
Bienvenue à PSTricks

PSTricks est un ensemble de macros \TeX basées sur PostScript, compatibles avec la plupart des paquets de macros \TeX , ceci incluant Plain \TeX , \LaTeX , \AMSTeX et $\AMS\LaTeX$. PSTricks vous permet la couleur, les graphiques, la rotation des figures, les arbres et les transparents. PSTricks met la cerise (PostScript) sur le gâteau (\TeX) !

Pour installer PSTricks, suivez les instructions du fichier `read-me.pst` inclus dans le package PSTricks. Même si PSTricks est déjà installé, jetez un coup d'œil à `read-me.pst`.

Ce Guide de l'utilisateur tend à être un manuel de référence, ce qui signifie qu'il n'a pas été conçu pour être lu de façon linéaire. Voici une stratégie recommandée : terminer la lecture d'un bref survol des caractéristiques de PSTricks. Ensuite feuillotez le guide entier pour avoir votre propre panorama. Revenez ensuite à la partie I (L'essentiel) et lisez-le de façon approfondie. Référez-vous aux sections restantes au fur et à mesure de vos besoins.

Quand vous ne savez pas comment procéder ou quand des problèmes surgissent, feuillotez l'appendice (Aide). Vous trouverez très souvent une solution. Il y a aussi un fichier \LaTeX d'exemples `samples.pst` qui est distribué avec PSTricks. Regardez ce fichier et inspirez-en vous.

La documentation est écrite avec \LaTeX . Quelques exemples utilisent certaines constructions spécifiques de \LaTeX , d'autres non. Cependant, il n'y a rien de spécifique à \LaTeX dans les macros, et il n'y a pas de choses qui ne fonctionnent pas sous \LaTeX . Ce paquet a été testé sous Plain \TeX , \LaTeX , $\AMS\LaTeX$ et \AMSTeX , et devrait fonctionner avec d'autres paquets de macros \LaTeX .



Le fichier des principales macros est `pstricks.tex/pstricks.sty`. Chacun des fichiers de macros de PSTricks a une extension `.tex` et une extension `.sty`; elles sont équivalentes, mais l'extension `.sty` signifie que vous pouvez inclure le fichier comme une option de style du document \LaTeX .

Il y a de nombreux fichiers de macros supplémentaires. Un fichier comme celui du dessus et celui de gauche, est utilisé dans le Guide de l'utilisateur pour vous rappeler que vous devez appeler un fichier avant d'utiliser les macros qu'il contient.

Pour la plupart des macros PSTricks, même si vous les utilisez mal, vous n'aurez pas d'erreur PostScript à la sortie. cependant il est recommandé de corriger toutes les erreurs \TeX avant d'imprimer votre document. Un petit nombre de macros PSTricks saute des erreurs PostScript sans avertissement.



Utilisez ces macros avec précaution, surtout si vous utilisez une imprimante en réseau, parce que des erreurs PostScript peuvent conduire à une bombe sur l'imprimante. De telles macros sont spécialement signalées, en utilisant un avertissement comme celui-ci

Avertissement : utilisez des macros qui ne s'arrêtent pas à des erreurs PostScript avec précaution. Les erreurs PostScript peuvent conduire à une bombe sur l'imprimante !

Retenez les conventions typographiques suivantes utilisées dans ce guide de l'utilisateur :

- Toutes les entrées de caractères littéraux, i.e. qui apparaissent en verbatim dans votre fichier, apparaissent en caractères droits ;
- Les arguments que vous êtes supposé substituer à une valeur (par exemple, angle) apparaissent en caractères penchés et penchés gras.
- L'entrée principale pour une macro ou un paramètre qui spécifie sa syntaxe apparaît dans une grande fonte en gras, excepté pour les arguments optionnels qui sont en taille moyenne. C'est comme cela que vous reconnaîtrez un argument optionnel.
- Les références à des paramètres ou des commandes PSTricks à l'intérieur de paragraphes sont en Gras.

L'essentiel

1 Arguments et délimiteurs

Voici quelques éléments de base sur les arguments et les délimiteurs qu'il est très important de connaître.

Les macros de PSTricks utilisent les délimiteurs suivants :

	accolades	{ argument }
crochets (uniquement pour les arguments en option)		[argument]
Parenthèses et virgule pour les coordonnées		(x, y)
= et , pour les paramètres		para1=valeur1, ...

Les espaces et les virgules peuvent aussi servir de délimiteurs sans argument, mais dans ce cas l'argument est développé avant de regarder les délimiteurs.

Toujours utiliser le point plutôt que la virgule pour noter un nombre décimal pour que PSTricks ne prenne pas la virgule comme un délimiteur.

L'erreur la plus courante est d'oublier les délimiteurs. Ceci génère des erreurs pour \TeX ou pour PSTricks ou d'autres messages peu informatifs comme :

```
! Use of \get@coor doesn't match his definition
! Paragraph ended before \pst@addcoor was complete
! Forbidden control sequence found while scanning use
of \check@arrow
! File ended while scanning use of \lput
```

Les délimiteurs sont la première chose à regarder quand vous rencontrez des erreurs dans une macro PSTricks.

Comme les macros de PSTricks peuvent avoir plusieurs arguments, il faut savoir que vous pouvez insérer un espace ou une nouvelle ligne entre deux arguments, exception faite des arguments entourés par des accolades. Si vous voulez quand même insérer une nouvelle ligne entre deux arguments entourés par des accolades, placez un caractère de commentaire (%) à la fin de la ligne.

En règle générale le premier caractère, en dehors d'une espace après une macro PSTricks ne doit pas être [ou (. Sinon, PSTricks peut penser que le [ou la (fait partie de la macro. Vous pouvez contourner cette interdiction en insérant une paire d'accolades {} entre la macro et le [ou la (.

2 Couleur

Les graduations de gris :

`black`, `darkgray`, `gray`, `lightgray` et `white`

et les couleurs :

`red`, `green`, `blue`, `cyan`, `magenta` et `yellow`

sont prédéfinies dans PSTricks.

Ceci signifie que ces noms peuvent être utilisés avec les objets graphiques qui sont décrits dans les paragraphes suivants. Ceci signifie aussi que la commande `\gray` (ou `\red`, etc.) peut être utilisée beaucoup mieux qu'avec `\rm` ou `\tt`, comme dans :

`{\gray Cette ligne est en gris }`

Les commandes `\gray`, `\red` etc. peuvent être emboîtées comme les commandes de fontes. Il n'y a qu'en peu d'occasions que les commandes diffèrent :

1) Les commandes de couleur peuvent être utilisées en mode mathématique ou hors mode mathématique (il n'y a pas de restrictions autres que celles dues à \TeX).

2) Les commandes de couleur affectent tout ce qui est dans leur portée (c'est-à-dire dans la ligne) et pas simplement des caractères.

3) La portée d'une commande de couleur ne dépasse pas sa page.

4) Les commandes de couleur ne sont pas aussi robustes que les commandes de fontes quand elles sont utilisées à l'intérieur de boîtes. Voir la page pour les détails. Vous pouvez éviter la plupart des problèmes en groupant de façon explicite les commandes de couleur (c'est-à-dire en incluant la portée de la commande par une paire d'accolades {}), quand elles sont dans l'argument d'une autre commande.¹

Vous pouvez définir ou redéfinir des couleurs ou des niveaux de gris supplémentaires avec les commandes suivantes. Dans chaque cas, *nombre* est un nombre entre 0 et 1. Les espaces sont utilisées en tant que délimiteurs ne mettez pas d'espaces superflues dans les arguments.

1. Ceci n'est pas nécessaire avec les commandes de boîtes LR PSTricks, regardez quand `\psverbboxtrue` est en action. Voir le paragraphe A.

`\newgray{couleur}{nomb.}`

nomb. est le niveau de gris qui sera utilisé par l'opérateur `setgray` de PostScript. 0 correspond au noir et 1 au blanc. Par exemple :

```
\newgray{darkgray}{0.25}
```

`\newrgbcolor{couleur}{nomb.1 nomb.2 nomb.3}`

nomb.1 nomb.2 nomb.3 est une spécification RGB (Rouge Vert Bleu) interprété par l'opérateur PostScript `setrgbcolor`. Par exemple :

```
\newrgbcolor{vert}{0 1 0}
```

`\newhsbcolor{couleur}{nomb.1 nomb.2 nomb.3}`

nomb.1 nomb.2 nomb.3 est la spécification teinte-saturation-intensité interprété par l'opérateur PostScript `sethsbcolor`. Par exemple :

```
\newhsbcolor{macouleur}{0.3 0.7 0.9}
```

`\newcmykcolor{couleur}{nomb.1 nomb.2 nomb.3 nomb.4}`

nomb.1 nomb.2 nomb.3 nomb.4 est une spécification cyan-magenta-black-yellow (cyan-magenta-noir-jaune) interprété par l'opérateur PostScript `nexcmykcolor`. Par exemple :

```
\newcmykcolor{sacouleur}{.5 1 0 .5}
```

Pour définir de nouvelles couleurs, la méthode RGB est sûre, hsb non recommandée, CMYK n'est pas supporté par tous les PostScript niveau 1 bien que ce soit le mieux pour imprimer des couleurs. Pour plus d'informations sur les modèles et les spécifications de couleurs, consultez *PostScript Language Reference Manual*, 2nd Edition (livre rouge), et un guide des couleurs.

Note pour le driver : la commande `\pstVerb` doit être définie.

3 Établissement des paramètres graphiques

PSTricks utilise un système de valeurs-clés des paramètres graphiques pour modifier à votre goût les macros qui génèrent des graphiques (par exemple des segments et des cercles), ou des graphiques combinés à du texte (par exemple des boîtes encadrées). Vous pouvez changer les valeurs par défaut des paramètres grâce à la commande

`\psset`, comme dans

```
\psset{fillcolor=yellow}  
\psset{linecolor=blue,framearc=0.3,dash=3pt 6pt}
```

La syntaxe générale est :

```
\psset{para.1=valeur1,para.2=valeur2,...}
```

Comme on le voit dans les exemples ci-dessus, les espaces sont utilisées comme délimiteurs pour quelques valeurs. Des espaces en trop sont permises, mais uniquement après la virgule qui sépare *para.=valeur* (ce qui est alors un bon endroit pour aller à la ligne s'il y a beaucoup de paramètres à modifier). Ainsi, le premier exemple est acceptable, le second non :

```
\psset{fillcolor=yellow, linecolor=blue}  
\psset{fillcolor= yellow, linecolor =blue}
```

Les paramètres sont décrits à travers tout le guide de l'utilisateur, quand le besoin s'en fait sentir.

À peu près toutes les macros qui utilisent des paramètres graphiques vous permettent d'inclure des modifications en tant que premier argument optionnel, entouré de crochets comme par exemple :

```
\psline[linecolor=green,linestyle=dotted](8,7)
```

qui trace un segment vert en pointillé. Ceci est équivalent à :

```
{\psset{linecolor=green,linestyle=dotted}}\psline(8,7)}
```

Pour beaucoup de paramètres, PSTricks calcule la valeur et la stocke dans une forme particulière, prête pour le travail par PostScript. Pour les autres, PSTricks stocke la valeur dans une forme que vous pouvez prévoir. dans ce dernier cas, ce manuel mentionne le nom de la commande où la valeur est stockée. Ceci afin d'utiliser la valeur pour établir d'autres paramètres. Ainsi :

```
\psset{linecolor=\psfillcolor,doublesep=.5\pslinewidth}
```

Cependant, même pour ces paramètres, PSTricks peut commettre quelques erreurs, et vous devez toujours établir ces paramètres en utilisant `\psset` ou comme un changement de paramètre optionnel, plutôt que de redéfinir la commande où la valeur est stockée.

4 Dimensions, coordonnées et angles

À chaque fois qu'un argument d'une macro de PSTricks est une dimension, l'unité est optionnelle. L'unité par défaut est définie par le paramètre :

unit=dim

Par défaut : 1cm

par exemple, avec la valeur par défaut de 1cm, les instructions suivantes sont équivalentes :

```
\psset{linewidth=.5cm}
\psset{linewidth=.5}
```

En n'explicitant pas les unités, vous pourrez changer l'échelle de votre graphique en changeant la valeur de **unit**.

Vous pouvez établir les coordonnées par défaut quand vous fixez des dimensions non-PSTricks aussi bien en utilisant les commandes

```
\pssetlength{cmd}{dimension}
\psaddtolength{cmd}{dimension}
```

où *cmd* est un registre de dimension (en parlé L^AT_EX, une longueur), et *dimension* est une longueur avec une unité optionnelle. Elles sont analogues à `\setlength` et `\addtolength` de L^AT_EX.

Les couples de coordonnées ont la forme (x, y) . L'origine du système de coordonnées est au point courant de T_EX. La commande `\SpecialCoor` vous permet d'utiliser les coordonnées polaires, dans la forme $(r; a)$, où r est le rayon (une dimension) et a un angle (voir ci-dessous). Vous pouvez utiliser des coordonnées cartésiennes. Pour une description complète de `\SpecialCoor`, voir le paragraphe 34.

Le paramètre **unit** établit les trois paramètres suivants :

xunit=dimension
yunit=dimension
runit=dimension

par défaut : 1cm
par défaut : 1cm
par défaut : 1cm

Voici donc les unités par défaut pour les coordonnées x , y et toutes les autres coordonnées. En fixant de manière indépendante ces unités, vous pouvez mettre à l'échelle les longueurs x et y de façon séparée. Après avoir changé **yunit** en 1pt, les deux lignes `\psline` ci-dessous sont équivalentes :

```
\psset{yunit=1pt}
\psline(0cm,20pt)(5cm,80pt)
\psline(0,20)(5,80)
```

Les valeurs des paramètres **xunit**, **yunit** et **runit** sont stockées dans les registres `\psxunit`, `\psyunit` et `\psunit` (aussi `\psrunit`).

Les angles, en coordonnées polaires et autres arguments doivent être des nombres donnant les angles en degrés par défaut. Vous pouvez changer les

unités pour les angles grâce à la commande

`\degrees[nombr.]`

nombr. doit être le nombre d'unités dans le cercle. par exemple, vous pouvez utiliser

`\degrees[100]`

pour faire un camembert quand vous connaissez les proportions en pourcentage. `\degrees` sans l'argument est équivalent à

`\degrees[360]`

La commande

`\radians`

est un raccourci pour

`\degrees[6.28319]`

`\SpecialCoor` vous permet de spécifier les angles dans les unités que vous voulez aussi bien.

5 Paramètres graphiques de base

La largeur et la couleur des lignes est déterminée par les paramètres :

`linewidth=dim`

Par défaut : 0.8pt

`linecolor=couleur`

Par défaut : black

`linewidth` est stockée dans le registre `\pslinewidth` et `linecolor` est stockée dans la commande `\pslinecolor`.

Les régions délimitées par des courbes ouvertes ou fermées peuvent être remplies en utilisant les paramètres :

fillstyle=style

fillcolor=couleur

En écrivant **fillstyle=none**, les régions ne sont pas coloriées. Quand on écrit **fillstyle=solid** les régions sont coloriées avec **fillcolor**. Les autres styles de **fillstyle** sont décrits dans le paragraphe 14.

Les objets graphiques ont tous une version étoilée (ex., **\psframe***) qui dessine un objet coloré par la commande **linecolor**. Par exemple



`\psellipse*(1,.5)(1,.5)`

Les courbes ouvertes peuvent avoir des flèches définies par le paramètre

arrows=flèches

Si **arrows=-**, il n'y a pas de flèches. Si **arrows=<->** il y a des flèches aux deux extrémités de la courbe. Vous pouvez aussi écrire **arrows=->** et **arrows=<-**. Avec les courbes ouvertes, vous pouvez aussi spécifier les flèches en tant qu'argument optionnel avec une paire de parenthèses. Cet argument vient après les arguments de paramètres optionnels. Exemple :



`\psline[linewidth=2pt]{->}(2,1)`

D'autres styles de flèches sont décrits dans le paragraphe 15.

Si vous utilisez

showpoints=true/false

Par défaut : false

avec la valeur *true* il y aura sur la courbe des points aux coordonnées utilisées ou points de contrôle de l'objet graphique² Le paragraphe 9 montre comment changer le style des points.

2. La valeur du paramètre est stockée dans le registre conditionnel **ifshowpoints**.



Objets graphiques élémentaires

6 Lignes et polygones

Les objets de cette section utilisent les paramètres suivants :

linearc=*dimension* **par défaut : 0pt**

Le rayon des arcs tracés dans les coins des (séries de) lignes et les polygones au moyen de `\psline` et `\pspolygon`. *dim* doit être positive.

framearc=*nombre* **par défaut : 0pt**

Dans `\psframe` et les macros mentionnées permettant de fabriquer des boîtes, le rayon des coins arrondis est fixé, par défaut à une fois et demie *nombre* multiplié par la largeur ou la hauteur de l'encadrement – celle qui est la plus petite –. *num* doit être compris entre 0 et 1.

cornersize=*relatif/absolu* **par défaut : relatif**

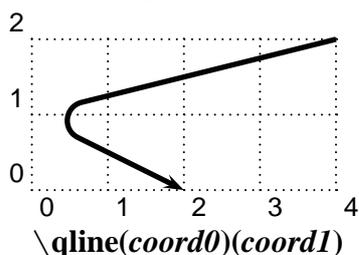
Si **cornersize** est relatif, alors le paramètre **framearc** détermine le rayon des coins arrondis pour `\psframe`, comme décrit ci-dessus (et alors le rayon dépend de la taille de l'encadrement). Si **cornersize** est absolu, alors le paramètre **linearc** détermine le rayon des coins arrondis pour `\textbfpsframe` (et alors le rayon est de taille constante).

Voici maintenant les lignes et les polygones :

`\psline*`[*para.*] {*flèches*}(*x0,y0*)(*x1,y1*) ... (*xn,yn*)

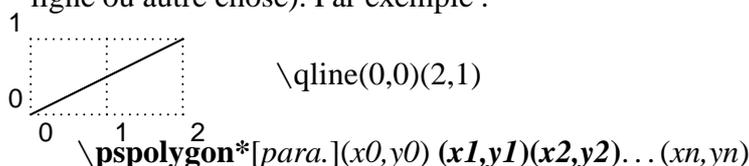
trace une suite de segments commençant et finissant aux points de coordonnées (*x0,y0*)(*x1,y1*)... (*xn,yn*).

Exemple

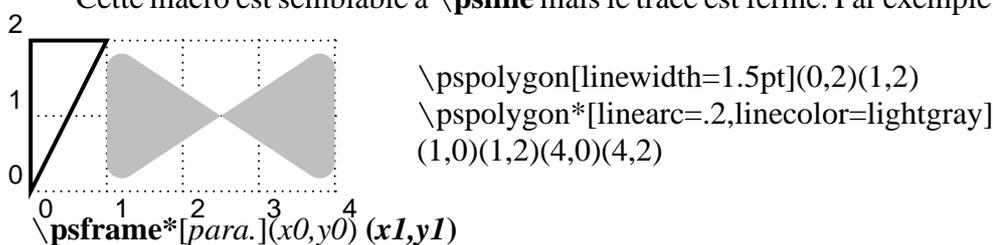


`\psline[linewidth=2pt,linearc=0.25]`
`{ -> } (4,2)(0,1)(2,0)`

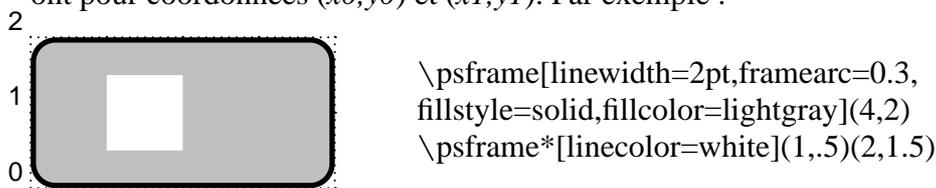
Ceci est une version plus dépouillée de `\psline` qui ne prend pas en compte le paramètre **flèches** et qui trace un seul segment. Il faut noter que les deux coordonnées sont obligatoires et qu'il n'y a pas de paramètre optionnel (utilisez `\psset` si vous voulez changer par exemple la largeur de la ligne ou autre chose). Par exemple :



Cette macro est semblable à `\psline` mais le tracé est fermé. Par exemple :



`\psframe` dessine un rectangle dont les coins diagonalement opposés ont pour coordonnées $(x0,y0)$ et $(x1,y1)$. Par exemple :

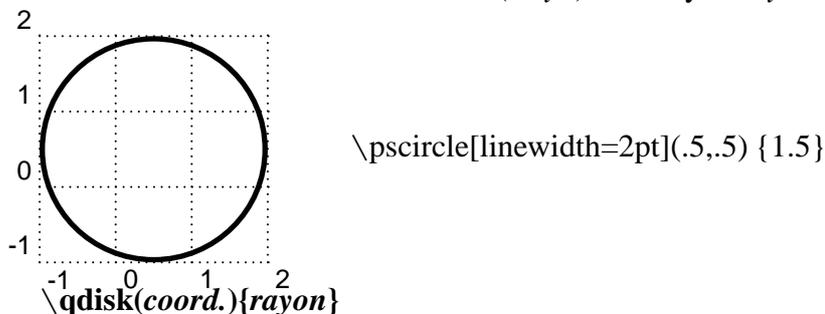


N.B. Remarquez que s'il n'y a qu'un seul point, l'autre sommet est l'origine.

7 Arcs, cercles et ellipses

`\pscircle*[para.](x0,y0){rayon}`

Ceci dessine un cercle centré en $(x0,y0)$ et de rayon *rayon*. Par exemple :

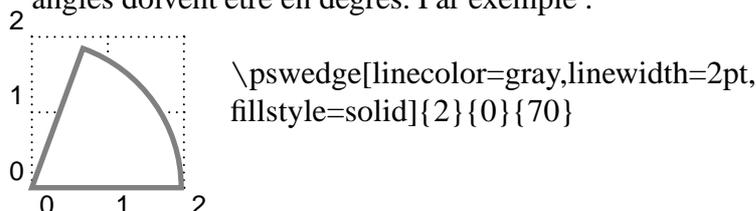


Ceci est une version simplifiée de `\pscircle`. Noter que les deux arguments sont obligatoires et qu'il n'y a pas de paramètres. Pour changer la couleur du disque utilisez auparavant `\psset`. Exemple :

● `\psset{linecolor=gray} \qdisk(0.2,0.2){+4pt}`

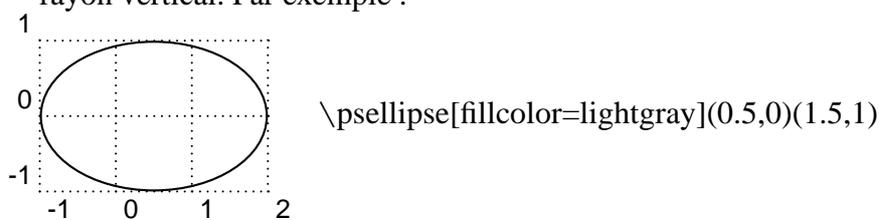
`\pswedge*[para.](x0,y0) {rayon} {angle1}{angle2}`

Cette macro dessine un arc centré en $(x0,y0)$, de rayon *rayon* en partant de l'angle1 jusqu'à l'angle2, l'arc étant décrit dans le sens anti-horaire. Les angles doivent être en degrés. Par exemple :



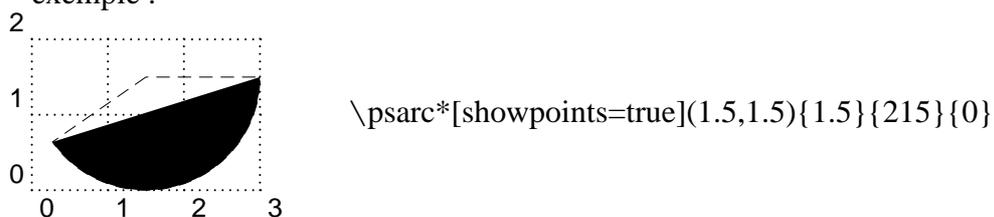
`\psellipse*[para.](x0,y0)(x1,y1)`

$(x0,y0)$ est le centre de l'ellipse, $x1$ est le rayon horizontal et $y1$ est le rayon vertical. Par exemple :



`\psarc*[para.]{flèches} {rayon}(angleA)(angleB)`

Ceci décrit un arc de l'angle A jusqu'à l'angle B, dans le sens des aiguilles d'une montre, sur un cercle de rayon *rayon* et centré à (x, y) . Il faut inclure soit l'argument des flèches soit l'argument du centre (x, y) . Par exemple :



Notez que `showpoints=true` trace une ligne en tireté du centre vers l'arc ; ceci est utile pour la composition de certaines figures.

`\psarc` utilise aussi les paramètres :

arcsepA=dim

Par défaut : 0pt

L'*angleA* est ajusté de sorte que l'arc touche juste une ligne de largeur *dim* qui part du centre de l'arc dans la direction de *angleA*

arcsepB=dim

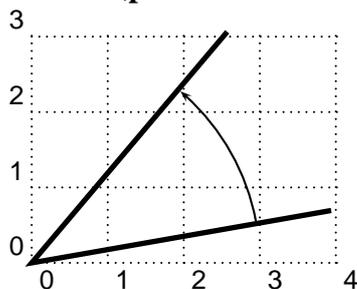
Par défaut : 0pt

Identique à **arcsepA** mais l'*angleB* est ajusté.

arcsep=dim**Par défaut : 0pt**

Ceci ajuste à la fois l'*angleA* et l'*angleB*.

Ces paramètres facilitent le tracé de deux droites sécantes et alors utilisent `\psarc` avec flèches pour indiquer l'angle entre elles. Par exemple :



```
\SpecialCoor
\psline[linewidth=2pt](4;50)(0,0)(4;10)
\psarc[arcsep=2pt]{->}{3}{10}{50}
```

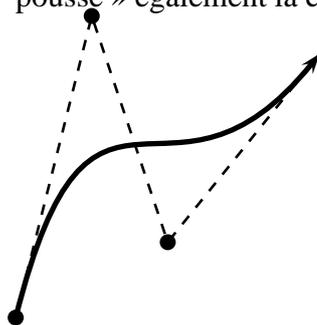
`\psarcn*[para.]{flèches}(x,y){rayon}(angleA)(angleB)`

Comme `\psarc` mais l'arc est dessiné dans le sens trigonométrique. On peut obtenir la même chose avec `\psarc` en échangeant *angleA* et *angleB* et le sens des flèches.³

8 Courbes

`\psbezier*[para.]{flèches}(x0,y0)(x1,y1)(x2,y2)(x3,y3)`

`\psbezier` dessine une courbe de Bézier avec les quatre points de contrôle. La courbe commence au premier point, tangente à la droite allant au deuxième point. Elle finit au dernier point, tangente à la droite contenant le troisième point. le deuxième et le troisième point, en plus de définir les tangentes, « pousse » également la courbe vers eux. Par exemple :



```
\psbezier[linewidth=2pt,showpoints=true]
{->}(0,0)(1,4)(2,1)(4,3.5)
```

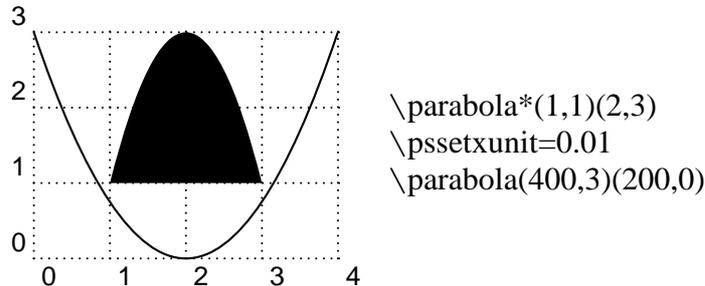
`showpoints=true` montre tous les points de contrôle et les relie par des tirets, ce qui est utile pour ajuster vos courbes de Bézier.

`\parabola*[para.]{flèches}(x0,y0)(x1,y1)`

`\parabola` dessine une parabole partant du point (x_0, y_0) et dont l'ex-

3. Cependant, avec les objets graphiques utilisant `pscustom` décrit dans la partie IV, `psarc` n'est pas redondant.

tremum est le point (x_1, y_1) . Par exemple :



Les trois objets graphiques suivants interpolent une courbe ouverte ou fermée passant par des points donnés. La courbe à chaque point intérieur est perpendiculaire à la bissectrice de l'angle \widehat{ABC} , où B est le point intermédiaire et A et C ses voisins. Si vous mettez les coordonnées à une autre échelle, la courbe **ne sera pas** mise à l'échelle proportionnellement.

La courbure de la courbe est contrôlée par les paramètres suivants :

curvature=*nombr1 nombr2 nombr3*

Par défaut : 1 0.1 0

Vous avez juste à faire varier ce paramètre pour obtenir ce que vous voulez. des valeurs individuelles en dehors de l'intervalle $[-1 ; 1]$ sont soit ignorées soit pour des essais. A, B et C font référence à des points consécutifs.

Les plus petites valeurs de *nombr.1* donnent une courbe plus tendue.

Les plus petites valeurs de *nombr.2* tendent la courbe là où l'angle \widehat{ABC} mesure plus de 45° et la détendent partout ailleurs.

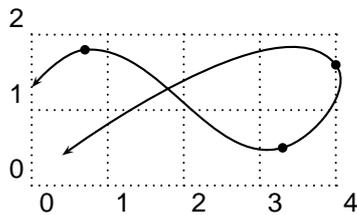
nombr.3 détermine la pente à chaque point. Si *nombr.3* = 0, la courbe est perpendiculaire en B à la bissectrice de \widehat{ABC} . Si *nombr.3* = 1, la courbe est parallèle à la droite (AC). Avec cette valeur (et cette valeur seulement) une mise à l'échelle des coordonnées entraîne une mise à l'échelle de la courbe proportionnellement. Cependant des valeurs positives peuvent donner de meilleurs résultats des points de coordonnées irrégulières. Des valeurs inférieures à -1 ou supérieures à 2 sont converties respectivement en -1 et 2 .

Voici les trois macros d'interpolation de courbes :

\pscurve*[*para.*]{*flèches*}(*x1,y1*) ... (*xn,yn*)

Cette macro interpole une courbe non fermée à travers les points. Par exemple :

Points

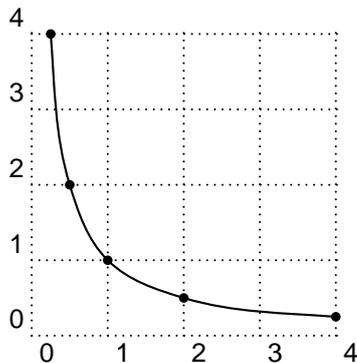


```
\pscurve[showpoints=true]{<->}(0,1.3)
(0.7,1.8)(3.3,0.5)(4,1.6)(0.4,0.4)
```

Vous noterez l'utilisation de **showpoints=true** pour voir les points. Ceci est d'un grand secours quand on veut construire une courbe.

```
\psecurve*[para.]{flèches}(x1,y1)...(xn,yn)
```

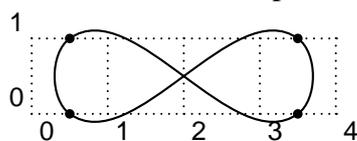
Comme **\pscurve**, mais la courbe débute au second point et s'arrête à l'avant-dernier point. Ceci vous permet de déterminer la façon dont la courbe devrait relier la premier et le dernier point (très utile en cas d'asymptote). Le « e » a quelque chose à voir avec « endpoints ». Par exemple :



```
\psecurve[showpoints=true](.125,8)(.25,4)
(.5,2)(1,1)(2,.5)(4,.25)(8,.125)
```

```
\psccurve*[para.]{flèches}(x1,y1)...(xn,yn)
```

Ceci interpole une courbe fermée passant par les points. « c » signifie « closed ». Par exemple :



```
\psccurve[showpoints=true](.5,0)(3.5,1)(3.5,0)(.5,1)
```

9 Points

L'objet graphique

```
\psdots*[para.](x1,y1)(x2,y2)...(xn,yn)
```

place un point à chaque coordonnée. En tant que « point » il dépend de la valeur de

dotstyle=style

Par défaut : *

Ceci détermine les points que vous obtenez quand **showpoints=true**.

Les styles de points sont très intuitifs :

<i>Style</i>	<i>Exemple</i>	<i>Style</i>	<i>Exemple</i>
*	• • • •	square	□ □ □ □
o	○ ○ ○ ○	square*	■ ■ ■ ■
+	+ + + +	pentagon	◊ ◊ ◊ ◊
triangle	△ △ △ △	pentagon*	● ● ● ●
triangle*	▲ ▲ ▲ ▲		

Comme avec les flèches, il y a un paramètre pour mettre les points à l'échelle :

dotscale=*nombr.1 nombr.2* **Par défaut : 1**

Les points sont mis à l'échelle horizontalement avec *nombr.1* et verticalement avec *nombr.2*. Si vous ne mettez qu'un nombre les points seront agrandis de la même façon dans les deux directions.

Il y a aussi un paramètre pour tourner les points :

dotangle=*angle* **Par défaut : 0**

Ainsi par exemple en écrivant **dotangle=45**, le style + vous donne × et le style square vous donne un losange. Remarquez que les points sont d'abord agrandis puis tournés.

La taille sans échelle du style | est contrôlée par le paramètre **tbar**size et la taille des autres styles est contrôlée par **dots**ize. Ceci est décrit au paragraphe 15. Le rayon, déterminé par la valeur de **dots**ize est celui des cercles et des disques. Les autres types de points sont de taille similaire⁴

Les tailles des points dépendent de **linew**idth en raison du paramètre **show**points. Cependant vous pouvez fixer la taille des points de façon arbitraire en fixant le second nombre dans le paramètre **dots**ize à 0. Exemple :

```
\psset{dotsize=3pt 0}
```

fixe la taille des points à 3pt, indépendamment de la valeur de **linew**idth.

10 Grilles

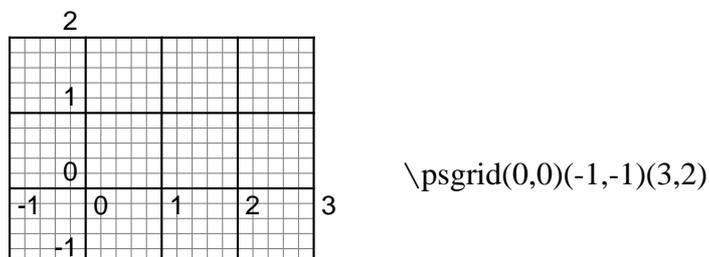
PSTricks possède une macro puissante pour dessiner des grilles et du papier à grapher.

```
\psgrid(x0,y0)(x1,y1)(x2,y2)
```

\psgrid dessine une grille dont les points diagonalement opposés ont pour coordonnées $(x1,y1)$ et $(x2,y2)$. Les intervalles sont numérotés par rap-

4. Les polygones sont agrandis de façon à avoir la même aire que les disques. Un losange est juste un carré tourné.

port au point de coordonnées $(x0,y0)$. Ces coordonnées sont toujours considérées comme des coordonnées cartésiennes. Par exemple :



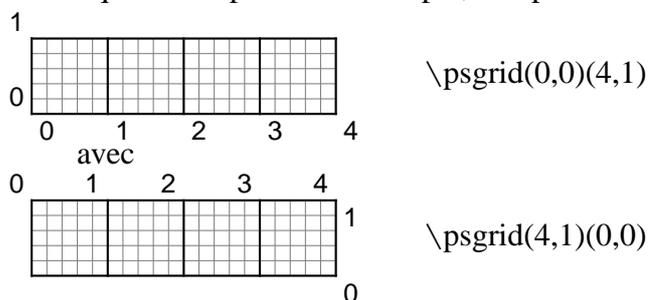
Notez que les coordonnées et la position des labels fonctionnent de la même façon qu'avec `\psaxes`.

Les divisions de la grille principale sont des multiples de **xunit** et de **yunit**. Les subdivisions sont permises. En général, les coordonnées sont entières, données sans unité.

Si les coordonnées $(x0,y0)$ sont omises, $(x1,y1)$ est utilisé. Par défaut ce couple $(x1,y1)$ est fixé à $(0,0)$. Si vous ne donnez aucun couple, alors les coordonnées de l'environnement `\pspicture` sont utilisées ou bien une grille de 10×10 est dessinée.

Ainsi vous pouvez inclure une commande `\psgrid` sans coordonnées dans un environnement `\pspicture` pour avoir une grille qui vous aidera pour positionner les objets sur la figure.

Les divisions de la grille principale sont numérotées, avec les labels positionnés près de la ligne verticale à $x0$ (jusqu'à $x2$) et près de la ligne horizontale à $x1$ (jusqu'à $y2$). $(x1, y1)$ désignant n'importe quel coin de la grille, tant que $(x2, y2)$ désigne le coin opposé, vous pouvez placer les labels du côté qu'il vous plait. Par exemple, comparez :



Les paramètres suivants s'appliquent uniquement à `\psgrid` :

gridwidth

Par défaut : 0.8pt

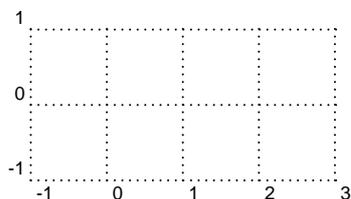
La largeur des lignes de la grille.

gridcolor

Par défaut : black

La couleur des lignes de la grille. griddots=<i>nombr.</i>	Par défaut : 0
Si <i>nombr.</i> est positif, les lignes de la grille sont en pointillés, avec <i>nombr.</i> points par division. gridlabels = <i>dim</i>	Par défaut : 10pt
La taille des nombres en abscisses et ordonnées. gridlabelcolor=<i>color</i>	Par défaut : black
La couleur des graduations de la grille. subgriddiv=<i>dim</i>	Par défaut : 0.4pt
La largeur des subdivisions de la grille. subgridwidth=<i>dim</i>	Par défaut : 0.4pt
La largeur des subdivisions de la grille. gridcolor=<i>color</i>	Par défaut : gray
La couleur des subdivisions de la grille. subgriddots=<i>nombr.</i>	Par défaut : 0

Comme **griddots**, mais pour les subdivisions.
Voici une grille qui vous est familière et qui illustre certains de ces paramètres :



```
\psgrid[subgriddiv=1,griddots=10, gridlabels=7pt](-1,-1)(3,1)
```

Notez que les valeurs de **xunit** et **yunit** sont des paramètres importants pour **\psgrid**, car elles déterminent l'espacement des divisions. Ainsi si leurs valeurs sont fixées à 1pt et si vous écrivez

```
\psgrid(0,0)(10in,10in)
```

vous obtiendrez une grille avec 723 divisions principales et 3 615 subdivisions ! (Actuellement, **\psgrid** permet 500 divisions ou subdivisions au maximum pour limiter les dégâts causés par de telles erreurs.) Probablement vous fixerez l'unité à .5 in ou à 1in, comme dans

```
\psgrid[unit=.5in](0,0)(20,20)
```

11 Tracés point par point



Les commandes de tracé point par point décrites dans ce paragraphe sont définies dans `pst-plot.tex/pst-plot.sty`, que vous devrez d'abord charger.

Tracés point par point

Les instructions `\psdots`, `\psline`, `\pspolygon`, `\pscurve`, `\psecurve` et `\psccurve` permettent de traduire des données en points de plusieurs façons. Cependant, vous devez d'abord générer les données et les entrer en tant que couple de coordonnées (x, y) . Les macros de tracés par point de ce paragraphe vous permettent d'obtenir et d'utiliser de manière différente ces données. (Le paragraphe 26 vous montre comment obtenir les axes.)

Le paramètre

`plotstyle=style`

Par défaut : line

détermine quelle sorte de points vous obtenez. les styles autorisés sont : dots, line, polygon, curve, ecurve, ccurve. Ainsi si le style choisi est polygon, alors la macro devient une variante de l'objet graphique `\pspolygon`.

Vous pouvez utiliser des flèches avec les styles qui sont des courbes ouvertes, mais il n'y a pas d'argument optionnel permettant de personnaliser ces flèches. Vous devez plutôt utiliser le paramètre **arrows**.

Attention : Aucune vérification d'erreur PostScript n'existe pour les arguments de données. Lire l'Appendice C avant d'inclure du code PostScript dans les arguments.

*Il y a des limites dépendantes du système sur la quantité de données que T_EX et PostScript peuvent traiter. Vous avez beaucoup moins de chances de dépasser les limites de PostScript quand vous utilisez les styles line, polygon et plots, avec **showpoints=false**, **linearc=0pt**, et en ne mettant pas de flèches.*

Notez que les listes de données générées ou utilisées par les commandes de tracés point par point ne peuvent contenir d'unités. Les valeurs de `\psxunit` et de `\psyunit` sont utilisées comme unités.

`\fileplot*[para.]{fichier}`

`\fileplot` est la méthode la plus simple pour représenter des fonctions point par point. Vous avez juste besoin d'un fichier qui contient la liste des coordonnées (sans unités), tel celui généré par Mathematica ou d'autres logiciels mathématiques. Les données peuvent être délimitées par des accolades, { }, des parenthèses, des virgules, et/ou des espaces. Entourer toutes les données avec des crochets [] augmentera sensiblement la vitesse de traitement des données, mais il y a des limites dépendant de la plate-forme utilisée donnant la quantité maximale de données que \TeX peut lire en une seule fois. (Le [doit être au début de la ligne.) le fichier ne doit contenir rien d'autre (pas même `\endinput`) exception faite des commentaires précédés de %.

`\fileplot` reconnaît seulement les styles `line`, `polygon`, et `dots plot`, et ignore les paramètres **arrows**, **linearc** et **showpoints**. La commande **`\listplot`**, décrite ci-dessous, peut également lire des données d'un fichier, sans ces restrictions, et avec un traitement par \TeX plus rapide. Cependant, vous avez moins de possibilité de dépasser la mémoire de PostScript ou de calculer aux limites de la pile.

Si vous trouvez que \TeX prend beaucoup de temps à analyser votre commande **`\fileplot`**, vous pourrez utiliser la commande **`\PSTtoEPS`** décrite page 80 .

Ceci réduira les exigences de mémoire de \TeX .

`\dataplot*[para.]{commandes}`

`\dataplot` permet également de traduire des listes de données engendrées par d'autres programmes, mais vous devez d'abord récupérer les données par l'une des commandes suivantes.

`\savedata{commande}[data]`

`\readdata{commande}{fichier}`

data ou les données dans le *fichier* doivent être conformes aux règles édictées ci-dessus pour les données dans **`\fileplot`** (avec **`\savedata`**, les données doivent être délimitées par [], et avec **`\readdata`**, entourer les données avec [] accélère le traitement.) Vous pouvez concaténer et réutiliser les listes, comme dans :

```
\readdata{\foo}{foo.data}
```

```
\readdata{\bar}{bar.data}
```

```
\dataplot{\foo\bar}
```

```
\dataplot[origin={0,1}]{\bar}
```

La combinaison **`\readdata-\dataplot`** est plus rapide que **`\fileplot`** si

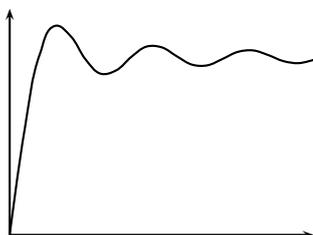
Tracés point par point

vous réutiliser les données. `\fileplot` est moins gourmand en mémoire \TeX que `\readdata` et `\dataplot` si vous utiliser aussi `\PSTtoEPS`.

Voici un tracé point par point de l'intégrale de $\sin x$. les données ont été produites par Mathematica, avec

```
Table[{x,N[SinIntegral[x]]},{x,0,20}]
```

et ensuite copiées dans ce document.



```
\psset{xunit=.2cm,yunit=1.5cm}
\savedata{mydata}[[{0,0},{1,0.946083},{2,1.60541},
{3,1.84865},{4,1.7582},{5,1.54993},{6,1.42469},
{7,1.4546},{8,1.57419},{9,1.66504},{10,1.65835},
{11,1.57831},{12,1.50497},{13,1.49936},{14,1.55621},
{15,1.61819},{16,1.6313},{17,1.59014},{18,1.53661},
{19,1.51863},{20,1.54824}]]
\dataplot[plotstyle=curve,showpoints=true,dotstyle=triangle]
{\mydata}
\psline{<->}(0,2)(0,0)(20,0)
```

`\listplot*[para.] {liste}`

`\listplot` est encore un autre moyen de tracer une courbe point par point en lisant des données. Ici *list* doit être une liste de données (couples de coordonnées), séparées par une espace. *list* est d'abord lue par \TeX et ensuite par PostScript. Cela signifie que *list* doit être un programme PostScript qui laisse sur la pile une liste de données, mais vous pouvez aussi inclure des données qui ont été récupérées avec `\readdata` et `\dataplot`. Cependant, quand vous utilisez les styles *line*, *polygon* ou *dots* avec l'option **showpoints=false**, **linearc=0pt** et sans flèches, `\dataplot` risque beaucoup moins de saturer la mémoire de PostScript ou les limites de la pile que `\listplot`. Dans l'exemple précédent, ces restrictions n'existent pas et ainsi l'exemple est équivalent à l'utilisation de `\listplot` :

```
...
\listplot[plotstyle=curve,showpoints=true,dotstyle=triangle]
{\mydata}
...
```

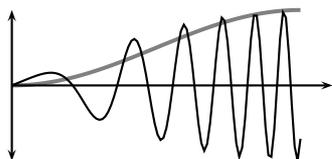
`\psplot*[para.]{xmin}{xmax}{fonction}`

`\psplot` peut être utilisé pour représenter une fonction définie par $f(x) =$ si vous connaissez un peu PostScript. *fonction* doit être écrite en code PostScript, i.e. en notation polonaise inversée (N.P.I.) pour calculer $f(x)$. Notez que vous devez utiliser la variable *x*. PostScript n'est pas fait pour du calcul scientifique, mais `\psplot` est bon pour tracer le graphe de fonctions assez simples. Par exemple :

```
\psplot[plotpoints=200]{0}{720}{x sin}
```

trace le graphe de la fonction $\sin x$ de 0 à 720 degrés, en calculant de façon approximative $\sin x$ tous les 3,6 degrés et en reliant ensuite les points obtenus avec `\psline`.

Voici le tracé de $\sin x \cos \left[\frac{x}{2} \right]^2$ et de $\sin^2 x$.

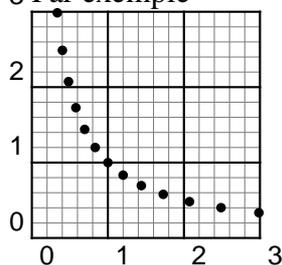


```
\psset{xunit=1.2pt}
\psplot[linecolor=gray,linewidth=1.5pt,
plotstyle=curve]
{0}{90}{x sin dup mul}
\psplot[plotpoints=100]{0}{90}{x sin x 2 div
2 exp cos mul}
\psline{<->}(0,-1)(0,1)
\psline{->}(100,0)
```

`\parametricplot*[para.]{tmin}{tmax}{fonction}`

Cette commande sert à représenter une courbe paramétrée $(x(t); y(t))$. *fonction* représente le code PostScript servant à calculer les couples $(x(t); y(t))$.

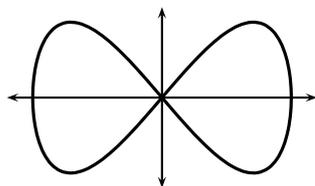
3 Par exemple



```
\parametricplot
[plotstyle=dots,plotpoints=13]{-6}{6}{1.2
t exp 1.2 t neg exp}
```

place 13 points de l'hyperbole $xy = 1$, démarrant à $\left[\frac{1}{2^6}; \frac{1}{2^6} \right]$ et finissant à $\left[\frac{1}{2^6}; \frac{1}{2^6} \right]$.

Voici un tracé point par point de la fonction paramétrique $(\sin(t); \sin(2t))$:



```
\psset{xunit=1.7cm}
\parametricplot[linewidth=1.2pt,
plotstyle=ccurve] {0}{360}{t sin t 2 mul sin}
\psline{<->}(0,-1.2)(0,1.2) \psline{<->}(-
1.2,0)(1.2,0)
```

Le nombre de points calculés par les commandes `\psplot` et `\parametricplot` est défini par le paramètre

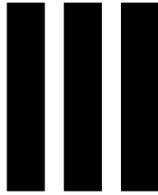
`plotpoints=entier`

Par défaut : 50

Tracés point par point

Utiliser les courbes et ses variantes au lieu de segments et augmenter la valeur de **plotpoints** sont deux moyens d'obtenir une courbe plus régulière. ces deux voies augmentent le temps de traitement. Le meilleur choix dépend de la complexité du calcul.

(Remarquez que toutes les lignes PostScript sont en définitive transformées en des séries (parfois courtes) de segments. Mathematica utilise en général `lineto` pour connecter les points. par défaut le nombre minimal de points dans Mathematica est 25, mais contrairement à `\psplot` et `\parametricplot` Mathematica augmente le nombre de points dans les régions où la fonction a de grandes variations.



Autres paramètres graphiques

Les paramètres graphiques décrits dans ce chapitre sont communs à tous ou à la majorité des objets graphiques.

12 Systèmes de coordonnées

Les manipulations suivantes du système de coordonnées s'applique uniquement à des purs objets graphiques.

Un moyen simple de déplacer l'origine du système de coordonnées à (x, y) est d'utiliser

origin={*coordon.*} **Par défaut : 0pt, 0pt**

C'est la seule fois que les coordonnées doivent être entourées d'accolades plutôt qu'avec des parenthèses.

Un moyen simple d'échanger les axes est d'utiliser le paramètre

swapaxes=true **Par défaut : false**

Ainsi vous pouvez changer d'avis sur l'orientation d'un tracé après avoir généré les données.

13 Styles de lignes

Les paramètres graphiques suivants (en plus de **linewidth** et **linecolor**) déterminent comment les lignes doivent être tracées, aussi bien quand ce sont des tracés ouverts ou fermés.

linestyle=style **Par défaut : solid**

Les styles permis sont `none`, `solid`, `dashed` et `dotted`

dash=dim1 dim2**Par défaut : 5pt 3pt**

Le modèle de tiret *noir et blanc* pour le style de ligne tiretée. Par exemple :



```
\psellipse[linestyle=dashed,dash=3pt](2,1)(2,1)
```

dotsep=dim**Par défaut : 3pt**

La distance entre les points dans le style de ligne en pointillés. Par exemple :



```
\psline[linestyle=dotted,dotsep=2pt]
{ | -> }(4,1)
```

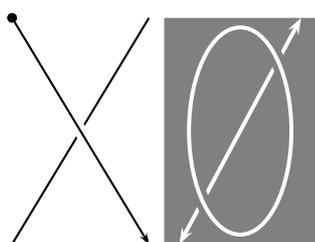
border=dim**Par défaut : 0pt**

Une valeur positive trace un bord de largeur *dim* et de couleur **bordercolor** de chaque côté de la courbe. Ceci est pratique pour donner l'impression qu'une ligne passe au sommet d'une autre. La valeur est sauvée dans le registre de dimension **\psborder**

bordercolor=color**Par défaut : white**

Voir **border** ci-dessus.

Par exemple :



```
\psline(0,0)(1.8,3)
\psline[border=2pt]{ *-> }(0,3)(1.8,0)
\psframe*[linecolor=gray](2,0)(4,3)
\psline[linecolor=white,linewidth=1.5pt]{ <-> }
(2.2,0)(3.8,3) \psellipse[linecolor=white,
linewidth=1.5pt,bordercolor=gray,border=2pt]
(3,1.5)(.7,1.4)
```

doubleline=true/false**Par défaut : false**

Quand on le met à true, une ligne double est tracée, avec un espace de **doublesep** et une couleur **doublecolor**. Ceci ne fonctionne comme espéré avec le style de ligne pointillés et quelques flèches par contre donnent de bons résultats.

doublesep=dim**Par défaut : 1,25 \pslinewidth**

Voir **doubleline** au-dessus.

doublecolor**Par défaut : white**Voir **doubleline** au-dessus.

Voici un exemple de lignes doubles :



```
\psline[doubleline=true,
linear=.5,doublesep=1.5pt]{->}(0,0)(3,1)(4,0)
```

shadow=true/false**Par défaut : false**

Avec `true`, une ombre, à la distance **shadowsize** de la courbe d'origine, dans la direction **shadowangle**, et avec la couleur **shadowcolor**.

shadowsize=dim**Par défaut : 3pt**Voir **shadow** au-dessus.**shadowangle=angle****Par défaut : - 45**Voir **shadow** au-dessus.**shadowcolor=color****Par défaut : darkgray**Voir **shadow** au-dessus.

Voici un exemple de la particularité **shadow**, qui devrait vous être familier.

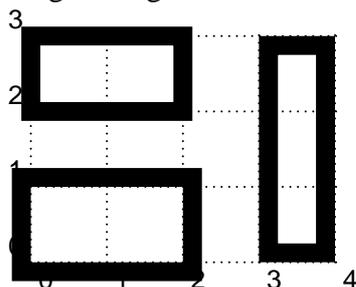


```
\pspolygon[linearc=2pt,shadow=true,
shadowangle=45,xunit=1.1](-1,-.55)
(-1,.5)(-.8,.5)(-.8,.65)
(-.8,.65)(-.2,.5)(1,.5)(1,-.55)
```

Voici un autre paramètre graphique relatif aux lignes mais qui s'applique seulement aux objets graphiques fermés **\psframe**, **\pscircle**, **\psellipse** et **\pswedge** :

dimen=outer/inner/middle**Par défaut : outer**

Il détermine si les dimensions sont relatifs à l'intérieur, à l'extérieur ou au milieu de la frontière. La différence est notable quand la largeur de la ligne est grande :



```
\psset{linewidth=.25cm}
\psframe[dimen=inner](0,0)(2,1)
\psframe[dimen=middle](0,2)(2,3)
\psframe[dimen=outer](3,0)(4,3)
```

Styles de lignes

Avec `\pswedge`, ceci affecte uniquement le rayon ; l'origine reste toujours au milieu de la frontière. La bonne affectation de ce paramètre dépend de la façon dont vous voulez aligner les autres objets.

14 Styles de remplissage

Le nouveau groupe de paramètres graphiques détermine comment les régions fermées sont remplies. Même les courbes ouvertes peuvent être remplies ; ceci n'affecte pas la couleur de la courbe.

`fillstyle=style`

Par défaut : rien

Les styles autorisés sont :

`none`, `solid`, `vlines`, `vlines*`, `hlines`, `hlines*`, `crosshatch`, et `crosshatch*`.

`vlines`, `hlines` et `crosshatch` tracent un motif de lignes selon les quatre paramètres décrits ci-dessous, préfixés par `hatch`. Les versions étoilées remplissent aussi l'arrière-plan, comme dans le style `solid`.

`fillcolor=color`

Par défaut : blanc

la couleur de l'arrière-plan dans les styles `solid`, `vlines*`, `hlines*` et `crosshatch`.

`hatchwidth=dim`

Par défaut : 0.8pt

Largeur des lignes.

`hatchsep=dim`

Par défaut : 4pt

La largeur de l'espace entre les lignes.

`hatchcolor=color`

Par défaut : noir

La couleur des lignes. Conservée dans `\textbfpshatchcolor`.

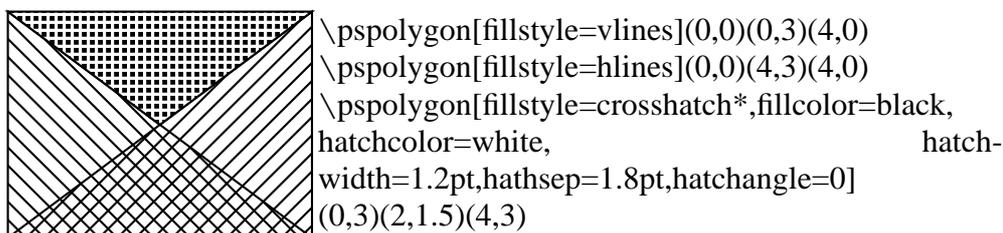
`hatchangle=rot`

Par défaut : 45

L'inclinaison des lignes en degrés. par exemple, si `\hatchangle` est mis à 45, le style `vlines` trace des lignes dans la direction Nord-Ouest-Sud-Est, et le style `hlines` trace des lignes dans la direction Sud-Ouest-Nord-Est et le style `crosshatch` les trace dans les deux directions.

Voici un exemple de `vlines` et des styles de remplissage décrits ci-dessus.

Styles de remplissage



Ne soyez pas surpris si la partie en damier de cet exemple (le dernier `\pspolygon` ressort bien sur les écrans basse résolution. PSTricks ajuste les lignes de façon à ce qu'elles aient la même largeur, mais l'espace entre elles, en l'espèce noir, peut avoir une largeur variable.

Chacun de ces objets purement graphiques (à l'exception de ceux commençant par `q`) a une version étoilée qui produit un objet plein de couleur **linecolor**. (Il met automatiquement **linewidth** à zéro, **fillcolor** à **linecolor**, **fillstyle** à **solid**, et **linestyle** à **none**.)

15 Pointes de flèches et autres

Les lignes et les autres courbes ouvertes peuvent être terminées par différentes pointes de flèches, des barres en T ou des cercles. Le paramètre

arrows=*style*

Par défaut : -

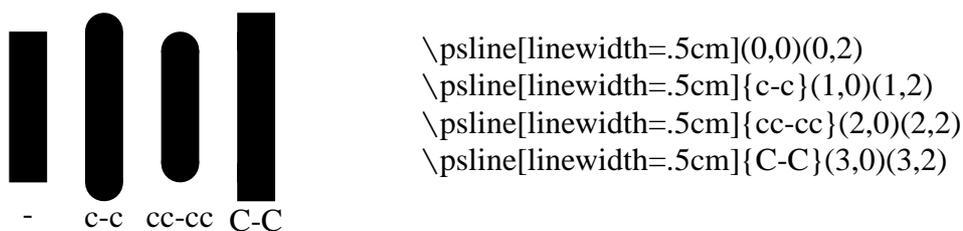
détermine ce que vous voulez. Il peut prendre les valeurs suivantes, qui sont intuitives :⁵

5. Ceci est une sorte de WYSIWYG pour \TeX

Valeur	Exemple	Nom
-		Rien
<->		Têtes de flèche
>-<		Flèches doubles retournées
<<->>		Doubles flèches
>>-<<		Doubles flèches retournées
-		Doubles barres en T au niveau des points extrêmes
* - *		Doubles barres en T centrés aux points extrêmes
[-]		Crochets
(-)		Parenthèses
o - o		Cercles centrés aux extrémités
* - *		Boules centrées aux extrémités
oo - oo		Cercles au niveau des points extrêmes
** - **		Boules au niveau des points extrêmes
c - c		Extrémités arrondies
cc - cc		Extrémités arrondies
C - C		Extrémités carrées

Vous pouvez aussi mélanger et apparier ; i.e. `->` , `*-)` et `[-)` sont tous des choix valides du paramètre **arrows**.

Cependant les flèches `c`, `cc` et `C` ne sont pas aussi évidentes. `c` et `C` correspondent au codage Postscript `linecap` respectivement à 1 et 2. `cc` est comme `c`, mais ajusté pour que la ligne s'étende jusqu'au point final. Ces styles de flèches sont bien visibles quand le **linewidth** est important :



Pratiquement toutes les courbes ouvertes vous permettent d'inclure les paramètres **arrows** comme argument optionnel, enfermés dans des accolades et avant tout autre argument (excepté les arguments de paramètres optionnels). i.e. à la place de

```
\psline[arrows=<-,linestyle=dotted](3,4)
```

vous pouvez écrire

```
\psline[linestyle=dotted]{<-}(3,4)
```

Les exceptions ne concernent que quelques macros qui ne supportent pas l'usage des flèches (elles commencent toutes par la lettre `q`).

La taille de ces fins de ligne est contrôlée par les paramètres suivants. Dans la description de ces paramètres, la largeur se réfère toujours à la dimension perpendiculaire à la ligne, et la longueur à la dimension dans la direction de la ligne.

arrowsize=*dim num* **Par défaut : 2pt 3**

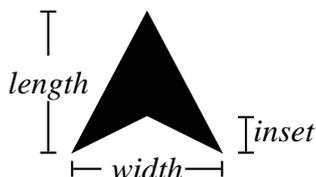
Largeur de la pointe de la flèche, comme décrit ci-dessous.

arrowlength=*num* **Par défaut : 1.4**

Longueur de la pointe de la flèche, comme décrit ci-dessous.

arrowwinset=*num* **Par défaut : 0.4**

Taille du rentré pour la pointe de la flèche, comme décrit ci-dessous.



$$\text{arrowsize} = \text{dim num}$$

$$\text{width} = \text{num} \times \text{linewidth} + \text{dim1}$$

$$\text{length} = \text{arrowlength} \times \text{width}$$

$$\text{inset} = \text{arrowwinset} \times \text{height}$$

tbar size=*dim num* **Par défaut : 2pt 5**

La largeur d'une barre en T, d'un crochet ou d'une parenthèse est *num* multipliée par **linewidth** plus *dim*.

bracketlength=*num* **Par défaut : 0.15**

La hauteur d'un crochet est *num* multipliée par sa largeur.

rbracketlength=*num* **Par défaut : 0.15**

La hauteur d'une parenthèse est *num* multipliée par sa largeur.

dotsize=*dim num* **Par défaut : .5pt 2.5**

Le diamètre d'un cercle ou d'un disque est *num* multipliée par **linewidth** plus *dim*.

arrowscale=*arrowscale=num1 num2* **Par défaut : 1**

Imaginer que la flèche pointe vers le bas. Ce paramètre multiplie la largeur des flèches par *num1* et la longueur (ou la hauteur) par *num2*. Si vous ne mettez qu'un seul nombre, les dimensions des flèches seront multipliées par ce nombre. Changer **arrowscale** vous permet d'obtenir des effets spé-

ciaux impossibles en changeant les paramètres décrits ci-dessus. i.e. vous pouvez changer la largeur des lignes utilisées pour dessiner les parenthèses ou les crochets.

16 Styles personnels

Vous pouvez définir des versions personnelles de n'importe quelle macro qui a comme premier argument optionnel des changements de paramètres en utilisant la commande `\newsobject` :

```
{\newsobject{name}{object}{par1=value1,...}}
```

comme dans

```
\newsobject{myline}{psline}{linecolor=green,linestyle=dotted}
\nnewsobject{mygrid}{psgrid}{subgridiv=1,
griddots=10,gridlabels=7pt}
```

Le premier argument est le nom de la nouvelle commande que vous voulez définir. Le deuxième argument est le nom de l'objet graphique à redéfinir. Noter que ces deux noms ne sont pas précédés de la contre-oblique. Le troisième argument contient les valeurs des paramètres que vous voulez définir.

Avec les exemples ci-dessus, la commande `\myline` fonctionne exactement comme l'objet graphique `\psline` sur lequel elle est basée, et vous pouvez même redéfinir les paramètres que vous aviez fixés dans la définition de `\myline`, comme dans :

```
\myline[linecolor=gray,dotsep=2pt](5,6)
```

Un autre moyen de définir des configurations avec des paramètres personnels est d'utiliser la commande

```
\newsstyle{name}{par1=value1,...}
```

Vous pouvez fixer le **style** de paramètres graphiques avec *name*, plutôt que d'établir les paramètres dans le deuxième argument de `\newsstyle`. Par exemple,

```
\newsstyle{mystyle}{linecolor=green,linestyle=dotted}
\psline[style=mystyle](5,6)
```

IV Graphiques personnels

17 Les bases

PSTricks possède une large palette d'objets graphiques, mais quelquefois vous avez besoin de quelque chose de spécial. Par exemple vous voulez ombrer la région entre deux courbes. La commande

```
\pscustom*[par]{commands}
```

vous permet de créer vos propres objets graphiques. Un *path* est une ligne, dans le sens mathématique plutôt que dans le sens visuel. Un chemin (path) peut posséder plusieurs segments discontinus, et peut être ouvert ou fermé. Postscript a des opérateurs divers pour créer des chemins. La fin du chemin est appelé le *point courant*, mais s'il n'y a pas de chemin, il n'y a pas de point courant. Pour rendre le chemin visible, Postscript peut remplir la région entourée par le chemin (ce que **fillstyle** et autres font), et tracer le chemin (ce que **linestyle** et autres font).

Au départ de **\pscustom**, il n'y a pas de chemin. Il y a plusieurs commandes que vous pouvez utiliser pour tracer des chemins. Quelques unes (les courbes ouvertes) permettent aussi de tracer des flèches. **\pscustom** remplit et trace le chemin à la fin, et pour des effets spéciaux, vous pouvez remplir ou tracer le chemin en utilisant **\psfill** ou **\psstroke** (voir plus bas).

Notes pour le driver : **\pscustom** utilise **\pstverb** et **\pstunit**. Il y a des limites dépendantes du système à la longueur de l'argument de **\special**. Vous pouvez rester dans les limites en utilisant **\pscustom** car tout le code Postscript accumulé par **\pscustom** est l'argument d'une seule commande **\special**.

18 Les paramètres

Vous devez penser à séparer le traçage et le remplissage des chemins quand vous fixez les paramètres. Les paramètres **linewidth** et **linecolor** affectent le tracé des flèches, mais comme les commandes de chemin ne relient ni ne remplissent les chemins, ces paramètres ainsi que **linestyle**, **fillstyle** n'ont pas d'autre effet (excepté dans quelques cas où **linewidth** est utilisé dans quelques calculs pour tracer les chemins). **\pscustom** et **\fill** utilisent **fillstyle** et les paramètres apparentés, et **\pscustom** et **\stroke** uti-

lisent `plinestyle` et les paramètres apparentés.

Par exemple, si vous incluez

```
\psline[linewidth=2pt,linecolor=blue,fillstyle=vlines]{<-}(3,3)(4,0)
```

dans `\pscustom`, alors les modifications de **linewidth** et de **linecolor** porteront sur la taille et la couleur de la flèche mais pas sur la ligne quand elle est tracée et les modifications de **fillstyle** n'auront pas d'effet du tout.

Les paramètres **shadow**, **border**, **doubleligne** et **showpoints** sont sans effet dans `\pscustom`, et les paramètres **origin** et **swapaxes** affectent `\pscustom` lui-même, mais il y a des commandes (décrites ci-dessous) qui vous permettra de réaliser ces effets spéciaux.

Les styles de ligne **dashed** et **dotted** nécessitent de connaître quelque chose sur le chemin de façon à ajuster correctement les traits ou les points. Vous pouvez donner cette information en fixant le paramètre

linetype=*int*

Par défaut : 0.

Si le chemin contient plus d'un segment discontinu, il n'y a pas de moyen d'ajuster le style de traits ou de points et il vaut mieux laisser la valeur par défaut de **linetype**. Voici les valeurs pour des chemins simples :

<i>Valeur</i>	Type de chemin
0	Courbe ouverte dans flèches.
-1	Courbe ouverte avec flèche au début.
-2	Courbe ouverte avec flèche à la fin.
-3	Courbe ouverte avec flèches de chaque côté.
1	Courbe fermée sans symétrie particulière
$n > 1$	Courbe fermée avec n segments symétriques

19 Objets graphiques

Vous pouvez utiliser la plupart des objets graphiques dans `\pscustom`. Ils traceront des chemins et feront des flèches, mais ne rempliront ni ne relieront les chemins

Il y a trois types d'objets graphiques :

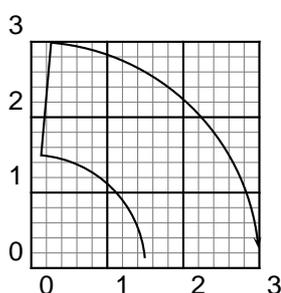
Special Les objets graphiques spéciaux incluent `\psgrid`, `\psdots`, `\qline` et `\qdisk`. Vous ne pouvez pas utiliser d'objets graphiques spéciaux dans `\pscustom`.

Closed Vous pouvez utiliser des objets graphiques fermés dans `\pscustom`

mais leurs effets sont imprévisibles.⁶ Généralement vous utiliserez des courbes ouvertes plus `\closepath` (voir plus bas) pour tracer des courbes fermées.

Open Les objets graphiques ouverts sont les commandes les plus utilisées pour tracer des chemins avec `\pscustom`. En assemblant plusieurs courbes ouvertes, vous pouvez tracer des chemins arbitraires. le reste de cette section concerne les objets graphiques ouverts.

Par défaut, les courbes ouvertes tracent une ligne droite entre le point courant, s'il existe et le début de la courbe, excepté quand la courbe commence par une flèche. Par exemple



```
\pscustom{ \psarc(0,0){ 1.5}{5}{85}
\psarcn{->}(0,0){3}{85}{5}}
```

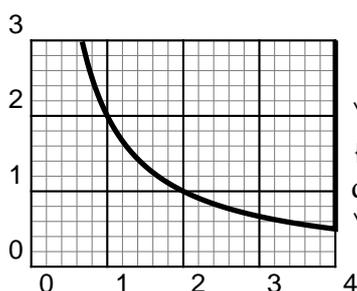
Également, les courbes suivantes utilisent le point courant, s'il existe, comme une première coordonnée :

`\psline` et `\pscurve`.

les commandes `-plot`, avec le `plotstyle` de ligne ou de courbe.

`\psbezier` si vous incluez seulement trois couples de coordonnées.

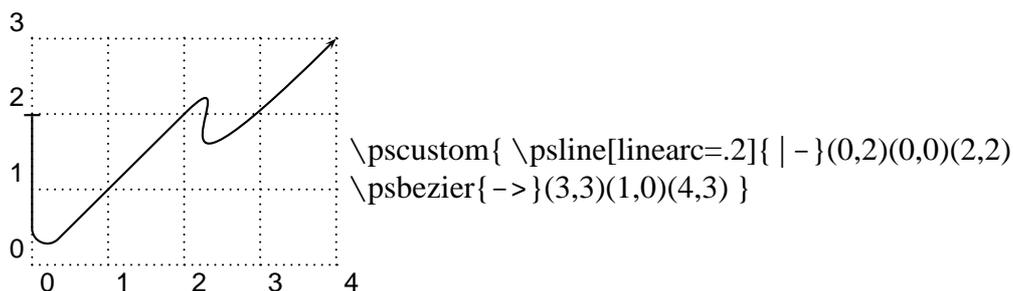
Par exemple :



```
\pscustom[linewidth=1.5pt]
{\psplot[plotstyle=curve]{.67}{4}{2 x
div}
\psline(4,3)}
```

Nous verrons plus tard comment faire ceci de façon plus intéressante. Voici un autre exemple :

6. Les objets fermés n'utilisent jamais le point courant comme une coordonnée, mais habituellement ils entoureront tout chemin existant, et ils peuvent tracer une ligne entre le point courant et la courbe fermée.

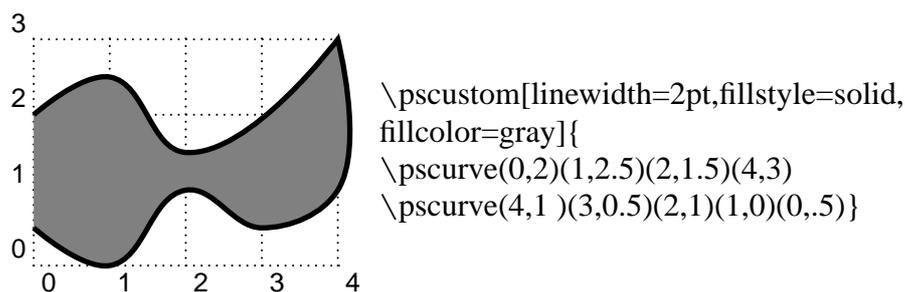


Cependant vous pouvez traiter la façon dont les courbes ouvertes traitent le point courant avec le paramètre

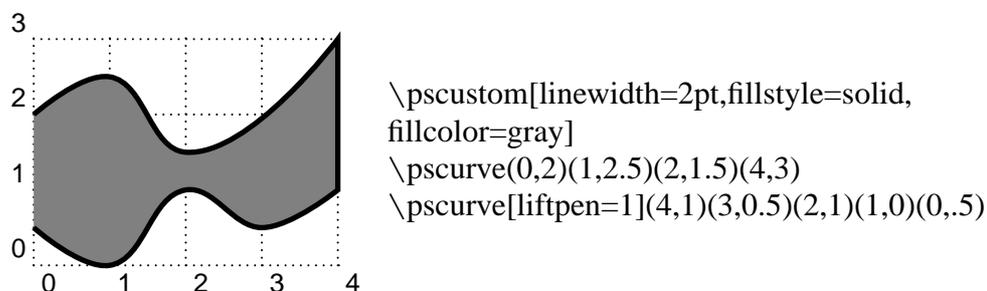
liftpen=0/1/2

Par défaut : 0.

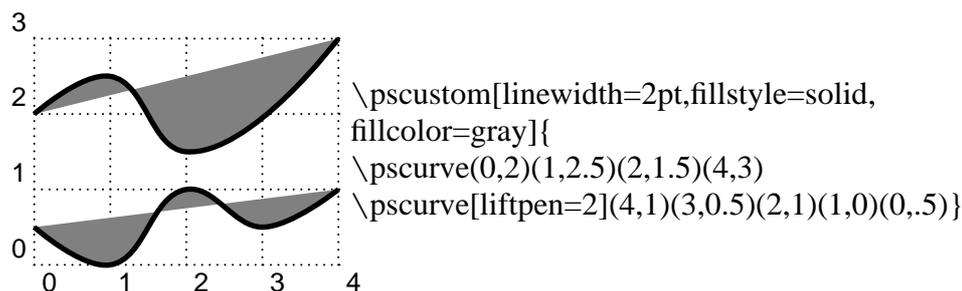
Si **liftpen=0**, vous avez le comportement par défaut décrit au-dessus. Par exemple :



Si **liftpen=1**, les courbes n'utilisent pas le point courant comme premières coordonnées (exception de **\psbezier**, mais vous pouvez empêcher ceci en incluant de façon explicite les premières coordonnées comme un argument). Par exemple :



Si **liftpen=2**, les courbes n'utilisent pas le point courant comme premières coordonnées, et elles ne tracent pas une ligne entre le point courant et le début de la courbe. Par exemple :



Plus loin nous utiliserons le deuxième exemple pour colorer la région entre deux courbes et ensuite tracer les courbes.

20 Astuces de sécurité

Les commandes décrites sous ce titre, qui peuvent être uniquement dans `\pscustom`, ne font pas courir de risque d'erreurs PostScript (en supposant que votre document compile sans erreurs \TeX).

Commençons avec quelques commandes de chemins, de remplissage et de rupture :

`\newpath`

Supprime le chemin et le point courant.

`\moveto(coor)`

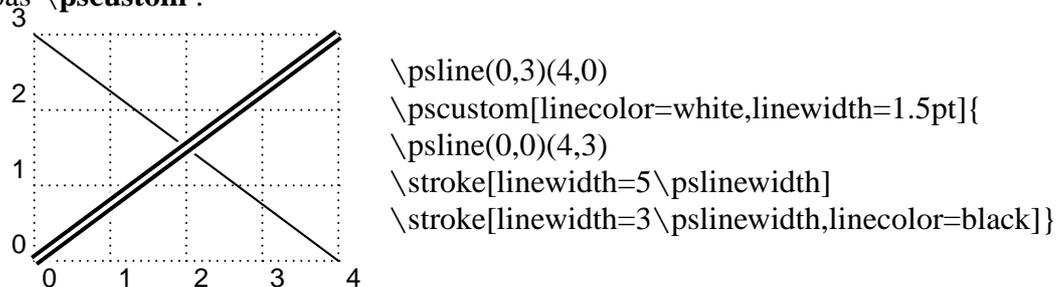
Ceci amène le point courant à (x, y) .

`\closepath`

Ceci ferme le chemin en joignant le début et la fin de chaque morceau (il peut y avoir plus d'un morceau si vous utilisez `\moveto`).⁷

`\stroke[par]`

Ceci rompt le chemin (de façon non destructive). `\pscustom` rompt automatiquement le chemin, mais vous souhaitez peut-être le rompre deux fois, i.e. ajouter une bordure. Voici un exemple qui trace une double ligne et rajoute une bordure (cet exemple est si simple qu'il ne nécessite même pas `\pscustom` :



7. Noter que le chemin est automatiquement fermé quand la région est remplie. Utiliser `closepath` si vous voulez aussi fermer la frontière.

\fill[par]

Ceci remplit la région (de façon non destructive). **\pscustom** remplit de façon automatique la région le mieux possible.

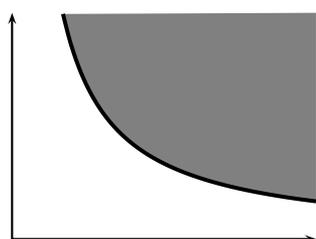
\gsave

Ceci conserve l'état des graphiques courants (i.e. le chemin, la couleur, la largeur de la ligne, le système de coordonnées ; etc.) **\grestore** restaure les graphiques. **\gsave** et **\grestore** doivent être utilisés par paire, emboîtés correctement en respectant les groupes **T_EX**.

\grestore

Voir ci-dessus.

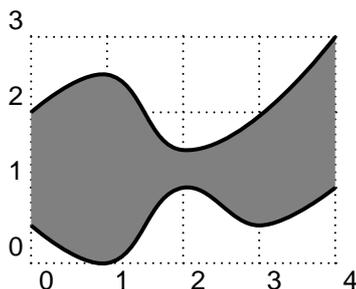
Voici un exemple qui améliore un exemple précédent, en utilisant **\gsave** et **\grestore** :



```
\psline{<->}(0,3)(0,0)(4,0)
\pscustom[linewidth=1.5pt]{
\psplot[plotstyle=curve]{.67}{4}{2
x div} \gsave
\psline(4,3)
\fill[fillstyle=solid,fillcolor=gray]
\grestore}
```

Observez que la ligne ajoutée par **\psline(4,3)** n'est jamais rompue, car elle est emboîtée entre **\gsave** et **\grestore**.

Voici un autre exemple :



```
\pscustom[linewidth=1.5pt]{
\pscurve(0,2)(1,2.5)(2,1.5)(4,3)
\gsave
\pscurve[liftpen=1](4,1)(3,0.5)(2,1)(1,0)(0,.5)
\fill[fillstyle=solid,fillcolor=gray]
\grestore}
\pscurve[linewidth=1.5pt](4,1)(3,0.5)(2,1)(1,0)(0,.5)
```

Noter comment j'ai dû répéter le **\pscurve** (j'aurais pu le répéter à l'intérieur de **\pscustom**, avec **liftpen=2**), car je voulais tracer une ligne entre les deux courbes pour enfermer la région mais je ne voulais pas briser cette ligne.

Le nouveau groupe de commandes modifie le système de coordonnées.

\translate(coor)

Translate le système de coordonnées de (x, y) . Cela déplace tout ce qui vient après (x, y) , mais n'affecte pas ce qui a déjà été tracé.

`\scale{num1 num2}`

Met à l'échelle le système de coordonnées avec les facteurs *num1 num2* ou horizontalement avec *num1* ou verticalement avec *num2*.

`\rotate{angle}`

Effectue une rotation du système de coordonnées de *angle*.

`\swapaxes`

Échange les coordonnées *x* et *y*. Équivalent à

`\rotate{-90}`
`\scale{-1 1 scale}`

`\msave`

Sauvegarde le système de coordonnées. Vous pouvez ensuite le restaurer avec `\mrestore`. Vous devez avoir emboîté les paires `\msave-mstore`. `\msave` and `\mrestore` n'ont pas besoin d'être correctement emboîtés en respect des groupes \TeX , ni de `\gsave` ou de `\grestore`. Cependant souvenez-vous que `\gsave` et `\grestore` affectent aussi le système de coordonnées. `\msave-mrestore` vous permet de changer le système de coordonnées, tout en dessinant une partie de chemin, et ensuite restaurer l'ancien système de coordonnées sans détruire le chemin. `\gsave-grestore`, par contre, affectent le chemin et tous les autres composants de l'état graphique.

`\mrestore`

Voir au-dessus ;

Et maintenant quelques astuces pour les ombres :

`\openshadow[par]`

Romp une copie du chemin courant, en utilisant les différents paramètres d'ombre.

`\closedshadow[par]`

Donne une ombre de la région enfermée par le chemin courant comme si c'était des régions opaques.

`\movepath[coor]`

Déplace le chemin de (x, y) . Utiliser `\gsave-\restore` si vous ne voulez pas perdre le chemin original.

21 Astuces assez sûres

Le groupe de commandes suivant est sûr, *aussi longtemps qu'il y a un point courant!*

\lineto(*coor*)

C'est une version rapide de **\lineto(*coor*)**.

\rlineto(*coor*)

Comme **\lineto**, mais (x, y) est interprété relativement au point courant.

\curveto(*x1,y1*)(*x2,y2*) (*x3,y3*)

C'est une version rapide de **\psbezier** $(x1,y1)(x2,y2)(x3,y3)$.

\rcurveto(*x1,y1*)(*x2,y2*) (*x3,y3*)

Comme **\curveto**, mais $(x1,y1)$, $(x2,y2)$ et $(x3,y3)$ sont interprétés relativement au point courant.

22 Pour les « hackers » seulement

Pour les « hackers » de PostScript, il y a quelques autres commandes. Lisez à tout prix d'abord l'appendice C avant d'utiliser ces commandes. Inutile de dire :

Attention : une mauvaise utilisation de ces commandes peut provoquer des erreurs PostScript.

L'environnement PostScript mis en ?uvre avec **\pscustom** a une unité égale à un point (pt) \TeX .

\code{*code*}

Insère le code PostScript brut.

\dim{*dim*}

Convertit les dimensions PSTricks en nombre de points , et les insère dans le code PostScript.

\coor(*x1, y1*)(*x2, y2*)... (*xn, yn*)

Convertit une ou plusieurs coordonnées PSTricks en paires de nombres (en unités points), et les insère dans le code PostScript.

\rcoor(*x1, y1*)(*x2, y2*)... (*xn, yn*)

Comme **\coor**, mais insère les coordonnées dans l'ordre inverse.

\file{fichier}

Comme `\code`, mais le PostScript brut est copié mot pour mot (excepté les commentaires delimités par %) du *fichier*.

\arrows{flèches}

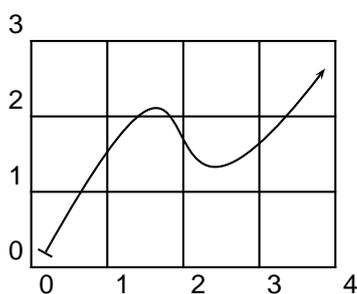
Ceci définit les opérateurs PostScript ArrowA et ArrowB de la façon suivante :

```
x2 y2 x1 y1 ArrowA
x2 y2 x1 y1 ArrowB
```

Chacun dessine une tête de flèche avec le bout à $(x1, y1)$ et la pointe à $(x2, y2)$. ArrowA laisse le point courant à la fin de la tête de la flèche, d'où un connecteur de ligne partira, et laisse $(x2, y2)$ sur la pile. ArrowB ne change pas le point courant, mais laisse

```
x2 y2 x1' y1'
```

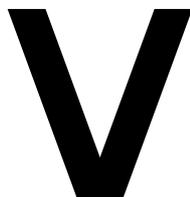
sur la pile, où $(x1', y1')$ est le point où un connecteur de ligne joindra. Pour donner une idée de comment ceci fonctionne, le code suivant montre grossièrement comment PSTricks trace une courbe de Bézier avec des flèches aux extrémités :



```
\pscustom{
\arrows{|->} \code{
80 140 5 5 ArrowA
30 -30 110 75 ArrowB
curveto}}
```

\setcolor{couleur}

Fixe la couleur à *couleur*.



Outils d'images

23 Images

Les objets graphiques, `\rput` et ses variantes ne changent pas le point courant dans le document \TeX (c'est-à-dire qu'ils créent une boîte de dimensions 0). Si vous mettez ensemble plusieurs de ces objets (et d'autres objets de dimensions 0) ils partagent le même système de coordonnées, et ainsi vous pouvez créer une image. Pour cette raison, ces macros sont appelées *objets d'image*.

Si vous créez une image ainsi, vous voudrez certainement donner à celle-ci une certaine dimension. Vous pouvez le faire en plaçant ces objets d'image dans l'environnement `\pspicture`, comme dans :

```
\pspicture*[baseline](x0, y0) (x1, y1)
objets d'image
\endpspicture
```

Les objets d'image sont placés dans un rectangle dont le coin inférieur gauche est à $(x0, y0)$ (par défaut $(0,0)$) et le coin supérieur droit à $(x1, y1)$. Par défaut la ligne de base est placée au bas du rectangle, mais l'argument optionnel `[baseline]` place la partie de la ligne de base, *baseline* à partir du bas. Ainsi, *baseline* est un nombre, généralement, mais pas nécessairement entre 0 et 1. Si vous mettez cette option mais la laissez vide (`[]`), alors la ligne de de base passe par l'origine.

Normalement, les objets d'image peuvent dépasser les limites du rectangle. Cependant, si vous mettez l'étoile (*), tout ce qui est en dehors du rectangle est supprimé.

En plus des objets d'image, vous pouvez mettre dans `\pspicture` tout ce qui ne prend pas de place. Ainsi vous pouvez faire des déclarations de fonte, utiliser `\psset` et vous pouvez grouper ces déclarations entre parenthèses. PSTricks vous avertira si vous incluez quelque chose qui prend trop de place.⁸

Les utilisateurs de \LaTeX pourront écrire

```
\begin{pspicture} ... \end{pspicture}
```

Vous pouvez utiliser des objets d'image de PSTricks dans un environnement d'image \LaTeX et inversement vous pouvez des images \LaTeX dans un

8. Quand les objets d'image PSTricks sont inclus dans un environnement `\pspicture` ils avalent tout espace qui suit ou qui précède. (Les objets PSTricks ignorent toujours les espaces qui suivent. Si vous voulez neutraliser des espaces précédentes quand vous n'êtes pas dans un environnement `\pspicture` (c'est-à-dire dans un environnement `\picture` de \LaTeX), utilisez la commande `KillGlue`. La commande inverse est `DontKillGlue`

environnement `\pspicture` de PSTricks. Cependant l'environnement `\pspicture` rend obsolète l'environnement `picture` de \LaTeX et a quelques petits avantages. Notez cependant que les arguments de l'environnement `\pspicture` fonctionnent différemment de ceux de \LaTeX .

Notes pour le driver : l'option de coupure (*) utilise `\pstVerb` et `\pstverbscale`.

24 Placer et tourner ce que l'on veut

PSTricks possède plusieurs commandes permettant de placer et de faire tourner un objet en mode H-R. Toutes ces commandes finissent par `put` et ont des similitudes avec la commande \LaTeX `\put`, mais ont d'avantage de possibilités. Comme la commande `\put` de \LaTeX et contrairement aux macros de rotation de boîtes décrites au paragraphe 29, ces commandes ne prennent pas de place. Elles peuvent être utilisées à l'intérieur ou à l'extérieur d'un environnement `\pspicture`.

La plupart des commandes `put` de PSTricks sont de la forme :

```
\put*argu.{rotation}(coordonnées) {objet}
```

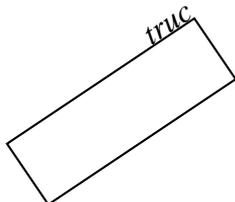
Avec l'option *, l'objet est d'abord placé dans une

```
\psframebox*[boxsep=false]{<objet>}
```

absorbant ainsi tout ce qui suit l'objet. Ceci est utile pour placer du texte au-dessus de quelque chose d'autre.

argu. fait référence à d'autres arguments variant d'une commande `put` à l'autre. L'option *rotation* désigne l'angle dont on veut faire tourner l'objet ; cet argument fonctionne pratiquement de la même façon pour toutes les commandes `put` et est décrit plus bas. L'argument (*coordonnées*) désigne les coordonnées du point de placement de l'objet, mais varie en fait pour chaque commande `put`. cet argument est marqué comme étant obligatoire, mais vous pouvez l'omettre si vous utilisez l'argument *rotation*

L'argument *rotation* doit être un angle comme décrit au paragraphe 4, mais cet angle peut être précédé par une *. Ceci concerne toutes les rotations (exceptées les rotations de boîtes décrites au paragraphe 29) à l'intérieur desquelles une commande `\rput` a été placée pour annuler cette rotation. Ceci est principalement utilisé pour placer un texte horizontalement à l'extrémité d'un objet qui a subi une rotation. Par exemple :



```
\rput{34}{\psframe(-1,0)(2,1)}
\rput[br]{*0}(2,1){\em truc }
```

Il y a aussi des lettres abrégées pour les commandes d'angles. Les voici :

<i>Lettre</i>	<i>Raccourci pour</i>	<i>équivalent à</i>	<i>Lettre</i>	<i>Raccourci pour</i>	<i>équivalent à</i>
U	Haut	0	N	Nord	*0
L	Gauche	90	W	Ouest	*90
D	Bas	180	S	Sud	*180
R	Droite	270	E	Est	*270

Ce paragraphe décrit seulement deux des commandes **put** de PSTricks. La plus basique est

```
\rput*[pointderéférence]{rotation}(x,y){baratin}
```

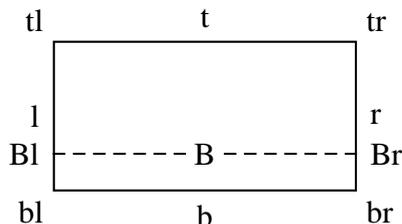
pointderéférence détermine le point de référence de *baratin*, et ce point est translaté de (x, y) .

Par défaut le point de référence est le centre de la boîte. Il peut être modifié en incluant une ou deux des commandes suivantes dans l'argument optionnel *pointderéférence* :

<i>Horizontal</i>	<i>Vertical</i>
l Gauche	t Haut
r Droit	b Bas
	B ligne de base

Visuellement voici où est placé le point de référence dans les différentes combinaisons (la ligne tiretée est la ligne de base) :

Placer et tourner ce que l'on veut



Voici une note marginale

Il y a de nombreux exemples de `\rput` dans ce document, mais pour l'instant voici un seul exemple :

```
\rput[b]{90}{Voici une note marginale}
```

Un des usages courants d'une macro comme `\rput` est de placer des étiquettes sur des objets. PSTricks a une variante spécialement destinée aux étiquettes :

```
\uput*{labelsep}[anglederéférence] {rotation}(x,y){étiquette}
```

Ceci place *étiquette* à la distance *labelsep* du point (x, y) dans la direction **anglederéférence**.

La valeur par défaut de *labelsep* est la dimension du registre

```
\pslabelsep,
```

que vous pouvez changer avec :

```
labelsep=dimension
```

Par défaut : 5 pt

(mais souvenez-vous que `\uput` a un argument optionnel pour établir les paramètres).

Voici un exemple simple :

```
(1,1)
.
\qdisk(1,1){1pt} \uput[45](1,1){(1,1)}
```

Voici un exemple plus intéressant où `\uput` est utilisé pour faire un « camembert » :⁹

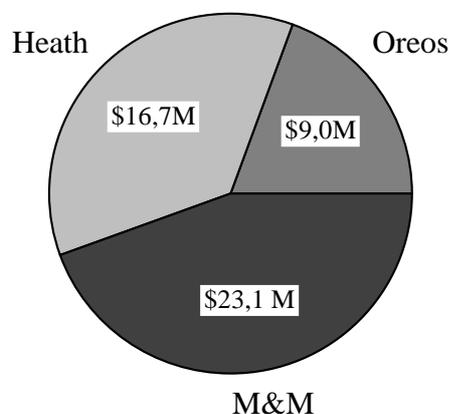
```
\psset{unit=1.2cm}
\pspicture(-2.2,-2.2)(2.2,2.2)
\pswedge[fillstyle=solid,fillcolor=gray]{2}{0}{70}
```

9. PSTricks est distribué avec un outil puissant pour convertir des données en camembert : `piechart.sh`. C'est un script UNIX écrit par Denis Girou.

```

\pswedge[fillstyle=solid,fillcolor=lightgray]{2}{70}{200}
\pswedge[fillstyle=solid,fillcolor=darkgray]{2}{200}{360}
\SpecialCoor
\pssetframesep=1.5pt
\rput(1.2;35){\psframebox*{\small$9,0M}}
\uput{2.2}[45](0,0){Oreos}
\rput(1.2;135){\psframebox*{\small$16,7M}}
\uput{2.2}[135](0,0){Heath}
\rput(1.2;280){\psframebox*{\small$23,1 M}}
\uput{2.2}[280](0,0){M&M}
\endpspicture

```



Vous pouvez utiliser les abréviations suivantes pour *refangle*, qui indique dans quelle direction l'angle pointe : ^{10 11}

10. Utiliser les abréviations quand c'est possible est plus efficace

11. Il y a une commande obsolète **Rput** qui a la même syntaxe que **uput** et cela fonctionne pratiquement de **rput**, et il donne le point dans *stuff* qui doit être aligné avec (x, y) . i.e.,

`qdisk(4,0){2pt}Rput[br](4,0){(x,y)}`

(x, y)

•

Voici les équivalences entre les abréviations *refangle* de **uput** et les abréviations *refpoint* de **Rput** :

uput	r	u	l	ur	ul	dr	dl	
Rput	l	b	r	t	bl	br	tr	rl

Certains préfèrent les conventions de **Rput** pour spécifier la position du contenu à celles de **uput**.

<i>lettre</i>	<i>raccourci pour</i>	<i>équiv. à</i>	<i>lettres</i>	<i>équiv. à</i>	
r	right	0	ur	up-right	45
u	up	90	ul	up-left	135
l	left	180	dl	down-left	225
d	down	270	dr	down-right	315

Le premier exemple peut ainsi s'écrire :

(1,1)

`\qdisk(1,1){1pt}`

`\uput[ur](1,1){(1,1)}`

Notes pour le driver : les macros de rotation utilisent `\pstverb` et `\pstrotate`.

25 Répétition

La macro

`\multirput*[point de référence]{angle}(x0,y0)(x1,y1){entier} {matériel}`

est une variante de `\rput` qui pose *entier* copies, démarrant à $(x0,y0)$ et avançant par pas de $(x1,y1)$ à chaque fois. $(x0,y0)$ et $(x1,y1)$ sont toujours interprétés comme des couples de coordonnées cartésiennes. Par exemple :

* * * * *
* * * * *

`\multirput(.5,0)(.3,.1){12}{*}`

Si vous voulez des copies de graphiques purs, il est plus efficace d'utiliser

`\multips{angle}(x0,y0)(x1,y1) {entier} {graphics}`

graphics peut être un ou plusieurs objets graphiques décrits dans la partie II, ou bien un `\pscustom`. Noter que `\multips` a la même syntaxe que `\multirput` excepté qu'il n'y a pas d'argument *point de référence* (car les graphiques ont des dimensions nulles). Aussi, contrairement à `\multirput`, les coordonnées peuvent être de tout type. Un avertissement `Overflow` `\hbox` indique que l'argument de l'objet graphique contient une sortie superflue ou un espace. Par exemple :

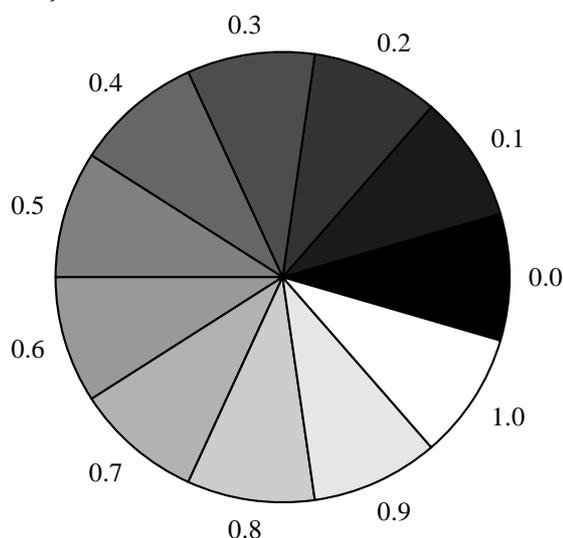
 `\def\zigzag{\psline(0,0)(.5,1)(1.5,-1)(2,0)}`
`\psset{unit=.25,linewidth=1.5pt}`
`\multips(0,0)(2,0){8}{\zigzag}`

Répétition



PSTricks est distribué avec une macro de boucle beaucoup plus générale, appelée `\multido`. Vous devez intégrer le fichier `multido.tex` ou `multido.sty`. Regardez la documentation `multido.doc` pour les détails. Voici un exemple de ce que vous pouvez faire :

```
\begin{pspicture}(-3.4,-3.4)(3.4,3.4)
\newgray{mygray}{0} % Initialise 'mygray' pour l'intérêt
\psset{fillstyle=solid,fillcolor=mygray} % de cette ligne.
\SpecialCoor
\degrees[1.1]
\multido{\n=0.0+.1}{11}{
\newgray{mygray}{\n}
\rput{\n}{\pswedge{3}{-.05}{.05}}
\uput{3.2}{\n}(0,0){\small\n}
}
\end{pspicture}
```



Toutes ces macros de boucles peuvent être emboîtées.

26 Axes



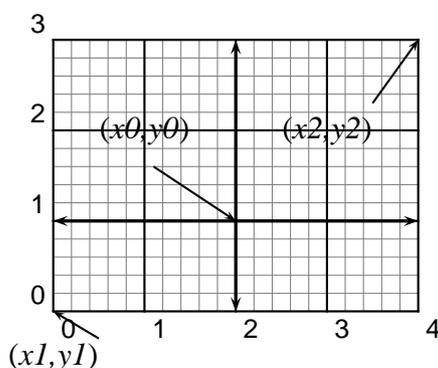
La commande d'axes décrite dans cette section est définie dans `pst-plot.tex` / `pst-plot.sty`, que vous devez appeler d'abord. `pst-plot.tex`, en marche appellera automatiquement `multido.tex`, qui est utilisé pour placer les labels sur les axes.

La macro pour tracer des axes est :

```
\psaxes*[par]{arrows}(x0,y0)(x1,y1)(x2,y2)
```

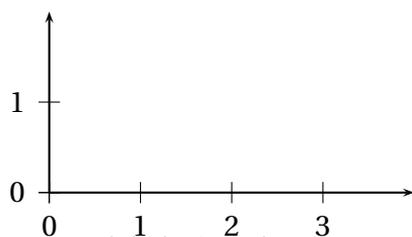
Les coordonnées doivent être cartésiennes. Elles fonctionnent de la même façon qu'avec `\psgrid`. c'est-à-dire que si vous imaginez que les axes sont

à l'intérieur d'un rectangle, $(x1,y1)$ et $(x2,y2)$ sont les coordonnées des sommets opposés. (i.e., que l'axe des abscisses va de $x1$ à $x2$ et celui des ordonnées de $y1$ à $y2$). Les axes ont pour point commun $(x0,y0)$. Par exemple :



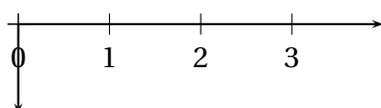
```
\psaxes[linewidth=1.2pt,labels=none,
ticks=none]{<->}(2,1)(0,0)(4,3)
```

Si $(x0,y0)$ est omis, alors l'origine est en $(x1,y1)$. Si $(x0,y0)$ et $(x1,y1)$ sont omis, $(0,0)$ pris comme origine par défaut. Par exemple, quand les axes entourent un simple domaine orthonormé, seul $(x2,y2)$ is nécessaire :



```
\psaxes{->}(4,2)
```

Les labels (nombres) sont mis près des axes, du même côté que $x1$ et $y1$. Ainsi, si nous encadrons un repère orthonormé différent, les nombres se retrouveront à la bonne place :

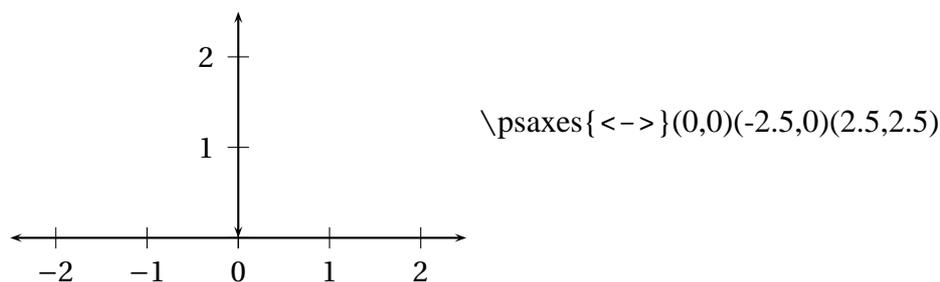


```
\psaxes{->}(4,-1)
```

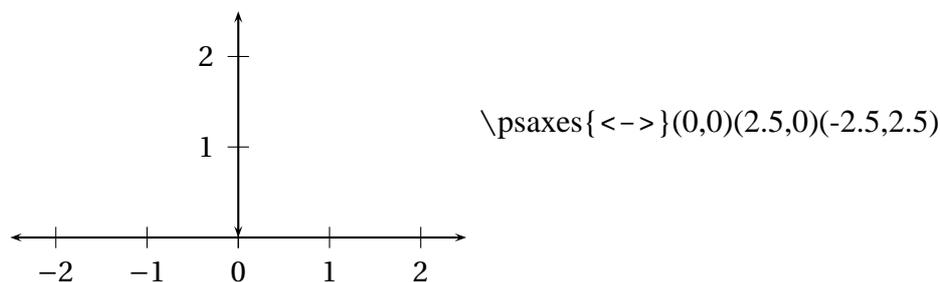
Aussi, si vous fixez le paramètre **arrows**, la première flèche est utilisée pour les pointes en $x1$ et $y1$, tandis que la seconde flèche est utilisée pour les pointes en $x2$ et $y2$. Ainsi, dans les exemples précédents, les pointes de flèche se retrouvent à la bonne place aussi.¹²

Quand les axes n'entourent pas un simple domaine orthonormé, c'est-à-dire quand l'origine n'est pas un coin, il y a une certaine latitude pour disposer les labels. Les règles pour positionner les labels et les flèches décrites ci-dessus continuent de s'appliquer, et vous pouvez positionner les nombres comme il vous plait en échangeant $y1$ and $y2$, ou $x1$ et $x2$. Par exemple, comparez

12. Mettre une première flèche dans les deux exemples précédents n'aura aucun effet car les flèches ne sont jamais tracées à l'origine.



avec ce que l'on obtient quand $x1$ and $x2$ sont échangés :



`\psaxes` place les graduations et les nombres sur les axes à intervalles réguliers, en utilisant les paramètres suivants :

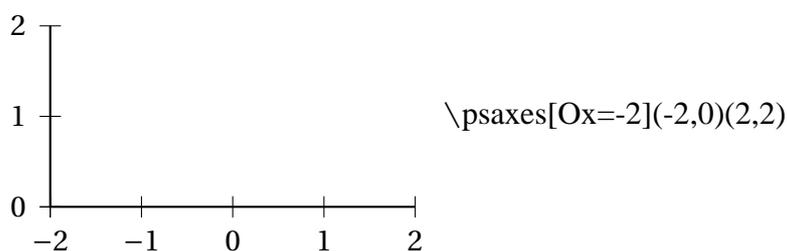
<i>Horizontal</i>	<i>Vertical</i>	<i>Défaut</i>	<i>Description</i>
Ox=num	Oy=num	0	Label à l'origine
Dx=num	Dy=num	1	incrément du label
dx=dim	oy=dim	Opt	Dist. entre les labels.

Quand **dx=dim** est à 0, `Dx\psxunit` est plutôt utilisé, et de même pour **dy**. Aussi, les valeurs par défaut de Opt pour **dx** et **dy** ne sont pas si bizarres que cela peut paraître.

Vous devez être très soigneux en fixant **Ox**, **Dx**, **Oy** et **Dy** à des valeurs non entières. `multido.tex` incrémente les labels en utilisant une arithmétique rudimentaire pour le nombre de décimales, et il se trompera à moins d'utiliser le même nombre de décimales pour **Ox** et **Dx**, ou **Oy** et **Dy**. La seule exception est que **Ox** ou **Oy** peuvent être des entiers, même si **Dx** ou **Dy** ne le sont pas. (L'inverse ne marche pas, cependant.)¹³

Noter que les premières coordonnées de `\psaxes` rentrées en argument déterminent la position physique de l'origine, mais elles n'affectent pas le label à l'origine. i.e., si l'origine est en (1, 1), l'origine est toujours notée 0 le long de chaque axe, à moins de changer de façon explicite **Ox** et **Oy**. Par exemple :

13. Par exemple **Ox=0** et **Dx=1.4** est bon, de même que **Ox=1** et **Dx=1.4**, mais **Ox=1.4** et **Dx=1**, ou **Ox=1.4** et **Dx=1.15**, n'est pas accepté. Si vous commettez cette erreur, PSTricks ne dira rien, mais vous n'aurez pas les bons labels.



Les graduations et les labels utilisent aussi quelques autres paramètres :

labels=*all/x/y/none* **Par défaut : all**

Pour spécifier où doivent apparaître les labels : sur les deux axes, l'axe des x , l'axe des y , ou nulle part.

showorigin=*true/false* **Par défaut : true**

Fixé à `true`, les labels sont placés à l'origine, aussi longtemps que les labels ne se finissent pas sur l'un des axes. Fixé à `false`, les labels ne sont jamais placés à l'origine.

ticks=*all/x/y/none* **Par défaut : all**

Pour spécifier si les graduations apparaissent : sur les deux axes, l'axe des x , l'axe des y , ou nulle part.

tickstyle=*full/top/bottom* **Par défaut : full**

Par exemple, si **tickstyle=top**, les graduations sont seulement du côté opposé à celui des labels. If **tickstyle=bottom**, les graduations sont du même côté que les labels. **full** donc des graduations s'étendant des deux côtés.

ticksize=*dim* **Par défaut : 3pt**

Les graduations ont la longueur *dim* au-dessus et/ou en-dessous de l'axe.

La distance entre les graduations et les labels is `\pslabelsep`, que l'on peut changer avec le paramètre **labelsep**.

Les labels sont écrits dans la fonte courante (l'un des exemples au-dessus était précédé de `\small` pour que les labels soient plus petits). Vous pouvez enjoliver les labels en redéfinissant les commandes :

`\pshlabel`

`\psvlabel`

i.e., si vous voulez changer la fonte des labels horizontaux, mais pas celle des labels verticaux, essayez quelque chose comme

`\def\pshabel#1{\small #1}`

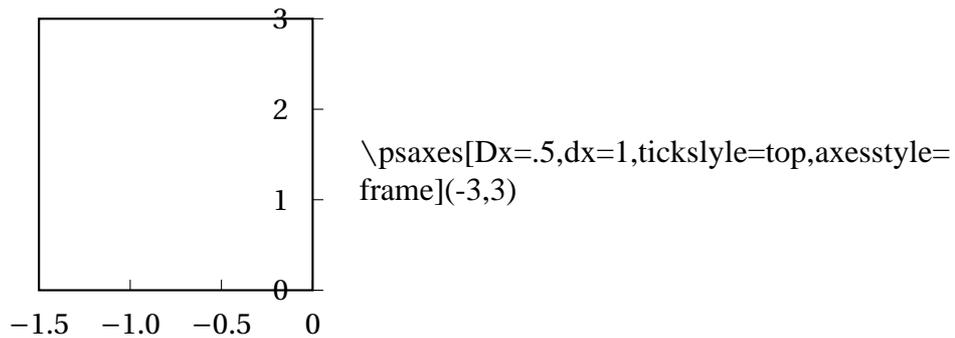
Vous pouvez choisir un cadre au lieu d'axes, ou pas d'axes (mais en

gardant les graduations et les labels), avec le paramètre :

axesstyle=axes/frame/none

Par défaut : axes

Les paramètres **linestyle**, **tillstyle** usuels et déjà mentionnés s'appliquent. Par exemple :



La macro `\psaxes` est assez souple, mais PSTricks contient quelques autres outils pour faire des axes à partir de rien. i.e., vous pouvez utiliser `\psline` et `\psframe` pour tracer les axes et le cadre, respectivement, `\multido` pour créer les labels (voir la documentation `multido.tex`), et `\multips` pour les graduations.

VI

Astuces pour le texte

27 Boîtes encadrées

Les macros pour encadrer les boîtes prennent leur argument, le mette dans une `\hbox`, et mettent un cadre Postscript autour. (Elles sont analogues aux `\fbox` de \LaTeX). Ainsi, ce sont des objets composites plutôt que de purs objets graphiques. En plus des paramètres graphiques pour `\psframe`, ces macros utilisent les paramètres suivants :

`\framesep=dim` **Par défaut : 3 pt**

Distance entre chaque côté du cadre et la boîte encadrée.

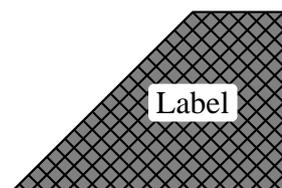
`\boxsep=true/false` **Par défaut : true**

Fixé à `true`, la boîte produite est de la taille du cadre ou de tout ce qui est dessiné autour de l'objet. Mis à `false`, la boîte produite est de la taille de tout ce qui est à l'intérieur, et ainsi le cadre est « transparent » à \TeX . Ce paramètre s'applique uniquement à `\psframebox`, `\pscirclebox`, et `\psovalbox`.

Voici les trois macros d'encadrement de boîtes :

`\psframebox*[par] {stuff}`

Un simple cadre (peut-être avec des coins arrondis) est tracé avec `\psframe`. l'option étoilée est d'un intérêt particulier. Elle engendre un cadre plein dont la couleur est définie par `fillcolor` (plutôt que `linecolor`, comme avec les objets graphiques fermés). Souvenez-vous que la valeur par défaut de `fillcolor` est `white`, et ceci a pour conséquence de cacher tout ce qui est derrière la boîte. Par exemple,



```
\pspolygon[fillcolor=gray,fillstyle=crosshatch*]
(0,0)(3,0)(3,2)(2,2)
\rput(2,1){\psframebox*[framearc=.3]{Label}}
```

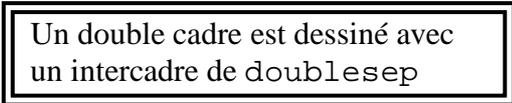
`\psdblframebox*[par] {truc}`

Ceci trace un double cadre. C'est juste une variante of `\psframebox`, défini par

```
\newpsobject{psdblframebox}{psframebox}{doublesep=\pslinewidth}
```

Par exemple,

```
\psdblframebox[linewidth=1.5pt]\parbox[c]{6cm}{\raggedright Un double
cadre est dessiné avec un intercadre de \texttt{doublesep}}
```



Un double cadre est dessiné avec
un intercadre de `doublesep`

```
\psshadowbox*[par]{truc}
```

Ceci dessine un simple cadre, avec une ombre.



```
\psshadowbox{Une grande idée !}
```

Vous pouvez obtenir l'ombre avec `\psframebox` juste en fixant le paramètre `shadowsize`, mais avec `\psframebox` les dimensions de la boîte ne renvoient pas l'ombre (c'est peut-être ce que vous voulez !).

```
\pscirclebox*[par]{truc}
```

Ceci dessine un cercle. Avec `boxsep=true`, la taille de la boîte est proche, mais peut être plus grande que celle du cercle. Par exemple :

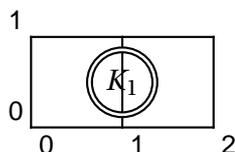


Vous êtes
là

```
\pscirclebox{\begin{tabular}{c} Vous êtes \\
là \end{tabular}}
```

```
\cput*[par]{angle (x, y)}{truc}
```

Ceci combine the fonctions de `\pscirclebox` et de `\rput`. C'est comme `\rput{<angle>}(x0, y0){\string\pscirclebox* [par]{<truc>}}`, mais il est plus efficace. Contrairement à la commande `\rput`, il n'y a pas d'argument pour changer le point de référence ; c'est toujours le centre de la boîte. Par contre, il y a un argument optionnel pour changer les paramètres graphiques . Par exemple



```
\cput[doubleline=true](1,.5){\large K_1}
```

```
\psovalbox*[par]{truc}
```

Ceci trace une ellipse. Si vous voulez un ovale avec des côtés droits et des coins arrondis, utilisez plutôt `\psframebox` avec une valeur positive pour **linearc** (cela dépend si **cornersize** est relatif ou absolu). Voici un exemple qui utilise **boxsep=false** :

Au prix de lancement
de \$13.99, cela rapporte
de se décider !

Au prix de lancement de
`\psovalbox[boxsep=false,linecolor=darkgray]`
{\$13.99}, cela rapporte de se déci-
der !

Vous pouvez définir des variantes de ces macros de cadres de boîtes en utilisant la commande `\newpsobject`.

Si vous voulez contrôler la taille finale du cadre, indépendamment du contenu nichez le `truc` dans quelque chose comme la commande `\makebox` de \LaTeX .

28 Coupures

La commande

`\clipbox[dim] {truc}`

met *truc* dans une `\hbox` et coupe ensuite tout autour la frontière de la boîte, à une distance *dim* de la boîte (la valeur par défaut est 0pt).

L'environnement `\pspicture` vous permet aussi de couper l'image de la frontière.

La commande

`\psclip{graphics} ... \endpsclip`

détermine le chemin de coupure vers le chemin dessiné par les objets graphiques, jusqu'à ce que la commande `\endpsclip` soit atteinte. `\psclip` and `\endpsclip` doivent être correctement insérés en respect des règles de \TeX . Seuls les graphiques purs (ceux décrits dans la partie II et `\pscustom`) sont autorisés. Un avertissement `Overfull \hbox` indique que l'argument *graphics* contient une sortie superflue ou un vide. Noter que les objets graphiques fonctionnent comme d'habitude, et que `\psclip` n'a pas d'autre conséquence sur le texte entouré. Voici un exemple :

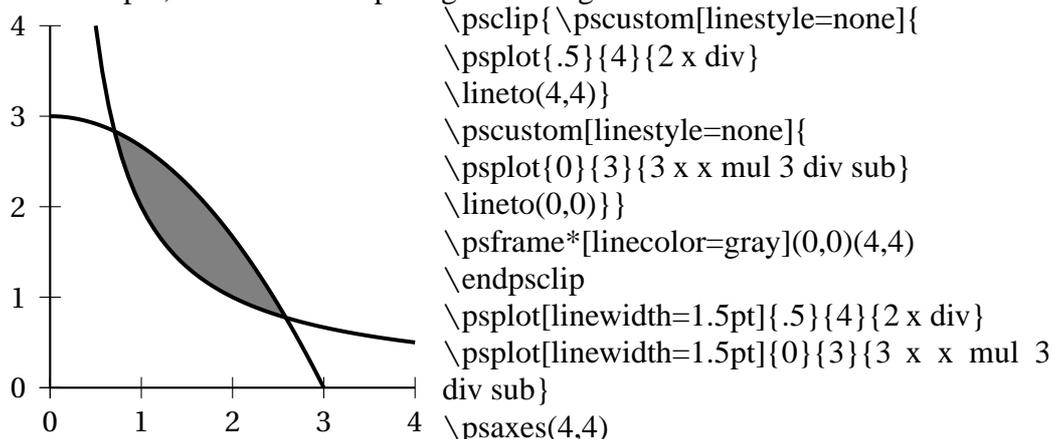
Coupures

« Une des meilleures nouvelles pièces que j'ai vu cette année : tranquille, poétique, ironique, » disait *Le Monde* ordinaire

```
\parbox{4.5cm}{
\psclip{\psccurve[linestyle=none]
(-3,-2)(0.3,-1.5)(2.3,-2)(4.3,-1.5)
(6.3,-2)(8,-1.5)(8,2)(-3,2)}
« Une des meilleures nouvelles pièces que j'ai
vu cette année : tranquille, poétique, ironique,
\ldots » disait \emph Le Monde sur l'extraor-
dinaire pièce sur un metteur en scène tchèque
et sur sa femme actrice, confrontant l'exil aus
États-Unis. \vspace{-1cm} \endpsclip}
```

Si vous ne voulez pas que le contour soit peint, vous devrez spécifier **linestyle=none** dans le paramètre. Vous pouvez en fait inclure plusieurs objets graphiques dans l'argument, et dans ce cas le chemin de coupure est fixé à l'intersection des chemins.

\psclip peut être un outil très utile dans les environnements d'image. Par exemple, ici il est utilisé pour griser la région entre deux courbes :



Notes pour le driver : les macros « clip » utilisent **\pstverbscale** et **\pstVerb**. Ne soyez pas surpris que les clips de PSTricks ne fonctionnent pas ou causent des problèmes (elles ne sont jamais robustes). **\endpsclip** utilise **\initclip**. Il peut interférer avec d'autres opérations « clip », spécialement si le document **T_EX** est converti en fichier .eps. La commande **\AltClipMode** entraîne que **\psclip** et **\endpsclip** utilisent **gsave** et **grestore** à leur place. Ceci pose problème à certains drivers, comme ceux de **NeXT_TTeX**, **TeXView**, spécialement si **\psclip** et **\endpsclip** ne finissent pas sur la même page.

29 Rotation et mise à l'échelle de boîtes

Il y a des versions des macros standard de rotation de boîte :

```
\rotateleft{truc}
```

`\rotateright{truc}`

`\rotatedown{truc}`

truc est placé dans une `\hbox` et ensuite tourné et mis à l'échelle, en prenant la place suffisante. Voici quelques exemples peu intéressants :

$\begin{array}{c} \text{Gauche} \\ \text{Bas} \end{array}$	$\begin{array}{c} \text{Droite} \\ \text{Bas} \end{array}$	$\{\Large\bf \quad \quad \quad \backslashrotateleft\{Gauche\}$ $\backslashrotatedown\{Bas\} \backslashrotateright\{Droite\}$
--	--	---

Il y a aussi deux macros de mise à l'échelle de boîtes :

`\scalebox{num1 num2}{truc}`

Si vous mettez deux nombres dans le premier argument, *num1* change l'échelle horizontalement et *num2* la change verticalement. Si vous ne mettez qu'un nombre, la boîte est mise à l'échelle de la même façon dans les deux dimensions. Vous ne pouvez pas mettre un facteur de zéro, mais les nombres négatifs sont autorisés, et ont pour effet de retourner la boîte autour des axes. on ne sait jamais, si vous voulez quelque chose comme -1 1 ceci (`\scalebox{-1 1}{ceci}`).

`\scaleboxto(x, y){truc}`

Cette fois-ci, le premier argument est une coordonnée (cartésienne), et la boîte est mise à l'échelle pour avoir la largeur *x* et la hauteur (plus la profondeur) *y*. Si l'une des dimensions est 0, la boîte est mise à l'échelle de la même façon dans les deux directions. Par exemple :

(4,2)Gros et long `\scaleboxto(4,2){Gros et long}`

PSTricks définit des environnements de boîtes LR pour toutes ces commandes de rotation et de mise à l'échelle de boîtes :

```
\pslongbox{Rotateleft}{\rotateleft}
\pslongbox{Rotateright}{\rotateright}
\pslongbox{Rotatedown}{\rotatedown}
\pslongbox{Scalebox}{\scalebox}
\pslongbox{Scaleboxto}{\scaleboxto}
```

Voici un exemple où nous `\Rotatedown` (retournons) les réponses aux exercices :

Question : Quelle était
la couleur du che-
val blanc d'Henri IV ?
... la question ?
Réponse : rélisez bien

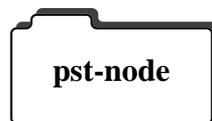
Question : Quelle était la cou-
leur du cheval blanc d'Henri
IV ? `\begin{Rotatedown}`
`\parbox{\hsize}{Réponse : re-`
`lisez bien la question \ldots`
`\hss}\end{Rotatedown}`

Lire la documentation de `fancybox.sty` pour les conseils sur la rotation d'un environnement `tableau` ou `figure`, et autres boîtes.

VII

Nœuds et leurs connexions

0.80.8



Toutes les commandes décrites dans cette partie sont contenues dans le fichier `pst-node/pst-node.sty`.

Les macros de nœuds et de connexion de nœuds vous permettent de connecter les informations et de placer les labels, sans savoir la position exacte de ce que vous voulez connecter ou d'où les lignes doivent être connectées. Ces macros sont utiles pour dessiner des graphes et des arbres, des diagrammes mathématiques, des diagrammes de syntaxe linguistique, et relier des idées de toutes sortes. Ce sont les trucs les plus astucieux dans PSTricks !

Bien que vous pensez pouvoir utiliser ces macros dans les dessins, en les plaçant et les faisant tourner avec `\rput`, vous pouvez déjà les utiliser n'importe où. Par exemple, je pourrais faire quelque chose comme ceci dans un guide sur le style des pages :

Avec le style de page `myfooters`, le nom de la section courante apparaît en bas de chaque page.

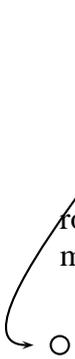
```

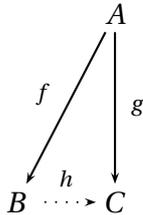
\makeatletter
\gdef\ps@temp{\def\@oddhead{ }\def
\@evenhead{ }
\def\@oddfoot{\small\sf
\ovalnode[boxsep=false]{A}{\rightmark}
\ncurve[ncurv=.5,angleB=240,angleA=180,
nodesep=6pt] {<-}{A}{B}
\hfil\thepage}
\let\@evenfoot\@oddfoot}
\makeatother \thispagestyle{temp}

```

Avec le style de page `myfooters`, le nom de la section courante apparaît en bas de chaque `\node{B}{page}`.

Vous pouvez utiliser les nœuds en mode mathématique et dans les environnements d'alignement aussi bien. Voici un exemple de diagramme commutatif :





```

\begin{array}{c@{\hspace 1
cm}c} & \rnode{a}{A}\[2cm]
\rnode{b}{B} & \rnode{c}{C}
\end{array} \psset{nodesep=3pt}
\everypsbox{\scriptstyle}
\ncline{->}{a}{b}\Bput{f}
\ncline{->}{a}{c}\Aput{g}
\ncline[linestyle=dotted]{->}{b}{c}\Aput{h}
\]

```

Les macros de n ?uds ont trois composants :

les définitions de n ?uds ; les définitions de n ?uds vous permette de leur assigner un nom et une forme. Voir la section 30.

les connexions de n ?uds ; les connexions de n ?uds connectent deux n ?uds identifiés par leur nom. Voir la section 31.

les labels de n ?uds ; les commandes de labels de n ?uds vous permette de donner un label à un n ?ud. Voir la section 32.

30 N ?uds

Le *nom* d'un n ?ud ne doit contenir que des lettres ou des chiffres, et doit commencer par une lettre.



Attention : Des noms de n ?uds incorrects peuvent engendrer des erreurs PostScript.

`\rnode[pointderéférence]{nom}{truc}`

Ceci assigne le *nom* au n ?ud, qui aura une forme rectangulaire afin de faire des connexions, avec le « centre » au point de référence (i.e., les connexions de n ?ud pointeront au point référence). `\rnode` a été utilisé dans les deux exemples ci-dessus.

`\Rnode(x, y){nom}{truc}`

Comme `\rnode`, mais le point de référence est calculé de façon différente. Il est fixé au milieu de la ligne de base, plus (x, y) . Si vous omettez l'argument (x, y) , la commande

`\RnodeRef`

est substituée. La définition par défaut de `\RnodeRef` est `0, 0.7ex`. par exemple sont équivalents :

```
\Rnode(0,.6ex){truc} et  
{\def\RnodeRef{0,.6ex}\Rnode{truc}}
```

`\Rnode` est utile quand les nœuds sont alignés sur la ligne de base, comme dans les diagrammes commutatifs. Avec `\rnode` les connexions de nœuds horizontaux ne seront pas tout à fait horizontaux, à cause de la différence de taille des lettres.

```
\pnode(x,y){nom}
```

Ceci crée un nœud de dimension zéro au point (x,y) (par défaut $(0,0)$)

```
\cnode*[par](x,y){rayon }{nom}
```

Ceci dessine un cercle et lui assigne l'appellation *nom*.

```
\circlenode*[par]{nom }{truc}
```

C'est une variante de `\pscirclebox` qui donne au nœud la forme d'un cercle.

```
\cnodeput*[par]{angle }{nom }{truc}
```

C'est une variante de `\cput` qui donne au nœud la forme d'un cercle.

```
\ovalnode*[par]{nom }{truc}
```

C'est une variante de `\psovalbox` qui donne au nœud la forme d'une ellipse.

La raison pour laquelle il n'y a pas de commande `\framenode` est, qu'en utilisant `\psframebox` (ou `\psshadowbox` ou `\psdblframebox`) dans l'argument de `\rnode` on obtient le résultat désiré.

31 Connexions de nœuds

Toutes les commandes de connexion de nœud commencent par `nc`, et elles ont toutes la même syntaxe :

```
\<nodeconnection> [ <para.> ] { <flèches> } { <n ?udA> } { <n ?udB> }
```

Une ligne de cette espèce est tracée du $n ?udA$ au $n ?udB$. Quelques-unes des commandes de connexion de nœuds sont un petit peu confuses, mais

Connexions de nœuds

avec un peu d'expérience vous arriverez à les comprendre et vous serez stupéfaits de tout ce qu'il est possible de faire.

Les connexions de points et de n ?uds peuvent être utilisées avec `\pscustom`. Le début de la connexion de n ?ud est attaché au point courant par un segment, comme avec `\psarc`.¹⁴

Quand on fait référence ci-dessous aux n ?uds A et B, nous nous référons seulement à l'ordre dans lequel les noms sont donnés en arguments dans les macros de connexion de n ?uds.

Quand le nom d'un n ?ud ne peut être trouvé sur la même page que la commande de connexion de n ?ud vous aurez soit pas de connexion ou une connexion à contresens. Malgré cela, \TeX ne signalera pas d'erreur.

Les connexions de n ?uds utilisent les paramètres suivants :

`nodesep=dimen.` Par défaut : 0

La bordure autour des n ?uds pour déterminer où connecter les lignes.

`offset=dimen.` Par défaut : 0

Après que le point de connexion soit déterminé, il est placé en haut pour le $n ?udA$ et en bas pour le $n ?udB$ d'une quantité *dimen.* assurant ainsi la connexion de la ligne à la droite du n ?ud.

`arm=dimen.` Par défaut : 10pt

Certaines connexions commencent par un segment de longueur *dimen.* avant de tourner dans une certaine direction.

`angle=angle` Par défaut : 0

Quelques connexions de n ?uds permettent de spécifier avec quel angle la connexion doit se faire avec le n ?ud.

`arcangle=angle` Par défaut : 8

Ceci s'applique seulement à `\ncarc` et est décrit plus bas.

`ncurv=num` Par défaut : .67

Ceci s'applique uniquement à `\nccurve` et à `\pccurve` et est décrit plus bas.

14. Regardez page 71 si vous voulez utiliser les n ?uds comme coordonnées dans d'autres macros de PSTriks.

loopsize=dimen.

Par défaut : 1cm

S'applique uniquement à `\ncloop` et à `\pccloop` décrits plus bas.

Vous pouvez fixer ces paramètres séparément pour les deux n ?uds. Ajouter just un A ou un B au nom du paramètre. i.e.

```
\psset{nodesepA=3pt, offsetA=5pt, offsetB=3pt, arm=1 cm}
```

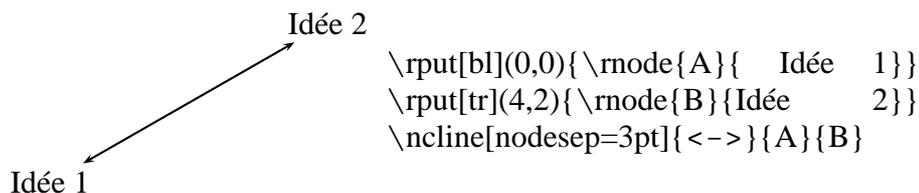
fixe **nodesep** pour le n ?ud A , mais laisse la valeur pour le n ?ud B inchangée, fixe **offset** pour les n ?uds A et B à différentes valeurs , et fixe **arm** pour les n ?uds A et B à la même valeur.

N'oubliez pas qu'en utilisant le paramètre **border**, vous pouvez créer l'impression qu'une connexion de n ?ud passe au-dessus d'une autre.

Voici une description de la commande de connexion de n ?ud individuelle :

```
\ncline*[par]{arrows} {n ?udA}{n ?udB}
```

Ceci trace une ligne droite entre les n ?uds. Seuls les paramètres **offset** et **nodesep** sont utilisés.

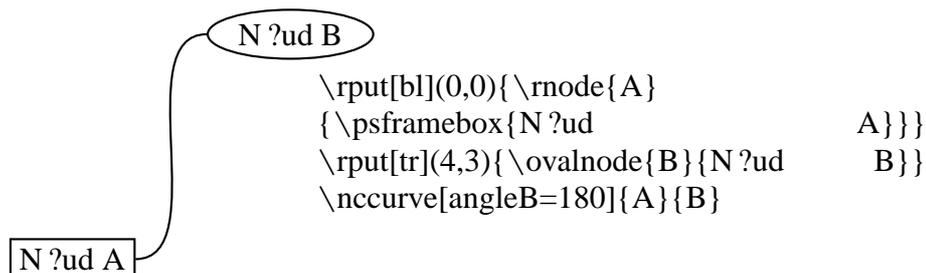


```
\ncLine*[par]{arrows} {n ?udA}{n ?udB}
```

Comme `\ncline`, mais les labels (avec `\lput`, etc.) sont placés comme si la ligne partait et finissait aux centres des n ?uds. ceci est commode si vous avez plusieurs lignes parallèles et que vous voulez aligner les labels, même si les n ?uds sont de tailles différentes, par exempl, dans les diagrammes commutatifs.

```
\nccurve*[par]{arrows} {n ?udA}{n ?udB}
```

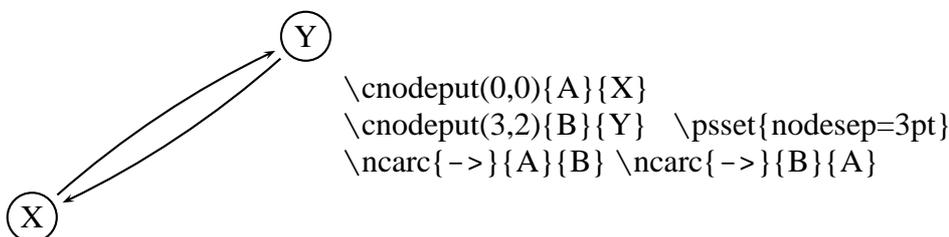
trace une courbe de Bezier entre les n ?uds. Il utilise les paramètres **nodesep**, **offset**, **angle** et **ncurv**.



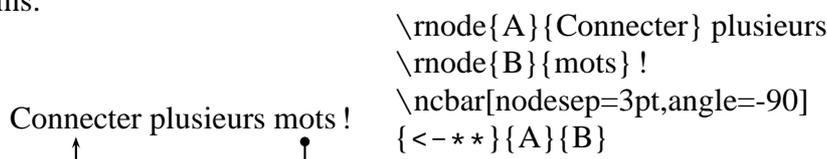
Connexions de n ?uds

`\ncarc*[par]{arrows} {n ?udA}{n ?udB}`

En fait une variante de `\nccurve`. i.e., qu'il connecte les n ?uds avec une courbe de Bézier, en utilisant les paramètres **nodesep**, **offset** et **ncurv**. Cependant, la courbe connecte le n ?ud A avec un angle **arcangleA** avec la ligne droite entre A et B, et connecte au n ?ud B avec un angle **arcangleB** avec la ligne droite entre B et A. Des valeurs égales pour **arcangleA** et **arcangleB** (i.e., 8 la valeur par défaut) et avec la valeur par défaut de **ncurv**, la courbe se rapproche d'un arc de cercle. `\ncarc` est un bon moyen de connecter deux n ?uds avec deux lignes.

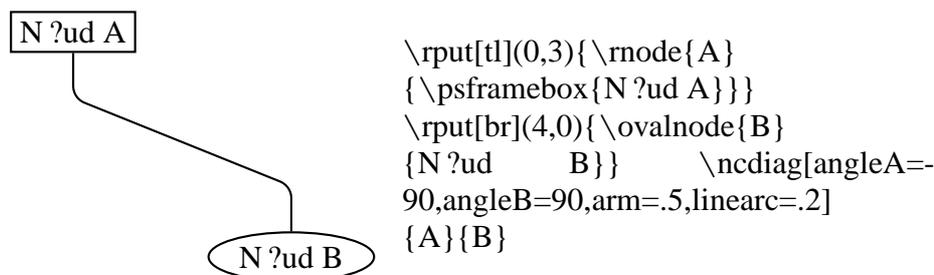


D'abord, les lignes sont tracées rattachées aux deux n ?uds en faisant un angle **angleA** et de longueurs **armA** and **armB**. Ensuite l'une des branches est allongée de sorte que les deux soient connectées, la ligne finale contient trois segments se rencontrant aux bons angles. En général la ligne entière a trois segments rectilignes. La valeur de **linearc** est utilisée pour arrondir les coins.



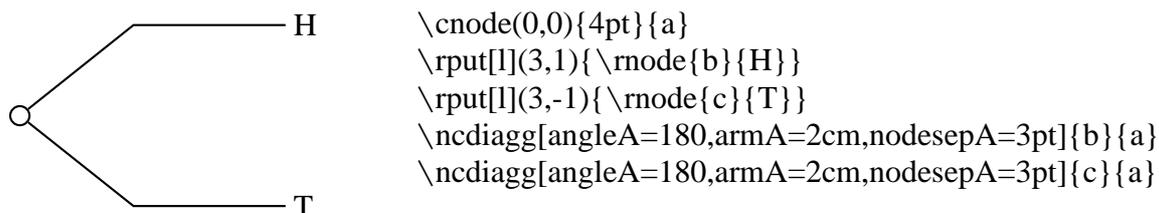
`\ncdiag*[par]{arrows} {n ?udA}{n ?udB}`

D'abord, les branches sont tracées en utilisant **angle** et **arm**. Ensuite elles sont connectées avec une ligne droite. En général, la ligne entière se compose de trois segments. La valeur de **linearc** est utilisée pour arrondir les coins.

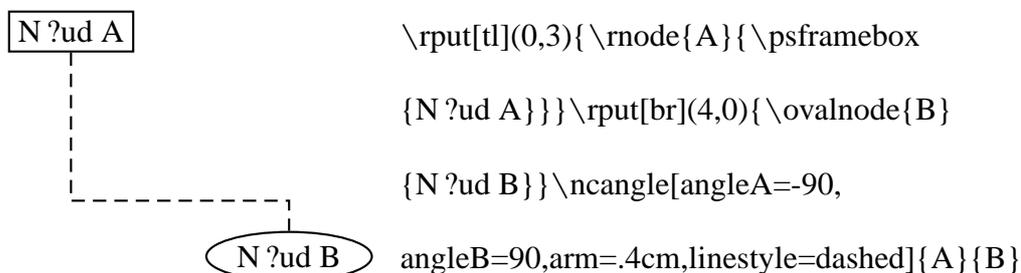


$\backslash\ncdiagg^*[par]{arrows} \{n ?udA\} \{n ?udB\}$

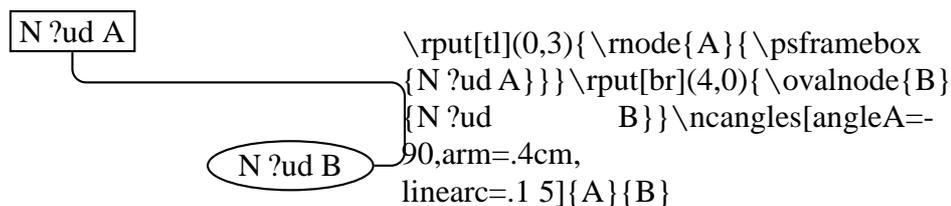
Semblable à $\backslash\ncdiag$, mais seule la branche pour le $n ?ud A$ is dessinée. l'extrémité de cette branche est ensuite connectée directement au $n ?ud B$. Typiquement la connexion a deux segments. La valeur de **linearc** est utilisée pour arrondir les coins.


 $\backslash\ncangle^*[par]{arrows} \{n ?udA\} \{n ?udB\}$

Les points de connexion de $n ?uds$ sont déterminés par **angleA** et **angleB** (et **nodesep** et **offset**). Puis une branche est dessinée pour le $n ?ud B$ en se servant de **armB**. Cette branche est connectée au $n ?ud A$ par un angle correct, qui rencontre aussi le $n ?ud$ avec un angle **angleA**. En général, le tracé entier se compose de trois segments, mais il peut y en avoir moins. La valeur de **linearc** est utilisée pour arrondir les coins. Simple, non ? Voici un exemple :

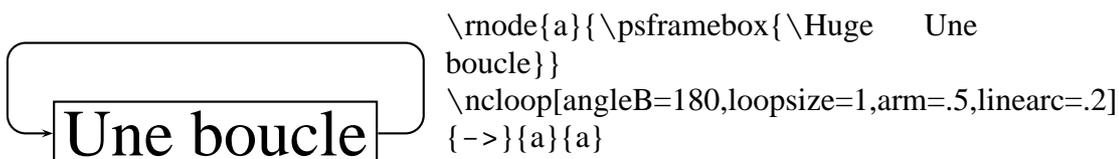

 $\backslash\ncangles^*[par]{arrows} \{n ?udA\} \{n ?udB\}$

Semblable à $\backslash\ncangle^*$, mais **armA** et **armB** sont utilisés. Les branches sont connectées avec un bon angle qui rencontre **armA** avec un angle aussi bon qu'il peut l'être. Généralement il y a quatre segments (d'où plus d'un angle que $\backslash\ncangle$, d'où le s dans $\backslash\ncangles$). La valeur de **linearc** est utilisée pour arrondir les coins. Comparer cet exemple avec le précédent :



`\ncloop*[par]{arrows} {n ?udA}{n ?udB}`

Le premier segment est **armA**, ensuite il tourne de 90 degrés vers la gauche, dessinant un segment de longueur **loopsize**. Le segment suivant est encore à angle droit ; il se connecte à **armB**. Par exemple :



`\ncircle*[par]{arrows} {n ?udA}{radius}`

Ceci dessine un cercle d'un noeud vers lui-même. C'est la seule commande de connexion de n ?ud de cette sorte. Le cercle débute avec un angle **angleA** et tourne autour du n ?ud dans le sens des aiguilles d'une montre, à la distance **nodesepA** du n ?ud. Les commandes de connexion de n ?uds sont des outils de tracés très intéressants, donnant une solution alternative à **\psline** pour connecter deux points. Il y a des variantes de commandes de connexion de n ?uds à cet effet. Chacune commence par **pc** (pour « point connexion ») à la place de **nc**. i.e.,

`\pcarc{<->}(3,4)(6,9)`
 donne le même résultat que
`\pnode(3,4){A}\pnode(6,9){B}\pcarc<->}{A}{B}`
 Seuls **\ncLine** et **\nccircle** n'ont pas de variantes **pc** :

`\pcline*[par]{arrows} (x1, y1)(x2, y2)`

Comme **\ncline**.

`\pccurve*[par] {arrows}(x1, y1)(x2, y2)`

Comme **\nccurve**.

`\pcarc*[par] {arrows}(x1, y1)(x2, y2)`

Comme **\ncarc**.

`\pcbar*[par] {arrows}(x1, y1)(x2, y2)`

Comme **\ncbar**.

`\pcdiag*[par] {arrows}(x1, y1)(x2, y2)`

Comme `\ncdiag`.

`\pcangle*[par] {arrows}(x1, y1)(x2, y2)`

Comme `\ncangle`.

`\pccloop*[par] {arrows}(x1, y1)(x2, y2)`

Comme `\ncloop`.

32 Relier des labels à des connexions de n ?uds

Nous en arrivons maintenant aux commandes pour rattacher des labels aux connexions de n ?uds. La commande de label de n ?ud doit venir juste après la connexion de n ?ud à laquelle elle doit être attachée. Vous pouvez rattacher plus d'un label, et un label peut contenir plusieurs n ?uds.

Les commandes de label de n ?ud doivent se terminer sur la même page \LaTeX que la connexion à laquelle le label correspond.

L'argument de coordonnées dans les autres commandes PSTricks est un simple nombre dans les commandes de labels de n ?uds : (*pos*). Ce nombre choisit un point sur la connexion de n ?ud, suivant à peu près le schéma suivant : chaque connexion de n ?ud a une potentialité de un ou plusieurs segments, y compris les branches et les connecteurs de ligne. Un nombre *pos* entre 0 et 1 choisit un point sur le premier segment du n ?ud A vers B, (une fraction *pos* du début à la fin du segment), un nombre entre 1 et 2 choisit un point sur le deuxième segment, et ainsi de suite. Chaque connexion de n ?ud a sa propre valeur de coordonnées par défaut, qui est utilisée par quelques versions courtes de commandes de label.

Voici les détails pour chaque connexion de n ?ud :

<i>Connexion</i>	<i>Segments</i>	<i>variations</i>	<i>Par défaut</i>
<code>\ncline</code>	1	$0 \leq pos \leq 1$	0.5
<code>\nccurve</code>	1	$0 \leq pos \leq 1$	0.5
<code>\ncarc</code>	1	$0 \leq pos \leq 1$	0.5
<code>\ncbar</code>	3	$0 \leq pos \leq 3$	1.5
<code>\ncdiag</code>	3	$0 \leq pos \leq 3$	1.5
<code>\ncdiagg</code>	2	$0 \leq pos \leq 2$	0.5
<code>\ncangle</code>	3	$0 \leq pos \leq 3$	1.5
<code>\ncloop</code>	5	$0 \leq pos \leq 4$	2.5
<code>\nccircle</code>	1	$0 \leq pos \leq 1$	0.5

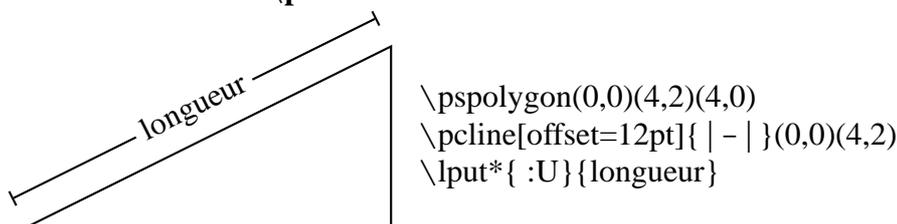
Il y a une autre différence entre les commandes de label de n ?ud et les autres commandes `\put`. En plus des différentes façons de spécifier l'angle de la rotation pour `\rput`, avec les commandes de label de n ?ud, l'angle

peut être de la forme `{ :angle}`. Dans ce cas, l'angle est calculé après rotation du système de coordonnées de telle sorte que la connexion du n ?ud à la position du label indique la droite (du n ?ud A au n ?ud B). i.e. si l'angle est `{ :U}`, alors le label court parallèlement à la connexion de n ?ud.

Voici les commandes de label de n ?ud :

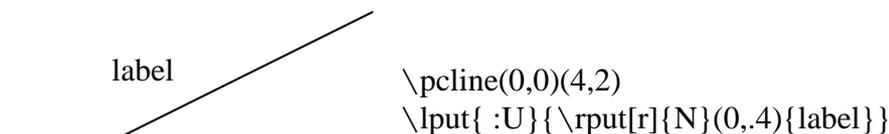
```
\lput*[refpoint]{rotation}(pos){truc}
```

Le `|` se présente pour le « label ». Voici un exemple illustrant l'utilisation de l'étoile optionnelle et de `:angle` avec `\lput`, ainsi que l'utilisation du paramètre `offset` avec `\pcline` :



(Souvenez-vous qu'avec les commandes `put`, vous pouvez omettre les coordonnées si vous mettez l'angle de la rotation. il est probable que vous utiliserez cette particularité avec les commandes de label de n ?ud.)

Avec `\lput` et `\rput`, il y a plusieurs contrôles sur la position du label. i.e.,

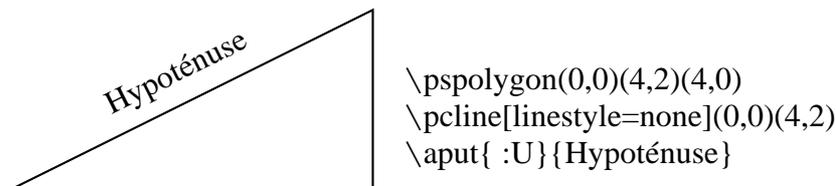


met le label en haut et à droite sur la page, avec le côté droit placé à 0,4 centimètres au-dessus de la position 0,5 de la connexion de n ?ud (au-dessus si la connexion de n ?ud indique la droite). Cependant, les commandes `\aput` et `\bput` décrites plus bas gèrent les cas les plus courants sans `\rput`.¹⁵

15. Il y a aussi une commande obsolète `Lput` pour placer les labels près des connexions de n ?ud. La syntaxe est : `Lput{<labelsep>}[<pointderéférence>]{<rotation>}<pos>{<truc>}` C'est un mélange de `Rput` et de `lput`, équivalent à `lput<pos>{Rput{<labelsep>}[<pointderéférence>]{<rotation>}(0,0){<truc>}`. `Mput` est une version courte de `Lput` sans les arguments `{<rotation>}` ou `(pos)`. `Lput` et `Mput` font encore partie de PSTricks pour cause de compatibilité descendante.

`\aput*[labelsep]{angle}(pos){truc}`

truc est placé à la distance `\pslabelsep` au-dessus de la connexion de `n ?ud`, avec la convention que les connexions de `n ?ud` vont vers la droite. `\aput` est une variante de connexion de `n ?ud` de `\uput`. Par exemple :



`\bput*[labelsep]{angle}(pos){truc}`

Comme `\aput`, mais *truc* est placé sous la connexion de `n ?ud`.

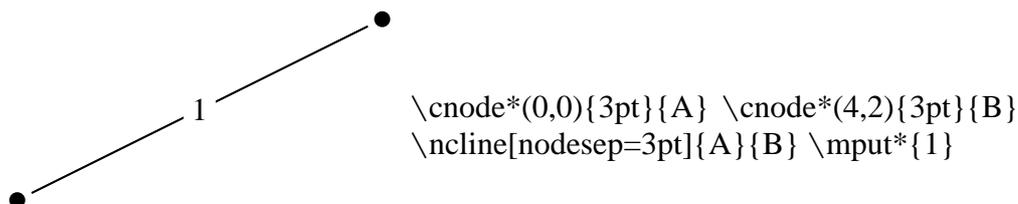
On voudrait en général utiliser les valeurs par défaut de la position et de la rotation, mais vous devez préciser au moins l'un des arguments. Aussi PSTricks possède quelques variantes :

`\mput*[pointderéférence]{truc}`

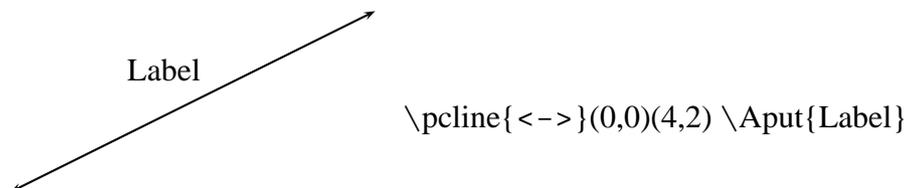
`\Aput*[labelsep]{truc}`

`\Bput*[labelsep]{truc}`

respectivement de `\lput`, `\aput` et `\bput`, qui n'ont pas d'arguments d'angle ou de positionnement. Par exemple :



Un autre :



Maintenant nous pouvons comparer `\ncline` avec `\ncLine`, et `\rnode` avec `\Rnode`. D'abord, voici un diagramme mathématique, avec `\ncLine` et `\Rnode` :

Relier des labels à des connexions de `n ?uds`

```

\l
\setlength{\arraycolsep}{1cm}
\def\tX{\tilde{\tilde{X}}}
\begin{array}{cc} \Rnode{a}{(X-A,N-A)} & \Rnode{b}{(\tX,a)} \\ \Rnode{c}{(X,N)} & \Rnode{d}{\LARGE(\tX,N)} \end{array}
\psset{nodesep=5pt,arrows=->}
\everypsbox{\scriptstyle}
\ncLine{a}{b}\Aput{a}
\ncLine{a}{c}\Bput{r}
\ncLine[linestyle=dashed]{c}{d}\Bput{b}
\ncLine{b}{d}\Bput{s}
\l

```

$$\begin{array}{ccc}
 (X-A, N-A) & \xrightarrow{a} & (\tilde{\tilde{X}}, a) \\
 \downarrow r & & \downarrow s \\
 (X, N) & \xrightarrow{\text{---} b \text{---}} & (\tilde{\tilde{X}}, N)
 \end{array}$$

Voici le même, mais avec `\ncline` et `\rnode`

$$\begin{array}{ccc}
 (X-A, N-A) & \xrightarrow{a} & (\tilde{\tilde{X}}, a) \\
 \downarrow r & & \downarrow s \\
 (X, N) & \xrightarrow{\text{---} b \text{---}} & (\tilde{\tilde{X}}, N)
 \end{array}$$

Notes pour le driver : les macros de n ?uds utilisent `\pstVerb` et `\pstverbscale`.



Astuces spéciales

33 Spires et zigzags

Le fichier `pst-coil.tex/pst-coil.sty` (et de façon optionnelle le fichier d'en-tête `pst-coil.pro`) définissent les objets graphiques suivants pour les spires et les zigzags :

`\pscoil*[par]{arrows}(x0, y0) (x1, y1)`

`\psCoil*[par]{angle1angle2}`

`\pszigzag*[par]{arrows}(x0,y0) (x1,y1)`

Ces objets graphiques utilisent the paramètres suivants :

<code>\coilwidth=dim</code>	Par défaut : 1 cm
<code>\coilheight=num</code>	Par défaut : 1
<code>\coilarm=dim</code>	Par défaut : 0.5 cm
<code>\coilaspect=angle</code>	Par défaut : 45
<code>\coilinc=angle</code>	Par défaut : 10

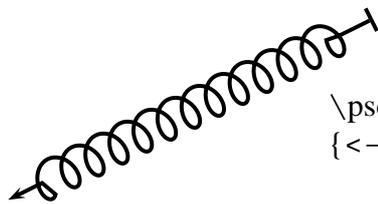
Les spires ou zigzag sont tracés avec une largeur (diamètre) de **coilwidth**, et avec la distance le long des axes pour chaque période (360 degrés) égale à

$$\text{coilheight} \times \text{coilwidth}.$$

`\pscoil` et `\psCoil` traçent une spire en « 3D », projetée sur les axes Ox-Oz. le centre de la spire 3D est sur le plan Oyz à un angle `pcoilaspect` de l'axe Oz . La spire est tracée avec des instructions PostScript `lineto`, joignant les points qui sont à l'angle **coilinc** l'un de l'autre le long de la spire . Aussi, augmenter **coilinc** rend la courbe plus douce mais ralentit l'impression. `\pszigzag` n'utilise pas les paramètres **coilaspect** et **coilinc**.

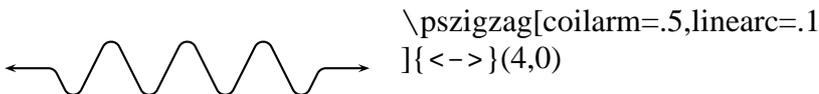
`\pscoil` et `\pszigzag` connectent $(x0,y0)$ et $(x1,y1)$, en partant et en finissant avec des segments de longueur **coilarmA** et **coilarmB**, respectivement. Fixer **coilarm** revient à fixer **coilarmA** et **coilarmB**.

Voici un exemple of `\pscoil` :



```
\pscoil[coilarm=.5cm,linewidth=1.5pt,coilwidth=.5cm]
{<-|}(4,2)
```

Voici un exemple of `\pszigzag` :



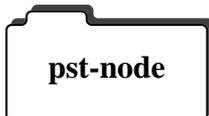
```
\pszigzag[coilarm=.5,linearc=.1]
{<->}(4,0)
```

Remarquez que `\pszigzag` utilise les paramètres **linearc**, et que les segments de départ peuvent être plus longs que **coilarm** pour tendre la spire.

`\psCoil` dessine seulement la spire horizontalement de l'*angle1* jusqu'à l'*angle2*. Utiliser `\rput` pour tourner ou translater la spire, si vous le désirez. `\psCoil` n'utilise pas le paramètre **coilarm**. Par exemple, avec **coilaspect=0** on obtient une courbe sinusoïdale :



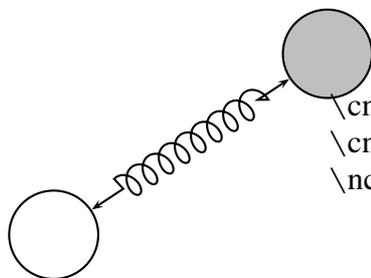
```
\psCoil[coilaspect=0,coilheight=1.33,
coilwidth=.75,linewidth=1.5pt]{0}{1440}
```



`pst-coil.tex` contient aussi des connexions de nœuds de spires et de zigzags. Vous devez aussi charger `pst-node.tex/pst-node.sty` pour vous en servir. Les connexions de nœuds sont :

```
\nccoil*[par]{arrows} {n ?udA}{n ?udB}
\nczigzag*[par]{arrows} {n ?udA}{n ?udB}
\pccoil*[par]{arrows} (x1, y1)(x2, y2)
\pczigzag*[par]{arrows} (x1, y1)(x2, y2)
```

Les points finaux sont choisis de la même façon qu'avec `\ncline` et `\pcline`, sinon ces commandes fonctionnent comme `\pscoil` et `\pszigzag`. Par exemple :



```
\cnode(.5,.5){.5}{A}
\cnode[fillstyle=solid,fillcolor=lightgray](3.5,2.5){.5}{B}
\nccoil[coilwidth=.3]{<->}{A}{B}
```

34 Coordonnées spéciales

La commande

\SpecialCoor

a une caractéristique particulière vous permettant de spécifier les coordonnées dans un grand nombre de variétés, en sus des coordonnées cartésiennes habituelles.¹⁶ la compilation est un peu plus lente et cette commande moins robuste, c'est la raison pour laquelle cette possibilité est utilisée à la demande plutôt que par défaut, mais vous ne remarquerez probablement pas la différence.

Voici les coordonnées possibles :

(x,y) Les coordonnées cartésiennes usuelles . i.e., (3,4).

(r ; a) Coordonnées polaires, avec un rayon r et un angle a . L'unité par défaut pour r est **unit**. i.e., (3 ; 110).

(n ?ud) le centre du n ?ud. i.e., (A).

([par]node) La position relative au n ?ud déterminée en utilisant les paramètres **angle**, **nodesep** et **offset** . i.e., ([angle=45]A).

(!ps) Code PostScript brut . *ps* doit s'étendre à une paire de coordonnées . Les unités **xunit** et **yunit** sont utilisées. Par exemple, si vous voulez utiliser des coordonnées polaires (3 ; 110) qui sont mises à l'échelle avec **xunit** et **yunit**, vous pouvez écrire

(!3 110 cos mul 3 110 sin mul)

(coor1/coor2) La coordonnée x pour *coor1* et la coordonnée y pour *coor2*. *coor1* et *coor2* peuvent être n'importe quelle coordonnée utilisée avec **\SpecialCoor**. Par exemple, (A | 1in ; 30).

\SpecialCoor vous permet aussi de spécifier des angles de plusieurs façons :

num Un nombre, comme d'habitude, avec des unités données par la com-

16. Il y a une commande obsolète **Polar** qui produit des coordonnées de la forme (r, a) interprétées comme des coordonnées polaires. L'utilisation de **Polar** est déconseillée car primo, elle ne permet pas de mélanger coordonnées cartésiennes et polaires comme le fait **SpecialCoor**, secundo car il n'est pas très clair quand on regarde un fichier par exemple en lisant (3,2), de savoir si ce sont des coordonnées cartésiennes ou polaires). La commande pour annuler **Polar** est **Cartesian**. Il y a un argument optionnel pour fixer les unités par défaut. i.e.,

Cartesian(< x >, < y >)

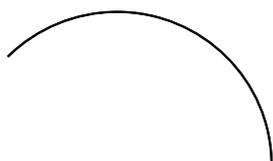
a l'effet de

psset{xunit=< x >,yunit=< y >}

Cartesian peut être utilisé à cet effet sans utiliser **Polar**.

mande `\degrees`.

(*coor*) Une coordonnée, indiquant vers où l'angle est dirigé. Vérifier bien d'inclure les `()`, en plus de toute autre chose fixant l'angle que l'argument utilisé. Par exemple, ce qui suit montre deux façons de tracer un arc de 0.8 inch de rayon de 0 à 135 degrés :



```
\SpecialCoor
\psarc(0,0){.8in}{0}{135}
\psarc(0,0){.8in}{0}{(-1,1)}
```

!ps Code PostScript brut. *ps* doit contenir un nombre. Les unités utilisés sont les mêmes que celles de *num*.

La commande

`\NormalCoor`

annule l'état `\SpecialCoor`.

35 Recouvrements (Overlays)

Les recouvrements sont surtout intéressants pour faire des diapos et les macros de recouvrement décrits dans cette section sont surtout intéressantes pour les constructeurs de macro \TeX qui veulent implanter des recouvrements dans un « package » de macros de diapos. Par exemple, *seminar.sty*, un style \LaTeX pour les notes et les diapos, se sert de *PSTricks* pour implémenter des recouvrements.

Les recouvrements sont construits en créant une `\hbox` et en sortant plusieurs fois la boîte, imprimant différents contenus dans la boîte à chaque fois. La boîte est créée avec les commandes

```
\overlaybox truc \endoverlaybox
```

Les utilisateurs de \LaTeX peuvent écrire :

```
\begin{overlaybox} <truc> \end{overlaybox}
```

Le contenu pour le recouvrement *string* doit se placer dans le champ de la commande

```
\psoverlay{string}
```

chaîne peut être n'importe quelle chaîne après développement. Tout ce qui n'est pas dans le champ de toute commande `\psoverlay` va sur le recouvrement principal et tout ce qui est dans le champ de `\psoverlay{all}` va sur tous les recouvrements. Les commandes `\psoverlay` peuvent être imbriquées et être utilisées en mode mathématiques.

La commande

`\putoverlaybox{chaîne}`

affiche ensuite le recouvrement *chaîne*.

Voici un exemple :

`\overlaybox`

`\psoverlay{all}`

`\psframebox[framearc=.15,linewidth=1.5pt]{`

`\psoverlay{main}`

`\parbox{3.5cm}{\raggedright`

Les flacons de mousse portent préjudice à l'environnement {`\psoverlay{one}`
moins que les dommages du papier,} dit une étude.}} `\endoverlaybox`

`\putoverlaybox{main} \hspace{.5in} \putoverlaybox{one}`

Les flacons de
mousse portent
préjudice à
l'environnement

dit une
étude.

moins que les
dommages du
papier,

Notes pour le driver : les recouvrements utilisent `\pstVerb` and `\pstverbscale`.

36 Le style de remplissage gradient



Le fichier `gradient.tex/gradient.sty`, avec le fichier d'en-tête PostScript `gradient.pro`, définit le **fillstyle** `gradient`, pour un ombrage gradué. Ce style de remplissage utilise les paramètres suivants :

gradbegin=couleur

Par défaut : gradbegin

La couleur de début et de fin.

Coordonnées spéciales

gradend=couleur**Par défaut : gradend**

La couleur au point milieu.

gradlines=ent.**Par défaut : 500**

Le nombre de lignes. Plus de lignes signifie une gradation plus fine, mais une impression plus lente.

gradmidpoint=num**Par défaut : .9**La position du point milieu, en fraction de la distance du haut au bas. *num* doit être entre 0 and 1.**gradangle=angle****Par défaut : 0**L'image subit une rotation de *angle*.

gradbegin et **gradend** seront de préférence des couleurs `rgb`, mais les gris et les couleurs `cmyk` marcheront aussi. Les définitions des couleurs `gradbegin` et `gradend` sont :

```
\newrgbcolor{gradbegin}{0 .1 .95}
\newrgbcolor{gradend}{0 1 1 }
```

Voici deux façons de changer les couleurs de gradient :

```
\newrgbcolor{gradbegin}{1 .4 0}
```

et

```
\psset{gradbegin=blue}
```

Essayer cet exemple :

```
\psframe[fillstyle=gradient,gradangle=45](10,-20)
```

37 Ajouter de la couleur aux tableaux



Le fichier `colortab.tex/colortab.sty` contient des macros qui, utilisées avec les commandes de couleurs comme celles de `PSTricks`, vous permette de colorier les cellules et les lignes dans les tableaux. Regarder `colortab.doc` pour plus de détails.

38 Texte courant le long d'un chemin

Texte courant le long d'un chemin



Le fichier `textpath.tex/textpath.sty` définit la commande `\pstextpath`, pour taper un texte le long d'un chemin. C'est très puissant, mais il y a quelques limitations :

- `textpath.tex` fonctionne seulement avec certains drivers DVI-to-PS. Voici ce qui est connu :
 - Il fonctionne avec le `dvips` de Rokicki, version 5.487 ou ultérieure (au moins jusqu'à v5.495).
 - Il ne fonctionne pas avec des versions plus récentes de `dvips`.
 - Il ne fonctionne pas avec `TeXview` (pour avoir un aperçu des fichiers avec `NeXT-TeX 3.0`, convertissez le fichier `.dvi` en un fichier PostScript avec `dvips-o` et utilisez `Preview`).
 - « ne marche pas » signifie que cela n'a pas d'effet pour le meilleur ou le pire.
 - Il peut fonctionner avec d'autres drivers. L'exigence est que le driver utilise seulement des opérateurs de présentation PostScript, ni chargé, ni lié pour montrer des caractères.
- Vous devez aussi avoir installé le fichier d'en-tête PostScript `textpath.ps`, et `\pstheader` doit être correctement défini dans `pstricks.con` pour votre driver.
- Comme d'autres macros PSTricks qui mettent en jeu des rotations de texte, elle fonctionne mieux avec des fontes PostScript.
- L'interprétation PostScript est lente avec `textpath.tex`.

À cause de tout ceci, aucun exemple n'est donné ici. Cependant, il y a un fichier `test tp-test.tex` et un fichier de sortie PostScript `tp-test.ps` qui est distribué avec PSTricks.

Voici la commande :

`\pstextpath[pos](x,y) {objet graphique}{texte}`

texte est placé le long du chemin, du départ vers la fin, défini par l'objet graphique PSTricks. (Cet objet autrement se comporte normalement. Fixer `linestyle=none` si vous ne voulez pas le faire apparaître.)

texte ne doit contenir que des caractères. Pas de règles \TeX , pas de PSTricks, et rien d'autre de spécial. (Ces choses ne sont pas source d'erreurs ; elles ne fonctionnent pas bien.) On peut écrire en mode mathématique, mais les opérateurs mathématiques qui sont construits avec plusieurs caractères (par

Coordonnées spéciales

exemple, des grands signe d'intégrales) peuvent être rompus. Des boîtes entières (par exemple, `\parbox`) sont possibles aussi, mais ceci est juste bon pour le divertissement.

pos est soit :

l justifié à gauche

c centré sur le chemin

r justifié à droite.

Le choix par défaut est l .

(x,y) est un écart. Les caractères sont placés à la distance x le long du chemin, et sont remontés de y . « remontés » signifie relativement au chemin, à tout point sur le sentier correspondant au milieu du caractère. (x,y) doivent être des coordonnées cartésiennes . Les deux coordonnées utilisent `\psunit` par défaut. Les coordonnées par défaut sont $(0, \text{\TPoffset})$, où `-.7ex` est une commande dont la valeur par défaut est `-.7ex`. Cette valeur donne un bon espacement aux caractères. Souvenez-vous que les unités ex sont pour les fontes en usage quand `\pstextpath` agit, et non à l'intérieur de l'argument *texte*.

D'autres choses que vous devez savoir :

- Comme avec `\rput` et les objets graphiques, c'est à vous de laisser de l'espace pour `\pstextpath`.
- Les résultats sont imprévisibles si le texte est plus long que le chemin.
- `\pstextpath` laisse l'écriture du texte à \TeX . Il capte juste l'opérateur `show` pour retracer le système de coordonnées.

39 Rayure et remplissage de chemins de caractères



Le fichier `charpath.tex/charpath.sty` définit la commande :

`\pscharpath*[par]{texte}`

Il raye et remplit les chemins de caractères en se servant de `linestyle` et de `fillstyle` de PSTricks.

Les restrictions sur les drivers DVI-to-PS précédemment recensées pour `\pstextpath` s'appliquent à `\pscharpath`. De plus, seules les fontes de sortie PostScript sont affectées.

Les fichiers d'exemple `chartest.tex` et `chartest.ps` sont distribués avec PSTricks.

Avec l'option `*`, le chemin de caractères n'est pas supprimé de l'envi-

ronnement PostScript à la fin. C'est surtout pour des coupures spéciales. Par exemple, vous pouvez utiliser `\pscharpath*` dans le premier argument de `\pstextpath`, et ensuite écrire du texte le long du chemin. Regardez le fichier d'exemple `denis1.tex`. (Malgré tout, vous ne pouvez pas combiner `\pscharpath` et `\pstextpath` de toute autre façon. Ainsi, vous ne pouvez pas taper des contours de caractères le long d'un chemin, et les remplir ou les hachurer ensuite avec `\pscharpath`.)

La commande

`\pscharclip*[par]{text} endpscharclip`

fonctionne comme `\pscharpath`, mais il fixe le chemin de coupure au chemin de caractères. Vous souhaitez peut-être fixer la position de ce chemin de coupure en utilisant `\rput` à l'intérieur de l'argument de `\pscharclip`. Comme `\psclip` et `\endpsclip`, `\pscharclip` et `\endpscharclip` doivent apparaître sur la même page et doit être correctement insérés dans le respect des groupes T_EX (à moins que `\AltClipMode` soit en action). Le fichier `denis2.tex` contient un exemple de `\pscharclip`.

40 Importation de fichiers EPS

PSTricks ne possède pas de fonctionnalités pour inclure des fichiers .EPS (Encapsulated PostScript), car il y a d'autres macros très bien faites et éprouvées pour cela. Si vous utilisez `d\eps` de Rokicki, essayer donc `epsf.tex/epsf.sty` du sus-dit.

Ce que PSTricks est capable de faire est d'embellir vos dessins EPS. Vous pouvez inclure un fichier EPS dans l'argument de `\rput`, comme dans

```
\rput(3,3){\epsfbox{myfile.eps}}
```

et de là vous pouvez inclure un fichier EPS dans l'environnement `\pspicture`. Avec `\psgrid`, vous pouvez trouver les coordonnées de tout graphique ou de tout texte que vous voulez rajouter. Ceci fonctionne même si l'image a un contour de boîte bizarre d'un point de vue T_EX.

Cependant ce n'est pas le meilleur moyen de travailler avec un fichier EPS. Si le fichier PostScript de contour de boîte est de la taille désirée de l'image finale, alors essayer

```
\hbox{<picture objects> \epsfbox{<file.eps>}}
```

Ceci placera toutes vos images au coin inférieur gauche de votre fichier EPS. `\epsfbox` prend soin de laisser la bonne quantité d'espace.

Si vous avez besoin des dimensions de la boîte d'un fichier EPS, vous

pouvez essayer les programmes de calculs automatiques de dimensions, comme `bbfig` (distribué avec `dvips` de Rokicki). Cependant, de tels programmes sont facilement pris en défaut ; le seul moyen sûr de déterminer les dimensions d'une boîte est visuelle. `\psgrid` est un bon outil pour cela.

41 Exportation de fichiers EPS



Vous devez charger `pst2eps.tex` ou `pst2eps.sty` pour utiliser les macros PSTricks décrites dans cette section.

Si vous voulez exporter un fichier EPS qui contient à la fois du graphique et du texte, alors vous avez besoin d'un driver DVI-to-PS qui supporte cette opération. Si vous voulez exporter du graphique pur, alors vous pouvez utiliser la commande `\PSTricksEPS`. Ces deux options sont décrites dans cette section.

Les versions les plus récentes de `dvips` de Rokicki comportent une `-E` option pour créer des fichiers EPS à partir de fichiers `.dvi` de \TeX . Ainsi,

```
dviufoo :dvi - E - ofoo :eps
```

Votre document doit tenir sur une seule page. `dvips` trouvera la boîte la plus grande qui englobera juste les caractères imprimés sur la page. Ceci fonctionne le mieux possible avec des fontes de sortie (PostScript), de façon à ce que le fichier EPS soit indépendant de la mise à l'échelle et de la résolution.

Il y a deux aspects embarrassants avec cette méthode. Vous désirez peut-être une boîte différente de celle qui est calculée par `dvips` (en particulier, `dvips` ignore tout le code PostScript engendré par PSTricks quand il calcule les dimensions de la boîte), et vous aurez du mal à supprimer des en-têtes et des pieds de page ajoutés par les routines de sortie.

PSTricks contient un environnement qui permet de contourner ces deux inconvénients :

```
\TeXtoEPS  
truc  
\endTeXtoEPS
```

C'est tout ce qui devrait apparaître dans votre document, mais les en-tête et tout ce qui devrait être normalement ajouté par les routines de sortie est ignoré. `dvips` essaiera à nouveau de trouver la boîte a plus proche, mais il

traitera *truc* comme s'il y avait un cadre autour. Ainsi, la boîte contiendra à coup sur *truc*, mais pourra être plus grande s'il y a des sorties en dehors des limites de cette boîte. Si les dimensions de la boîte ne sont pas correctes, alors vous devrez éditer les

```
%%BoundingBox <llx lly urx ury>
```

spécifications dans le fichier EPS à la main.

Si votre objectif est de faire un fichier EPS pour l'inclure dans d'autres documents, alors `dvips-E` est la marche à suivre. Cependant, il peut être utile d'engendrer un fichier EPS à partir d'objets graphiques PSTricks et de l'inclure dans le même document,¹⁷ plutôt que d'inclure juste les graphiques PSTricks directement, car `TEX` traite les graphiques PSTricks seulement quand le fichier EPS est créé la première fois ou mis à jour. Ainsi, vous pouvez éditer votre fichier et prévisualiser les graphiques, sans avoir à traiter tous les objets graphiques PSTricks à chaque fois que vous faites une correction typographique. Cette accélération peut être significative avec des graphiques complexes tels que ceux de `\pslistplot` avec beaucoup de données.

Pour créer un fichier EPS à partir d'objets graphiques PSTricks, utiliser

```
\PSTtoEPS[par]{file}{objetsgraphiques }
```

Le fichier est immédiatement créé, et ainsi vous pouvez l'inclure dans le même document (après la commande `\PSTtoEPS`) autant de fois que vous le voulez. Contrairement à `dvips-E`, seuls les objets graphiques purs sont traités (par exemple, les commandes `\rput` n'ont pas d'action).

`\PSTtoEPS` ne peut pas calculer les dimensions de la boîte du fichier EPS. Vous devez le spécifier vous-même, en fixant les paramètres suivants :

<code>bbllx=dim</code>	Par défaut : -1 pt
<code>bbly=dim</code>	Par défaut : -1 pt
<code>bburx=dim</code>	Par défaut : 1 pt
<code>bbury=dim</code>	Par défaut : 1 pt

Noter que si le fichier EPS doit simplement être intégré dans un dessin PSTricks avec `\rput` il vaut mieux abandonner la boîte créée par défaut.

`\PSTricksEPS` utilise aussi les paramètres suivants :

17. Voir la section précédente sur l'importation de fichiers EPS

headerfile=*file***Par défaut : s**

Ce paramètre spécifie au début du fichier Postscript qu'ils sont inclus dans un fichier EPS. L'argument peut contenir un ou plusieurs noms de fichiers, séparés par des virgules. Si vous avez plusieurs fichiers, la liste entière doit être enfermée par deux accolades { }.

headers=*none/all/user***Par défaut : none**

Avec *none*, aucun en-tête de fichiers n'est incluse. Avec *all*, l'en-tête de fichier utilisée par PSTricks et l'en-tête de fichier spécifié par le paramètre **headerfile** sont inclus. Avec *user*, seuls les en-tête de fichiers spécifiés par le paramètre **headerfile** sont inclus. Si le fichier EPS doit être inclus dans un document \TeX qui utilise les mêmes macros PSTricks et ainsi charge les fichiers d'en-tête PSTricks appropriés de toutes façons (en particulier si le fichier EPS doit être inclus dans le même document), alors **headers** doit être fixé à *none* ou *user*.

Aide

A Boîtes

Beaucoup de macros PSTricks ont un argument pour le texte qui est interprété en mode restreint horizontal (en parlé \LaTeX , mode LR) et ensuite transformé de cette façon. C'est toujours le dernier argument de la macro et il est écrit {truc} dans ce guide de l'utilisateur. Ainsi les macros d'encadrements, de rotation, de mise à l'échelle de positionnement et de nuds. J'appellerai ces macros « boîtes LR », et j'utiliserai l'encadrement comme exemple principal dans le texte ci-dessous.

Dans le mode restreint horizontal, l'entrée, composée de caractères réguliers et de boîtes est disposée sur une ligne (longue ou courte). Il n'y a pas de cassure de ligne, ni de plaçage du matériel en mode vertical comme une équation entière en mode **display**. Cependant, le fait que vous puissiez inclure une autre boîte signifie qu'il n'y a pas en fait de restrictions.

D'abord, les environnements d'alignement comme `\halign` ou `tabular` de \LaTeX sont juste des boîtes, et ainsi ne présentent aucun problème. Les environnements d'image et les macros de boîtes sont aussi juste des boîtes. Pour l'instant, il n'y a pas une seule commande PSTricks qui ne peut être mise directement dans l'argument d'une macro « boîtes LR ». Cependant, des paragraphes entiers ou autre matériel en mode vertical comme des équations en mode **display** doivent être placés dans une `\vbox` ou une `\parbox` \LaTeX ou une `minipage`. Les utilisateurs de \LaTeX devront regarder `fancybox.sty` et sa documentation, `fancybox.doc`, pour plus d'éclaircissements et les astuces dans l'utilisation des commandes de boîtes LR.

Les macros de « boîtes LR » PSTricks ont quelques particularités que l'on ne retrouve pas dans les autres macros de « boîtes LR », comme les commandes standard de macros de « boîtes LR » \LaTeX .

Avec les commandes de macros de « boîtes LR » \LaTeX , les contenus sont toujours traités en mode texte, même quand la boîte apparaît en mode mathématique. PSTricks, d'un autre côté préserve le mode mathématique, et entreprend la préservation du style mathématique. \TeX a quatre styles mathématiques : `text`, `display`, `script` et `scriptscript`. En général, si la macro de boîte intervient en mode `displayed math` (mais pas en `subscript math`), les contenus sont traités en style `display`, sinon les contenus sont traités en style `text` (cependant ici les macros PSTricks peuvent faire des erreurs, mais elles n'y sont pour rien). Si vous ne donnez le style correct incluez de façon explicite une commande `\textstyle`, `\displaystyle`, `\scriptstyle` ou `\scriptscriptstyle` au début de l'argument de la macro de boîte.

Dans le cas où vous voulez que vos commandes PSTricks de boîtes LR

traitent les mathématiques comme les autres commandes de boîtes LR, vous pouvez alterner cette particularité oui ou non avec les commandes

`\psmathboxtrue`

`\psmathboxfalse`

Vous pouvez avoir des commandes (de la même veine, mais pas uniquement restreintes aux commandes mathématiques) de façon automatique en insérant au début de chaque boîte LR la commande¹⁸ :

`\everypsbox{commandes}`.

Si vous voulez définir un environnement de boîte LR *nom* à partir d'une commande de boîte LR *cmd*, utilisez

`\pslongbox{nom}{cmd}`

Par exemple, après

`\pslongbox{MonCadre}{\psframebox}`

vous pouvez écrire

`\MonCadre <truc>\endMonCadre`

au lieu de

`\psframebox{<truc>}`

Les utilisateurs de \LaTeX pourront écrire

`\begin{MonCadre} <truc>\end{MonCadre}`

Il est mieux pour vous d'être sûr que *cmd* est une commande de boîte LR PSTricks ; si ce n'est pas le cas, des erreurs désagréables peuvent survenir.

Les environnements comme ceci ont des propriétés agréables :

- La syntaxe est plus lisible quand *truc* est long.
- Il est plus facile de construire des commandes de boîtes LR composites. Par exemple, voici un environnement de minipage encadrée pour \TeX :

```
\pslongbox{MyFrame}{\psframebox}
\newenvironment{fminipage}
{\MyFrame\begin{minipage}}
```

18. C'est un registre symbolique

```
{\end{minipage}\endMyFrame}
```

- Vous incluez du texte verbatim et autres codes `\catcode` dans *truc*.

Le reste de cette section détaille l'inclusion de texte verbatim dans les commandes et les environnements de boîtes LR, pour ceux qui sont intéressés. `fancybox.sty` contient aussi quelques macros intéressantes et des trucs, quelques uns étant utiles pour les commandes de boîtes LR.

La raison pour laquelle vous ne pouvez normalement pas inclure du texte verbatim dans l'argument d'une commande de boîtes LR est que \TeX lit l'argument entier avant de compiler les changements de `\catcode`, et à ce point il est trop tard pour changer les catégories de codes. Si ceci est de l'hébreu pour vous¹⁹, essayer juste cet exemple \LaTeX pour voir le problème :

```
\psframebox\verb+\foobar+\endMyFrame
```

Les environnements de boîtes LR définis avec `\pslongbox` n'ont pas ce problème car *truc* n'est pas interprété comme un argument. Ainsi, ceci marche :

```
\pslongbox{MyFrame}{\psframebox  
\MyFrame \verb+\foo{bar}+\endMyFrame  
\foo{bar}}
```

Les commandes

`\psverbboxtrue`

`\psverbboxfalse`

mettent en marche ou non, respectivement, un mode spécial `PSTricks` qui vous permet d'inclure du texte verbatim dans n'importe quelle commande de boîte LR. Par exemple :

19. Incidemment, beaucoup de macros de langues, comme `greek.tex`, utilisent des astuces `catcode` qui peuvent poser problème dans les macros de boîtes LR.

```
\psverbboxtrue
\psframebox{\verb+\foo{bar}+}
```

```
\foo{bar}
```

Cependant, ce n'est pas aussi robuste. Vous devez de façon explicite grouper les commandes de couleurs dans *truc*, et les commandes de boîtes LR qui habituellement ignorent les espaces qui suivent *{truc}* ne le feront pas lorsque `\psverbboxtrue` est effectif.

B Encore plus de tuyaux et d'astuces

1 Comment puis-je tourner/encadrer ceci et qu'en est-il avec \TeX ?

Regarder `fancybox.sty` et sa documentation.

2 Comment puis-je supprimer le PostScript de façon à pouvoir utiliser mon document avec un driver dvi non-PostScript? Placer la commande

```
\PSTricksOff
```

au début de votre document. Vous serez alors capable d'imprimer ou de prévisualiser des brouillons de votre document (moins le PostScript, et peut-être une apparence un peu drôle) avec tout driver dvi.

3 Comment améliorer la traduction des demi-tons?

Ceci peut être à considérer quand vous avez un demi-ton dans le fond et du texte par dessus. Vous pouvez essayer de taper

```
\pstverb{106 45 {dup mul exch dup mul add 1.0 exch sub} setscreen}
```

avant le demi-ton, ou en en-tête (comme dans les en-têtes et les pieds de page, pas comme les en-têtes de fichiers PostScript), si vous voulez avoir l'effet sur chaque page.

`setscreen` est un opérateur dépendant de la plate-forme.

4 Quels caractères spéciaux sont-ils actifs avec `PSTricks`?

C Insertion de code PostScript

Pour apprendre le langage PostScript, consulter le « *Adobe's PostScript Language Tutorial and Cookbook* » (le "Blue Book"), ou *PostScript by Example* (1992) de Henry McGilton and Mary Campione. Les deux sont édités chez Addison-Wesley. Vous trouverez que l'Appendice du "Blue Book", plus une explication de comment marche une pile, est tout ce dont vous

avez besoin pour écrire un code simple pour programmer des nombres (par exemple, pour spécifier des coordonnées ou des points en se servant de PostScript).

Vous souhaitez définir des macros \TeX pour inclure des fragments PostScript dans divers endroits. Toutes les macros \TeX sont développées avant d'être traduites en PostScript. Ce n'est pas toujours clair de savoir ce que cela signifie. Par exemple, supposer que vous écriviez

```
\def\mydata{23 43}
\psline(!47 \mydata add)
\psline(!47 \mydata add)
\psline(!47 \mydata add)
\psline(!47 \mydata add)
```

Vous obtiendrez une erreur PostScript pour chacune des commandes $\backslash\text{psline}$. Pour voir comment l'argument est développé, essayez d'utiliser $\backslash\text{edef}$ et $\backslash\text{show}$ de \TeX . Par exemple,

```
\def\mydata{23 43}
\edel\temp{47 \mydata add}
\show\temp
\edef\temp{47 \mydata\ add}
\show\temp
\edef\temp{47 \mydata add}
\show\temp
\edel\temp{47 \mydata{ } add}
\show\temp
```

\TeX développe le code, assigne ses valeurs à $\backslash\text{temp}$, et ensuite affiche les valeurs de $\backslash\text{temp}$ sur votre console. Tapez *return* pour compiler. Vous verrez que les quatre exemples se développent, respectivement, en :

```
47 23 43add
47 23 43\ add
47 23 43\penalty \@M \ add
47 23 43{ } add
```

Tout ce que vous vouliez c'est une espace entre le 43 et add. La commande $\backslash\text{space}$ fera l'affaire :

```
\psline(!47 \mydata\space add)
```

Vous pouvez inclure des paires de parenthèses { }; elles passeront en verbatim pour PostScript. Cependant, pour inclure une parenthèse ouvrante ou fermante, vous devez utiliser, respectivement,

Insertion de code PostScript

`\pslbrace`

`\psrbrace`

Ne vous ennuyez pas à taper `\}` ou `\{`.

Quoique vous insériez comme code PostScript dans un argument PSTricks, le dictionnaire au sommet de la pile du dictionnaire est `tx@Dict`, qui est le dictionnaire principal de PSTricks. Si vous voulez définir vos propres variables, vous avez deux options :

Simplest Toujours inclure un `@` dans les noms de variable, car PSTricks n'utilise jamais `@` dans ses noms de variables. Vous avez un risque de dépassement du dictionnaire `tx@Dict`, qui dépend de votre interpréteur PostScript. Vous avez aussi des chances d'avoir des conflits avec les définitions de quelqu'un d'autre, s'il y a plusieurs auteurs sur le même document.

Safest Créer un dictionnaire nommé `TDict` pour vos calculs de travail. Soyez sûr de le supprimer de la pile du dictionnaire à la fin de tout code inséré dans l'argument. Ainsi,

```
TDict 10 dict def TDict begin <your code> end
```

D Pour régler un problème

1 Pourquoi le document bombe t-il au moment de l'impression quand le premier élément d'un fichier \LaTeX est un flottant ?

Quand le premier élément d'un fichier \LaTeX est un flottant, les `\special` dans le préambule sont ignorés. En particulier, le `\special` d'inclusion du fichier d'en-tête PSTricks est perdu. La solution est de mettre quelque chose avant le flottant, ou d'inclure le fichier d'en-tête par une option de commande en ligne avec votre driver de dvi-to-ps.

2 J'ai converti un fichier `.dvi` en PostScript, et l'ai adressé par courriel à un collègue. Pour moi il s'imprime bien mais bombe sur son imprimante.

Voici la raison la plus vraisemblable (mais pas la seule) de ce problème. Les fichiers PostScript que vous obtenez en utilisant PSTricks peuvent avoir des lignes longues. Ceci devrait être toléré par tout interpréteur PostScript correct, mais les lignes peuvent être écourtées dans l'envoi par courriel.

Il n'y a pas de moyen de fixer ceci dans PSTricks, mais vous pouvez emballer les lignes de vos fichiers PostScript dans l'envoi par courriel. Par exemple, sous UNIX vous pouvez utiliser `uuencode` et `uudecode`, ou vous pouvez utiliser le script AWK pour mettre ensemble les lignes :

```
#!/bin/sh
```

Pour régler un problème

```
# Ce script emballe toutes les lignes
# traitement (si le script est appelé wrap) :
# wrap < infile > outfile
awk'
BEGIN {
N = 78 # longueur de ligne maximale
}
{ if (length($0)<=N)
print else {
curlength = 0
for (i = 1 ; i <=NF ; i++) {
if ((curlength = curlength + length($i) + 1) > N) {
printf printf curlength = length($i)
else
printf %s }
printf } }'
```

3 Les commandes de couleur entraînent l'insertion d'un espace vertical supplémentaire.

Par exemple, ceci peut arriver si vous démarrez une `\parbox` ou une `p{ }` de colonne \LaTeX avec une commande de couleur. la solution habituelle est de faire précéder commandes de couleur de `\leavevmode`.

4 Les commandes de couleur interfèrent avec d'autres macros de couleur que j'utilise.

Essayez de mettre la commande `\altcolormode` au début de votre document. Ceci peut vous aider ou non. Faites bien attention que la portée des commandes de couleur ne s'étendent pas sur plusieurs pages. C'est en général un schéma de couleur moins robuste.

5 Comment empêcher que les flottants prennent la même couleur que ce qui l'entoure ?

C'est facile : placer juste une commande de couleur explicite au début du flottant, par exemple, `\black`.

6 Quand j'utilise une commandes de couleur dans une macro de boîte ou avec `\setbox`, les couleurs deviennent reserrées.

```
Si \mybox est un registre de boîte , et que vous écrivez
\green Ho Hum.
\setbox \mybox= \hbox{Foo bar \blue fee fum}
Hi Ho. \red Diddley-dee
\box \mybox hum dee do
```

Pour régler un problème

alors quand `\mybox` est insérée, la couleur en cours est rouge et `so Foo bar` devient rouge (plutôt que vert, qui était la couleur en cours quand la boîte a été mise). La commande qui fait revenir de `bleu` à la couleur précédente `vert`, quand la boîte est établie, est exécutée après la fermeture de la boîte, ce qui signifie que `Hi Ho` est vert, mais `hum dee` est toujours de couleur bleue.

Ce fonctionnement curieux est du au fait que `TEX` ne supporte pas la couleur de façon interne, de la façon dont il supporte les commandes de fontes. La première chose à faire est de mettre de façon explicite les commandes de couleurs dans la boîte. Ensuite, soyez sûr que la couleur est noire quand vous introduisez votre boîte. Troisièmement, faites les autres changements de couleurs explicites là où c'est nécessaire si vous continuez à avoir des problèmes. Le schéma de couleur invoqué par `\altcolormode` est légèrement meilleur si vous suivez les deux premières règles.

Noter que les différentes macros de boîtes utilisent `\setbox` et ainsi ces anomalies peuvent survenir de façon inattendue.

Index

- AltClipMode, 55, 78
- \altcolormode, **88**, 89
- angle (parameter), **61**, 62, 63, 72
- angleA (parameter), 63-65
- angleB (parameter), 63, 64
- \Aput, **68**
- \aput, 67, **68**, 68
- arcangle (parameter), **61**
- arcangleA (parameter), 63
- arcangleB (parameter), 63
- arcsep (parameter), **13**
- arcsepA (parameter), **12**, 12, 13
- arcsepB (parameter), **13**
- arm (parameter), **61**, 63
- armA (parameter), 63-65
- armB (parameter), 63-65
- arrowinset (parameter), **30**, 30
- arrowlength (parameter), **30**, 30
- \arrows, **40**
- arrows (parameter), 9, 11, 19, 20, **28**, 29, 48
- arrowscale (parameter), **30**, 30
- arrowsize (parameter), **30**
- axesstyle (parameter), **51**
- bbllx (parameter), **80**
- bbly (parameter), **80**
- bburx (parameter), **80**
- bbury (parameter), **80**
- \black, 89
- \blue, 89
- border (parameter), **25**, 25, 33, 62
- bordercolor (parameter), **25**, 25
- boxsep (parameter), **52**, 53, 54
- \Bput, **68**
- \bput, 67, **68**, 68
- bracketlength (parameter), **30**
- \Cartesian, **72**, 72
- \circlenode, **60**
- \clipbox, **54**
- \closedshadow, **38**
- \closepath, 34, **36**, 36
- \cnode, **60**
- \cnodeput, **60**
- \code, **39**, 40
- coilarm (parameter), **70**, 70, 71
- coilarmA (parameter), 70
- coilarmB (parameter), 70
- coilaspect (parameter), **70**, 70, 71
- coilheight (parameter), **70**, 70
- coilinc (parameter), **70**, 70
- coilwidth (parameter), **70**, 70
- \coor, **39**, 40
- cornersize (parameter), **10**, 10, 54
- \cput, **53**, 60
- curvature (parameter), **14**
- \curveto, **39**, 39
- dash (parameter), **25**
- dashed (parameter), 33
- \dataplot, **20**, 20, 21
- \degrees, **8**, 8, 72
- \dim, **39**
- dimen (parameter), **26**
- \DontKillGlue, **42**
- dotangle (parameter), **16**, 16
- dotscale (parameter), **16**
- dotsep (parameter), **25**
- dotsize (parameter), 16, **30**
- dotstyle (parameter), **16**, 16
- dotted (parameter), 33
- doublecolor (parameter), 25, **26**
- doubleline (parameter), **25**, 25, 26, 33
- doublesep (parameter), **25**, 25
- Dx (parameter), **49**, 49
- dx (parameter), **49**, 49
- Dy (parameter), **49**, 49
- dy (parameter), **49**
- \endoverlaybox, **73**
- \endpscharclip, **78**, 78
- \endpsclip, **54**, 54, 55, 78

-
- `\endpspicture`, **41**
 - `\endTeXtoEPS`, **79**
 - `\everypsbox`, **83**
 - file, **40**
 - `\fileplot`, **20, 20**
 - `\fill`, **33, 37**
 - fillcolor (parameter), **9, 27, 28, 52**
 - fillstyle (parameter), **9, 27, 28, 32, 33, 51, 74, 77**
 - framearc (parameter), **10, 10**
 - `\framenode`, **60**
 - framesep (parameter), **52**
 - gradangle (parameter), **75**
 - gradbegin (parameter), **74, 75**
 - gradend (parameter), **74, 75**
 - gradhines (parameter), **75**
 - gradmidpoint (parameter), **75**
 - `\gray`, **4**
 - `\grestore`, **37, 37, 38**
 - gridcolor (parameter), **18**
 - griddots (parameter), **18, 18**
 - gridlabelcolor (parameter), **18**
 - gridlabels (parameter), **18**
 - gridwidth (parameter), **18**
 - `\gsave`, **37, 37, 38**
 - hatchangle (parameter), **27, 27**
 - hatchcolor (parameter), **27**
 - hatchsep (parameter), **27**
 - hatchwidth (parameter), **27**
 - headerfile (parameter), **81, 81**
 - headers (parameter), **81, 81**
 - `\KiIIGlue`, **42**
 - labels (parameter), **50**
 - labelsep (parameter), **44, 50**
 - liftpen (parameter), **35, 35, 37**
 - linearc (parameter), **10, 10, 19-21, 54, 63, 64, 71**
 - linecolor (parameter), **8, 8, 9, 24, 28, 32, 33, 52**
 - linestyle (parameter), **24, 25, 28, 32, 33, 51, 55, 76, 77**
 - `\lineto`, **39, 39**
 - linetype (parameter), **33, 33**
 - linewidth (parameter), **8, 8, 11, 16, 24, 28-30, 32, 33**
 - `\listplot`, **20, 21, 21**
 - loopsize (parameter), **62, 65**
 - `\Lput`, **67, 67**
 - `\lput`, **62, 67, 67, 68**
 - `\movepath`, **38**
 - `\moveto`, **36, 36**
 - `\Mput`, **67, 67**
 - `\mput`, **68**
 - `\mrestore`, **38, 38**
 - `\msave`, **38, 38**
 - `\multido`, **47, 51**
 - `\multips`, **46, 46, 51**
 - `\multirput`, **46, 46**
 - `\ncangle`, **64, 64, 66**
 - `\ncangles`, **64, 64**
 - `\ncarc`, **61, 63, 63, 65, 66**
 - `\ncbar`, **63, 65, 66**
 - `\nccircle`, **65, 65, 66**
 - `\nccoil`, **71**
 - `\nccurve`, **61, 62, 63, 65, 66**
 - `\ncdiag`, **63, 64-66**
 - `\ncdiagg`, **64, 66**
 - `\ncLine`, **62, 65, 68**
 - `\ncline`, **62, 62, 65, 66, 68, 69, 71**
 - `\ncloop`, **62, 65, 66**
 - ncurv (parameter), **61, 62, 63**
 - `\nczigzag`, **71**
 - `\newcmypcolor`, **5**
 - `\newgray`, **5**
 - `\newsbcolor`, **5**
 - `\newpath`, **36**
 - `\newpsobject`, **31, 31, 54**

`\newpsstyle`, **31**, 31
`\newrgbcolor`, **5**
`nodesep` (parameter), **61**, 62-64, 72
`nodesepA` (parameter), 65
`\NormalCoor`, **73**
`offset` (parameter), **61**, 62-64, 67, 72
`\openshadow`, **38**
`origin` (parameter) **24**, 33
`\ovalnode`, **60**
`\overlaybox`, **73**
`Ox`, `Oy` (parameter), **49**, 49, 50
`oy` (parameter), **49**, 49
`\parabola`, **14**, 14
parameters :
`Dx`, **49**, 49
`Dy`, **49**, 49
`Ox`, **49**, 49, 50
`Oy`, **49**, 49, 50
`angleA`, 63-65
`angleB`, 63, 64
`angle`, **61**, 62, 63, 72
`arcangleA`, 63
`arcangleB`, 63
`arcangle`, **61**
`arcsepA`, **12**, 12, 13
`arcsepB`, **12**, 13
`arcsep`, **13**
`armA`, 63-65
`armB`, 63-65
`arm`, **61**, 63
`arrowinset`, **30**, 30
`arrowlength`, **30**, 30
`arrowscale`, **30**, 30
`arrowsize`, **30**
`arrows`, 9, 11, 19, 20, **28**, 29, 48
`axesstyle`, **51**
`bbllx`, **80**
`bbllx`, **80**
`bburx`, **80**
`bbury`, **80**
`bordercolor`, **25**, 25
`border`, **25**, 25, 33, 62
`boxsep`, **52**, 53, 54
`bracketlength`, **30**
`coliarmA`, 70
`coilarmB`, 70
`coilarm`, **70**, 70, 71
`coilaspect`, **70**, 70, 71
`coilheight`, **70**, 7024, 33
`coilinc`, **70**, 70
`coilwidth`, **70**, 70
`cornersize`, **10**, 10, 54
`curvature`, **14**
`dash`, **25**
`dashed`, 33
`dimen`, **26**
`dotangle`, **16**, 16
`dotscale`, **16**
`dotsep`, **25**
`dotsize`, 16, **30**
`dotstyle`, **16**, 16
`dotted`, 33
`doublecolor`, 25, **26**
`doubleline`, **25**, 25, 26, 33
`doublesep`, **25**, 25
`dx`, **49**, 49
`dy`, 49
`fillcolor`, 9, **27**, 28, 52
`fillstyle`, 9, **27**, 28, 32, 33, 51, 74, 77
`framearc`, **10**, 10
`framesep`, **52**
`gradangle`, **75**
`gradbegin`, **74**, 75
`gradend`, **74**, 75
`gradlines`, **75**
`gradmidpoint`, **75**
`gridcolor`, **18**
`griddots`, **18**, **18**
`gridlabelcolor`, **18**
`gridlabels`, **18**
`gridwidth`, **18**
`hatchangle`, **27**, 27

-
- hatchcolor, **27**
 - hatchsep, **27**
 - hatchwidth, **27**
 - headerfile, **81**, 81
 - headers, **81**, 81
 - labels, **50**
 - labelsep, **44**, 50
 - liftpen, **35**, 35, 37
 - linearc, **10**, 10, 19-21, 54, 63, 64, 71
 - linecolor, **8**, 8, 9, 24, 28, 32, 33, 52
 - linestyle, **24**, 25, 28, 32, 33, 51, 55, 76, 77
 - linetype, **33**, 33
 - linewidth, **8**, 8, 11, 16, 24, 28-30, 32,33
 - loopsize, **62**, 65
 - ncurv, **61**, 62, 63
 - nodesep, **61**, 62-64, 72
 - nodesepA, 65
 - offset, **61**, 62-64, 67, 72
 - origin, **24**, 33
 - oy, **49**, 49
 - plotpoints, **22**, 22
 - plotstyle, **19**, 19, 34
 - pspicture, 41
 - rbracketlength, **30**
 - rectarc, 54
 - runit, **7**, 8
 - shadowangle, **26**, 26
 - shadowcolor, **26**, 26
 - shadowsize, **26**, 26, 53
 - shadow, **26**, 26, 33
 - showorigin, **50**
 - showpoints, **9**, 12, 14-16, 19-21, 33
 - style, 31
 - subgridcolor, **18**
 - subgriddiv, **18**
 - subgriddots, **18**
 - subgridwidth, **18**
 - swapaxes, **24**, 33
 - tbarsize, 16, **30**
 - ticks, **50**
 - ticksize, **50**
 - tickstyle, **50**, 50
 - unit, **7**, 7, 19, 72
 - xunit, **7**, 8, 17, 18, 72
 - yunit, **7**, 7, 8, 17, 18, 72
 - \parametricplot, **22**, 22, 23
 - \pcangle, **66**
 - \pcarc, **65**
 - \pcbar, **65**
 - \pccoil, **71**
 - \pccurve, 61, **65**
 - \pcdiag, **65**
 - \pcline, **65**, 67, 71
 - \pclloop, 62, **66**
 - \pczigzag, **71**
 - \plotfile, 20
 - plotpoints (parameter), **22**, 22
 - plotstyle (parameter), **19**, 19, 34
 - \pnode, **60**
 - \Polar, **72**, 72
 - \psaddtolength, **7**
 - \psarc, **12**, 12, 13, 61
 - \psarcn, **13**, 13
 - \psaxes, 17, **48**, 49-51
 - \psbezier, **13**, 13, 34, 35
 - \psborder, 25
 - \psccurve, **15**, 19
 - \pscharclip, **78**, 78
 - \pscharpath, **77**, 78
 - \pscicle, **11**, 26
 - \pscicle*, 11
 - \psciclebox, 52, **53**, 53, 60
 - \psclip, **54**, 54, 55, 78
 - \psCoil, **70**, 70, 71
 - \pscoil, **70**, 70, 71
 - \pscure, **15**, 15, 19, 34, 37
 - \pscustom, 13, **32**, 32-34, 36, 37, 39, 46, 54, 61
 - \psdblframebox, **53**, 60
 - \psdots, **15**, 19, 34

- `\psecure`, **15**, 19
- `\psellipse`, **12**, 26
- `\psfill`, 32
- `\psframe`, 9, 10, **11**, 11, 26, 51, 52
- `\psframebox`, **52**, 52-54, 60
- `\psgrid`, **17**, 17-19, 34, 48, 78, 79
- `\pshatchcolor`, 27
- `\pslabelsep`, **44**, 50, 68
- `\pslbrace`, **87**
- `\psline`, 7, **10**, 10, 11, 19, 22, 31, 34, 51, 65, 86
- `\pslinecolor`, 8
- `\pslinewidth`, 8
- `\pslongbox`, **83**, 84
- `\psmathboxfalse`, **83**
- `\psmathboxtrue`, **83**
- `\psovalbox`, 52, **54**, 60
- `\psoverlay`, **73**, 74
- `\pspicture`, 17, **41**, 41, 42, 54, 78
- `pspicture` (parameter), 41
- `\psplot`, **21**, 21-23
- `\pspolygon`, 10, **11**, 19, 28
- `\psrbrace`, **87**
- `\psrunit`, 8
- `\psset`, 5, **6**, 6, 11, 41
- `\pssetlength`, **7**
- `\psshadowbox`, **53**, 60
- `\pstextpath`, **76**, 76, 77
- `\pstheader`, 76
- `\PSTricksEPS`, 79, 80
- `\PSTricksOff`, **85**
- `\pstroke`, 32
- `\pstrotate`, 46
- `\PSTtoEPS`, 20, **80**, 80
- `\pstunit`, 32
- `\pstVerb`, 5, 42, 46, 55, 69, 74
- `\pstverb`, 32
- `\pstverbscale`, 42, 55, 69, 74
- `vpsunit`, 8, 77
- `\psverbboxfalse`, **84**
- `\psverbboxtrue`, 4, **84**, 85
- `\pswedge`, **12**, 26
- `\psxlabel`, **51**
- `\psxunit`, 8, 19
- `\psylabel`, **51**
- `\psyunit`, 8, 19
- `\pszigzag`, **70**, 70, 71
- `\putoverlaybox`, **74**
- `\qdisk`, **11**, 34
- `\qline`, **10**, 34
- `\radians`, **8**
- `rbracketlength` (parameter), **30**
- `\rcoor`, **40**
- `\rcurveto`, **39**
- `\readdata`, **20**, 20, 21
- `rectarc` (parameter), 54
- `\red`, 4
- `\rlineto`, **39**
- `\Rnode`, **59**, 60, 68
- `\rnode`, **59**, 59, 60, 68, 69
- `\RnodeRef`, **59**, 60
- `\rotate`, **38**
- `\Rotatedown`, **56**
- `\rotatedown`, **56**
- `\rotateleft`, **55**
- `\rotateright`, **56**
- `\rput`, 41, **43**, 43-46, 53, 58, 67, 71, 78, 80
- `\Rput`, **45**, 45, 67
- `runit` (parameter), 7, 8
- `\savedata`, **20**, 20
- `\scale`, **38**
- `\scalebox`, **56**
- `\scaleboxto`, **56**
- `\setcolor`, **40**
- `shadow` (parameter), **26**, 26, 33
- `shadowangle` (parameter), **26**, 26
- `shadowcolor` (parameter), **26**, 26
- `shadowsize` (parameter), **26**, 26, 53
- `showorigin` (parameter), **50**
- `showpoints` (parameter), **9**, 12, 14-16, 19-21, 33
- `\SpecialCoor`, 7, 8, **72**, 72, 73

`\stroke`, 33, **36**
`style` (parameter), 31
`subgridcolor` (parameter), **18**
`subgriddiv` (parameter), **18**
`subgriddots` (parameter), **18**
`subgridwidth` (parameter), **18**
`\swapaxes`, **38**
`swapaxes` (parameter), **24**, 33
`tbar` (parameter), 16, **30**
`\TeXtoEPS`, **79**
`ticks` (parameter), **50**
`ticksize` (parameter), **50**
`tickstyle` (parameter), **50**, 50
`\TPoffset`, 77
`\translate`, **38**
`unit` (parameter), **7**, 7, 19, 72
`\uput`, **44**, 44, 45, 68
`xunit` (parameter), **7**, 8, 17, 18, 72
`yunit` (parameter), **7**, 7, 8, 17, 18, 72