B$

solve_trig

$B\,A$ **solve_trig** $X \longrightarrow A$ est une matrice triangulaire supérieure et X est l'unique vecteur solution de l'équation $AX = B$

splitcpath

t *cpathobj* **splitcpath** $cpathobj_1\ cpathobj_2 \longrightarrow$ sépare, à partir du point de paramètre t, le chemin continu paramétré *cpathobj* en 2 sous-chemins paramétrés

sqrt

a **sqrt** $c \longrightarrow c = \sqrt{a}$

square

A **square** $- \longrightarrow$ dessine un carré au point A dans le repère jps

srcurveto

$dx_1\ dy_1\ dx_2\ dy_2\ dx_3\ dy_3$ **srcurveto** $- \longrightarrow$ **scurveto** relatif

srlineto

$dx\ dy$ **srlineto** $- \longrightarrow$ **slineto** relatif

srmoveto

$dx\ dy$ **srmoveto** $- \longrightarrow$ **smoveto** relatif

stockcurrentcpath

$-$ **stockcurrentcpath** $- \longrightarrow$ sauvegarde le chemin continu courant sous forme de chemin continu paramétré, et réaffecte **lastcpath** et **lastcppath** en conséquence

subc

$z\ z'$ **subc** $Z \longrightarrow Z = z - z'$ est la différence des complexes z et z'

sub

$a\ b$ **sub** $c \longrightarrow c = a - c$

subticks

$-$ **subticks** $- \longrightarrow$ trace tous les sous-tirets des axes Ox et Oy

subv3d

$\vec{u}\ \vec{v}$ **subv3d** $\vec{w} \longrightarrow \vec{w} = \vec{u} + \vec{v}$

subv3d

$\vec{u}\ \vec{v}$ **subv3d** $\vec{w} \longrightarrow \vec{w} = \vec{u} + \vec{v}$

subv

$u\ u'$ **subv** $\vec{U} \longrightarrow \vec{U} = \vec{u} - \vec{u}'$ est la différence des vecteurs $\vec{u}$ et $\vec{u}'$

sum

$[\ a_0 \ldots a_n\ ]$ **sum** $s \longrightarrow$ le réel s est la somme $s = \sum_{i=0}^{n} a_i$

surfaceparam3d

$xmin\ pas_x\ xmax\ ymin\ pas_y\ ymax\ f$ **surfaceparam3d** $- \longrightarrow$ Dessine la surface $f(x, y) = z$ sur $[xmin; xmax] \times [ymin; ymax]$. f est un éxécutable.

symcercle

$cerc\ I$ **symcercle** $cerc' \longrightarrow$ le cercle $cerc'$ est le symétrique du cercle $cerc$ par rapport au point I

symcpath

$cpathobj_1\ I$ **symcpath** $cpathobj_2 \longrightarrow cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la symétrie de centre I

symell

$ell\ I$ **symell** $ell' \longrightarrow$ l'ellipse ell' est la symétrique de l'ellipse ell par rapport au point I

sympath

$pathobj_1\ I$ **sympath** $pathobj_2 \longrightarrow pathobj_2$ est le chemin image de $pathobj_1$ par la symétrie de centre I

sympoint3d

M A **sympoint3d** $M' \longrightarrow M'$ est l'image de *M* par la symétrie de centre *A*

sympoint3d

M A **sympoint3d** $M' \longrightarrow M'$ est l'image de *M* par la symétrie de centre *A*

sympoint

A I **sympoint** $A' \longrightarrow$ le point A' est le symétrique du point *A* par rapport au point *I*

sympol

pol I **sympol** $pol' \longrightarrow$ le polygône pol' est le symétrique du polygône *pol* par rapport au point *I*

T

tab3dto2d

*array*1 **tab3dto2d** *array*2 $\longrightarrow$ transforme un tableau de points 3d en tableau de points 2d

tailletangente

tailletangente $\longrightarrow$ la taille, exprimée en unités de l'axe *Ox*, des tangentes tracées par **tangente**.. **valeur par défaut :** 1

tangente+

x {*f*} **tangente+** – $\longrightarrow$ trace la demi-tangente à droite pour la courbe de la fonction *f* au point d'abscisse *x*.

tangente-

x {*f*} **tangente-** – $\longrightarrow$ trace la demi-tangente à gauche pour la courbe de la fonction *f* au point d'abscisse *x*.

tangente

x {*f*} **tangente** – $\longrightarrow$ trace les 2 demi-tangentes à gauche et à droite pour la courbe de la fonction *f* au point d'abscisse *x*.

tangente

x string **tangente** – $\longrightarrow$ trace la tangente à la courbe de la fonction *f* au point d'abscisse *x*, où *f* est l'exécutable désigné par la chaîne de caractères *string*. Attention, le calcul utilise l'exécutable f' dont le nom est obtenu en adjoignant à la chaîne *string* le caractère **'**. L'exécutable f' doit donc être défini.

tanh

a **tanh** $c \longrightarrow c = \text{th}\, a$

tan

a **tan** $c \longrightarrow c = \tan a$ (*a* en degré)

Tan

a **Tan** $c \longrightarrow c = \tan a$ (*a* en radian)

Tbc

string/*lit* **Tbc** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire encadré dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tb

string/*lit* **Tb** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire encadré dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

TCc

string/*lit* **TCc** *tnode* $\longrightarrow$ Construit un nœud d'arbre circulaire de rayon fixe *Circleradius* dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tcc

string/*lit* **Tcc** *tnode* $\longrightarrow$ Construit un nœud d'arbre circulaire dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

TC

string/*lit* **TC** *tnode* $\longrightarrow$ Construit un nœud d'arbre circulaire de rayon fixe *Circleradius* dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tc

string/*lit* **Tc** *tnode* $\longrightarrow$ Construit un nœud d'arbre circulaire dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tdiac

string/*lit* **Tdiac** *tnode* $\longrightarrow$ Construit un nœud d'arbre en forme de losange et dont le contenu, centré, est soit

string, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tdia

string/*lit* **Tdia** *tnode* ⟶ Construit un nœud d'arbre en forme de losange et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tdot

– **Tdot** *tnode* ⟶ Construit un nœud d'arbre contenant un point (en utilisant la commande **dot**)

#tex#

#tex# *expr* ⟶ – : compile l'expression *expr* avec TEX et affecte le résultat pour l'utilisation des commandes *labeltex*

Tf

– **Tf** *tnode* ⟶ Construit un nœud d'arbre fantôme (contenu vide et non relié au nœud père)

ticks

– **ticks** – ⟶ trace tous les tirets des axes *Ox* et *Oy*

times

A **times** – ⟶ dessine une croix au point *A* dans le repère *jps*

tnparametres

tnode proc **tnparametres** *tnode* ⟶ Définit les paramètres du nœud d'arbre *tnode* par *proc*

Toptaff

Toptaff *tnode array* ⟶ *tnode* : Applique au nœud d'arbre *tnode* les transformations décrites par le tableau *array*, qui est du type **[** (*dx dy*) [$scale_x scale_y$] {α} **]**

Tovalc

string/*lit* **Tovalc** *tnode* ⟶ Construit un nœud d'arbre en forme d'ellipse et dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Toval

string/*lit* **Toval** *tnode* ⟶ Construit un nœud d'arbre en forme d'ellipse et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

traceaxes

– **traceaxes** – ⟶ trace les axes *Ox* et *Oy*

traceOx

– **traceOx** – ⟶ trace l'axe *Ox*

traceOy

– **traceOy** – ⟶ trace l'axe *Oy*

tracerepere

– **tracerepere** – ⟶ trace les axes *Ox* et *Oy*, des flèches au bout des axes, ainsi que des flèches unités

trait

A B α **trait** – ⟶ calcule le point A' image de *A* par l'homothétie de centre *B* et de rapport $|\alpha|$, puis le point B' image de *B* par l'homothétie de centre *A* et de rapport $|\alpha|$. Si α est positif, trace la commande est équivalente à **[A' B'] ligne**, et si α est négatif, alors la commande invoque le tracé de la droite $(A'B')$ privée du segment $[A'B']$.

translatecercle

cerc $\vec{u}$ **translatecercle** *cerc'* ⟶ le cercle *cerc'* est l'image du cercle *cerc* par la translation de vecteur $\vec{u}$

translatecpath

$cpathobj_1$ *u* **translatecpath** $cpathobj_2$ ⟶ $cpathobj_2$ est le chemin continu image de $cpathobj_1$ par la translation de vecteur *u*

translatedroite

D $\vec{u}$ **translatedroite** D' ⟶ la droite D' est l'image de la droite *D* par la translation de vecteur $\vec{u}$

translateell

ell $\vec{u}$ **translateell** *ell'* ⟶ l'ellipse *ell'* est l'image de l'ellipse *ell* par la translation de vecteur $\vec{u}$

translatepath

$pathobj_1$ *u* **translatepath** $pathobj_2$ ⟶ $pathobj_2$ est le chemin image de $pathobj_1$ par la translation de vecteur *u*

translatepoint3d

M u **translatepoint3d** $M' \longrightarrow M'$ est l'image de *M* par la translation de vecteur $\vec{u}$

translatepoint3d

M u **translatepoint3d** $M' \longrightarrow M'$ est l'image de *M* par la translation de vecteur $\vec{u}$

translatepoint

A $\vec{u}$ **translatepoint** $A' \longrightarrow$ le point A' est l'image du point *A* par la translation de vecteur $\vec{u}$

translatepol

pol $\vec{u}$ **translatepol** $pol' \longrightarrow$ le polygône pol' est l'image du polygône *pol* par la translation de vecteur $\vec{u}$

TRc

string/*lit* **TRc** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire avec un cadre non dessiné, et dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Trc

string/*lit* **Trc** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire dont le contenu, centré, est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

trianglespherique

A B C **trianglespherique** $- \longrightarrow$ trace le triangle sphérique *ABC*, où les points *A*, *B* et et *C* sont donnés par leurs coordonnées sphériques respectives (r, θ_1, ϕ_1), (r, θ_2, ϕ_2) et (r, θ_3, ϕ_3)

trianglespherique

A B C **trianglespherique** $- \longrightarrow$ trace le triangle sphérique *ABC*, où les points *A*, *B* et et *C* sont donnés par leurs coordonnées sphériques respectives (r, θ_1, ϕ_1), (r, θ_2, ϕ_2) et (r, θ_3, ϕ_3)

trianglespherique*

A B C **trianglespherique*** $- \longrightarrow$ version étoilée de **trianglespherique**

trianglespherique*

A B C **trianglespherique*** $- \longrightarrow$ version étoilée de **trianglespherique**

trig_marks

array **trig_marks** $- \longrightarrow$ Numérote les axes *Ox* et *Oy* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

trig_xmarks

array **trig_xmarks** $- \longrightarrow$ Numérote l'axe *Ox* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

trig_ymarks

array **trig_ymarks** $- \longrightarrow$ Numérote l'axe *Oy* en fractions de π en utilisant les fractions définies par *array* qui est un tableau de chaines de caractères. Par exemple, $[(1)(-3/2)(2/3)]$ est un tableau valide

TR

string/*lit* **TR** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire avec un cadre non dessiné, et dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

Tr

string/*lit* **Tr** *tnode* $\longrightarrow$ Construit un nœud d'arbre rectangulaire dont le contenu est soit *string*, soit le label TEX défini par *lit*, soit l'objet picture désigné par *string*

tronque_cube

$solid_1$ *n* **tronque_cube** $solid_2 \longrightarrow solid_2$ est le solide obtenu en coupant chaque coin du parallélépipède $solid_1$. *n* est un réel supérieur à 2 indiquant la proportion à respecter pour la tronquature

truncate

num_1 **truncate** $num_2 \longrightarrow$ enlève la partie fractionnaire de num_1

U

ubpict

A [*xscale yscale*] $\{\alpha\}$ *string* **ubpict** $- \longrightarrow$ Se place en haut du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument $\{\alpha\}$ sont optionnels

ubtexlabel3d

ubtexlabel3d $- \longrightarrow$ Analogue 3d de la commande **ubtexlabel**

ubtexlabel

A [*xscale yscale*] {α} **ubtexlabel** – ⟶ Se place en haut du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ubtext3d

ubtext3d – ⟶ Analogue 3d de la commande **ubtext**

ubtext

string A [*xscale yscale*] {α} **ubtext** – ⟶ Se place en haut du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ucpict

A [*xscale yscale*] {α} *string* **ucpict** – ⟶ Se place en haut du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

uctexlabel3d

uctexlabel3d – ⟶ Analogue 3d de la commande **uctexlabel**

uctexlabel

A [*xscale yscale*] {α} **uctexlabel** – ⟶ Se place en haut du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

uctext3d

uctext3d – ⟶ Analogue 3d de la commande **uctext**

uctext

string A [*xscale yscale*] {α} **uctext** – ⟶ Se place en haut du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ulpict

A [*xscale yscale*] {α} *string* **ulpict** – ⟶ Se place en haut à gauche du point *A*, puis affiche l'objet désigné par la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ultexlabel3d

ultexlabel3d – ⟶ Analogue 3d de la commande **ultexlabel**

ultexlabel

A [*xscale yscale*] {α} **ultexlabel** – ⟶ Se place en haut à gauche du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

ultext3d

ultext3d – ⟶ Analogue 3d de la commande **ultext**

ultext

string A [*xscale yscale*] {α} **ultext** – ⟶ Se place en haut à gauche du point *A*, puis affiche la chaîne *string* avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle α. Le tableau d'échelle et l'argument {α} sont optionnels

unitaire3d

$\vec{u}$ **unitaire3d** $\vec{v}$ ⟶ Si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

unitaire3d

$\vec{u}$ **unitaire3d** $\vec{v}$ ⟶ Si $\vec{u} = \vec{0}$, alors $\vec{v} = \vec{0}$, sinon $\vec{v} = \vec{u}/\|\vec{u}\|$

unites

– **unites** – ⟶ trace les flèches unités sur chacun des axes

up

– **up** $\vec{u}$ ⟶ $\vec{u}$ est le vecteur $(0, 1)$

urtexlabel3d

urtexlabel3d – ⟶ Analogue 3d de la commande **urtexlabel**

urtexlabel

A [*xscale yscale*] {*alpha*} **urtexlabel** – ⟶ Se place en haut à droite du point *A*, puis dessine le label TEX en cours avec l'échelle (*xscale*, *yscale*) et après une rotation d'angle *alpha*. Le tableau d'échelle et l'argument {*alpha*} sont optionnels

urtext3d

urtext3d – ⟶ Analogue 3d de la commande **urtext**

usecolor

– **usecolor** – ⟶ charge le package *color*

uselabo

– **uselabo** – ⟶ charge le package *labo*

V

vad just

vad just ⟶ taille, en points postscript, du décalage vertical appliqué, s'il y a lieu par les commandes de positionnement de l'environnement 'picture' . **valeur par défaut :** 3, 75

variance

[$a_0 \ldots a_n$] **variance** v ⟶ le réel v est la variance de la série des a_i.

vecteur3d

A B **vecteur3d** u ⟶ $u = \overrightarrow{AB}$

vecteur3d

A B **vecteur3d** u ⟶ $u = \overrightarrow{AB}$

vecteur

A B **vecteur** $\vec{u}$ ⟶ A et B sont des points, et $\vec{u} = \overrightarrow{AB}$

vectprod3d

$\vec{u}$ $\vec{v}$ **vectprod3d** $\vec{w}$ ⟶ $\vec{w} = \vec{u} \wedge \vec{v}$

vectprod3d

$\vec{u}$ $\vec{v}$ **vectprod3d** $\vec{w}$ ⟶ $\vec{w} = \vec{u} \wedge \vec{v}$

verticale

a **verticale** D ⟶ dépose sur la pile la droite verticale D d'équation $x = a$

verticale?

D **verticale?** *bool* ⟶ vrai si la droite D est verticale, faux sinon

vert

– **vert** – ⟶ sélectionne la couleur vert

videsolid

solid **videsolid** – ⟶ ajoute au solide *solid* ses faces internes

view_square_matrix

M **view_square_matrix** $a_{1,1} \ldots a_{1,n}$ **()** $a_{1,1} \ldots a_{1,n}$ **()** ... **()** $a_{n,1} \ldots a_{n,n}$ ⟶ dépose sur la pile les coefficients de la matrice carrée M

W

wedge_

α β A r **wedge_** – ⟶ ajoute au chemin courant la portion de camembert de centre A, de rayon r, délimité par les angles α et β

wedge

α β A r **wedge** – ⟶ trace la portion de camembert de centre A, de rayon r, délimité par les angles α et β

widthangledroit

widthangledroit ⟶ taille, en points postscript, d'un des côté de l'angle droit tracé par **angledroit**. **valeur par défaut :** 5

withcontrolpoints

– **withcontrolpoints** – ⟶ active le tracé des points de contrôle lors du dessin d'une courbe de Bézier par **draw** ou **bezier_curve**

withoutcontrolpoints

– **withoutcontrolpoints** – ⟶ désactive le tracé des points de contrôle lors du dessin d'une courbe de Bézier par **draw** ou **bezier_curve**

X

xdpoint

x D **xdpoint** A ⟶ si la droite D n'est pas verticale, alors A est le point de D d'abscisse x. Erreur sinon

xmark

x **xmark** – ⟶ inscrit la numérotation au point d'abscisse x de l'axe Ox

xmarks

– **xmarks** – ⟶ inscrit toute la numérotation de l'axe *Ox*

xmarkstyle

string A **xmarkstyle** – ⟶ Procédure utilisée par les commandes **xmark** et dérivées pour inscrire la chaîne *string* au point *A*

xmax

xmax ⟶ borne supérieure sur l'axe *Ox*. **valeur par défaut :** 5

xmin

xmin ⟶ borne inférieure sur l'axe *Ox*. **valeur par défaut :** −5

xstquadrillage

xstquadrillage ⟶ pas sur l'axe *Ox* pour le quadrillage tracé avec la commande **quadrillage**. **valeur par défaut :** 1

xsubtick

x **xsubtick** – ⟶ trace un sous-tiret au point d'abscisse *x* de l'axe *Ox*

xsubticks

– **xsubticks** – ⟶ trace tous les sous-tirets de l'axe *Ox*

xtick

x **xtick** – ⟶ trace un tiret au point d'abscisse *x* de l'axe *Ox*

xticks

– **xticks** – ⟶ trace tous les tirets de l'axe *Ox*

Y

ydpoint

y D **ydpoint** *A* ⟶ si la droite *D* n'est pas horizontale, alors *A* est le point de *D* d'ordonnée *y*. Erreur sinon

ymark

y **ymark** – ⟶ inscrit la numérotation au point d'ordonnée *y* de l'axe *Oy*

ymarks

– **ymarks** – ⟶ inscrit toute la numérotation de l'axe *Oy*

ymarkstyle

string A **ymarkstyle** – ⟶ Procédure utilisée par les commandes **ymark** et dérivées pour inscrire la chaîne *string* au point *A*

ymax

ymax ⟶ borne supérieure sur l'axe *Oy*. **valeur par défaut :** 5

ymin

ymin ⟶ borne inférieure sur l'axe *Oy*. **valeur par défaut :** −5

ystquadrillage

ystquadrillage ⟶ pas sur l'axe *Oy* pour le quadrillage tracé avec la commande **quadrillage**. **valeur par défaut :** 1

ysubtick

y **ysubtick** – ⟶ trace un sous-tiret au point d'ordonnée *y* de l'axe *Oy*

ysubticks

– **ysubticks** – ⟶ trace tous les sous-tirets de l'axe *Oy*

ytick

y **ytick** – ⟶ trace un tiret au point d'ordonnée *y* de l'axe *Oy*

yticks

– **yticks** – ⟶ trace tous les tirets de l'axe *Oy*

Z

ZoomFactor_x

ZoomFactor_x ⟶ Facteur de zoom en *x*. **valeur par défaut :** 100

ZoomFactor_y

ZoomFactor_y ⟶ Facteur de zoom en *y*. **valeur par défaut :** 100