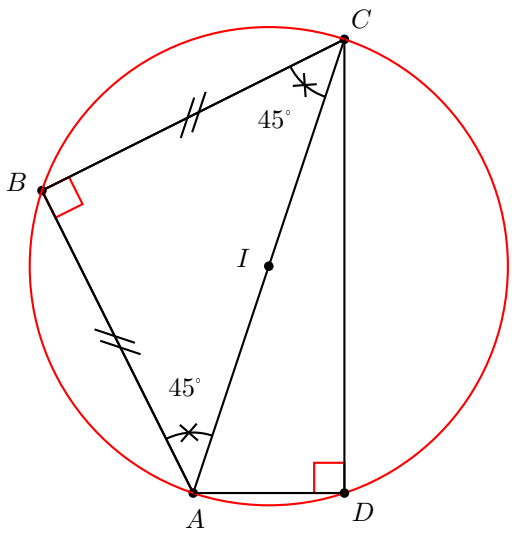
Chronique 13

Euclide – Notions de base

En géométrie, on a souvent des figures à construire ; en analyse, on a souvent des courbes à tracer. Les outils de `PsTricks` sont déjà très performants mais les points sont représentés par leurs coordonnées. Ce serait plus pratique d'avoir un système qui permet de travailler avec des points représentés par leurs noms, comme on fait en géométrie.
C'est exactement ce qui se passe dans le package `euclide` qui sera l'objet de quatre chroniques.
L'auteur de ce package (qui date déjà d'une bonne dizaine d'années) est Dominique Rodriguez et je ne peux que le remercier pour ce magnifique travail !

13.1 Première figure

Voici une figure de géométrie relativement simple mais contenant déjà pas mal d'objets :



On y voit un triangle rectangle ADC, un triangle rectangle isocèle ABC de même hypoténuse $[AC]$, et leur cercle circonscrit de centre I, milieu de $[AC]$.
Les angles droits sont marqués, les angles à la base du triangle isocèle aussi ; de plus les côtés de même longueur sont codés.
On peut tout faire avec `PsTricks` mais les codages des côtés égaux et des angles ne sont pas commodes à faire (particulièrement le codage de l'angle $\widehat{ABC}$).

J'ai donc utilisé le package `euclide` dont le vrai nom est `pst-eucl` qui vient en complément de PsTricks (ce qui veut dire que toutes les instructions de `PsTricks` sont utilisables en plus des instructions de `euclide`).
On écrira donc dans le préambule :
```
\usepackage{pst-all}%      pour PsTricks
\usepackage{pst-eucl}%     pour euclide
```
Attention : je me suis rendu compte qu'il y avait un conflit entre le package `diagbox` (qui permet de tracer des traits obliques dans un tableau) et le package `pst-eucl` ; pour résoudre ce conflit, il faut charger le package `diagbox` avant le package `pst-eucl`.

13.1.1 Les points

Les points se définissent au moyen de l'instruction `\pstGeonode` ; on peut définir plusieurs points qui, par défaut, sont placés automatiquement sur le graphique avec leurs noms :
```
\pstGeonode(0,0){A}(-1,2){B}(1,3){C}(1,0){D}
```
Les noms des points sont écrits automatiquement en mode mathématique : A, B, C et D, ce qui veut dire qu'il ne faut pas mettre le signe $. Si on veut qu'ils ne soient pas écrits en mode mathématique, il suffit d'utiliser le booléen `PtNameMath` :
```
\psset{PtNameMath=false}
```
Les noms sont placés automatiquement horizontalement à droite du dessin du point ; on peut modifier cette position au moyen de la variable `PosAngle` qui donne, en degrés, l'angle fait par la position du point avec l'horizontale droite (qui correspond à 0) :
```
\pstGeonode[PosAngle=-45](1,0){D}
```
Quand il y a plusieurs points, les positions des points peuvent être définies par une liste :
```
\pstGeonode[PosAngle={-90,180,60,-45}](0,0){A}(-1,2){B}(1,3){C}(1,0){D}
```
Enfin, les dessins des points sont gérés par la variable `PointSymbol` ; et si l'on ne veut pas que les points soient dessinés, on entrera en option : `PointSymbol=none`.

13.1.2 Les triangles

Le package `euclide` est assez performant dans la gestion des triangles : on peut les tracer en utilisant une seule instruction qui définit à la volée les sommets par leurs coordonnées et en marquant leurs noms. Par exemple le triangle ABC sera défini par l'instruction :
```
\pstTriangle(0,0){A}(-1,2){B}(1,3){C}
```
Première remarque : les points sont définis par l'instruction `\pstTriangle`, ce n'est donc pas la peine de les avoir définis préalablement au moyen de `\pstGeonode`.
Le noms des points sont placés automatiquement et de façon intelligente, sur la bissectrice de l'angle au sommet. On peut naturellement modifier les positions des points au moyen des trois variables `PosAngleA`, `PosAngleB` et `PosAngleC` correspondant respectivement au premier sommet du triangle, au deuxième et au troisième.
On peut contrôler les dessins des trois sommets par les variables `PointSymbolA`, `PointSymbolB` et `PointSymbolC`, et même leurs noms par les variables `PointNameA`, `PointNameB` et `PointNameC` ; mais pourquoi changerait-on leurs noms ?
L'épaisseur du trait est gérée par la variable `linewidth`, tout comme dans `PsTricks` ; la valeur par défaut est `\pslinewidth` et on peut régler la largeur du trait en fonction de cette largeur :
```
linewidth=1.5\pslinewidth
```
On peut, naturellement, définir la largeur du trait en points ou en millimètres.

13.1.3 Marquage des angles

Angle droit

L'angle droit $\widehat{ABC}$ se marque au moyen de l'instruction `\pstRightAngle`. La marque de l'angle droit est un petit carré placé automatiquement dont le côté vaut `0,28 unit` ; comme je l'ai trouvée

un peu trop grande, je l'ai réduite au moyen de la variable `RightAngleSize` :

```
\pstRightAngle[linecolor=red,RightAngleSize=0.2]{A}{B}{C}
```

Dans le marquage d'un angle droit, l'ordre des points ne compte pas : on obtient le même marquage en entrant `{A}{B}{C}` ou `{C}{B}{A}` ; il faut quand même que le sommet correspondant à l'angle droit soit au milieu !

Angle non droit

Un angle non droit se marque avec l'instruction `\pstMarkAngle`. Là encore, certaines variables permettent de tout contrôler comme `Mark`, si on veut coder l'angle, ou `LabelSep`, si on veut changer la distance entre le sommet et le label :

```
\pstMarkAngle[Mark=MarkCros,LabelSep=0.7]{C}{A}{B}{$45\degres$}
```

Les angles sont définis dans le sens trigonométrique donc l'angle `{C}{A}{B}` n'est pas le même que l'angle `{B}{A}{C}`.
C'est l'option `Mark=MarkCros` qui marque une petite croix dans l'angle ; les autres marques possibles sont `pstslash` (un trait), `pstslashh` (deux traits), `pstslashhh` (trois traits), `MarkHash`, `MarkHashh` et `MarkHashhh` (respectivement un, deux et trois traits épais), et enfin `MarkCross` (double croix).
Attention aux majuscules-minuscules dans les noms de ces variables !
Ces marques servent aussi à coder des segments (voir paragraphe 13.1.4).
L'étiquette (ici 45°) peut être n'importe quel texte ; elle est placée à une distance de `1 unit` du sommet de l'angle ; comme je trouve que c'est un peu loin, j'ai écrit `LabelSep=0.7`.
Si on ne veut pas d'étiquette, on entre une chaine vide `{}` en quatrième paramètre de l'angle.
Enfin, pour tracer un angle orienté, il suffit de rentrer `arrows=->` comme option.

13.1.4 Marquage des segments

Il reste à coder les segments égaux du triangle isocèle ABC ; c'est l'instruction `\pstSegmentMark` qui fait le travail, et c'est la variable `SegmentSymbol` qui détermine la marque de codage :

```
\pstSegmentMark[SegmentSymbol=MarkHashh]{A}{B}
```

Par défaut, le segment est tracé en même temps que sa marque et la marque par défaut est `MarkHashh`.
Si la marque est l'une des trois `MarkHash`, on peut modifier l'angle que fait le trait avec le segment grâce à la variable `MarkAngle`.
On peut même modifier la longueur de ces marques avec la variable `MarkHashLength`, et même l'écart qu'il y a entre deux traits avec la variable `MarkHashSep`.

13.1.5 Cercle et milieu

Première méthode

- Pour tracer le cercle de diamètre $[AC]$, on utilise l'instruction `\pstCircleAB` :

```
\pstCircleAB[linecolor=red]{A}{C}
```

- Une instruction permet de déterminer, tracer et nommer le milieu d'un segment ; c'est `\pstMiddleAB` qu'il faut utiliser :

```
\pstMiddleAB{A}{C}{I}
```

  On définit ainsi le milieu I du segment $[AC]$.

Seconde méthode

- On trace d'abord le milieu I du segment $[AC]$ par l'instruction `\pstMiddleAB{A}{C}{I}`.
- On trace le cercle de centre I passant par le point A en utilisant l'instruction `\pstCircleOA` :

```
\pstCircleOA[linecolor=red]{I}{A}
```

13.1.6 Le code de la figure

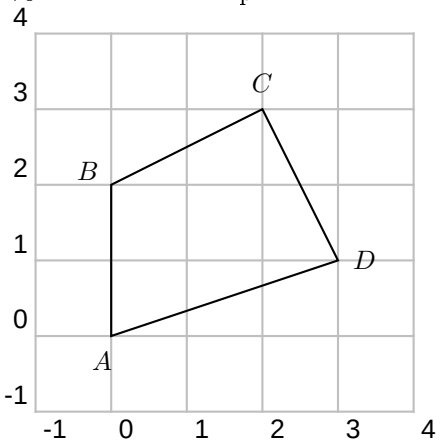
Voici le code complet de la figure de la page 63 :

```
\def\xmin{-1.5} \def\xmax{2.5} \def\ymin{-0.5} \def\ymax{3.5}
\begin{pspicture}(\xmin,\ymin)(\xmax,\ymax)
  \pstTriangle(0,0){A}(-1,2){B}(1,3){C}%            triangle ABC
  \pstRightAngle[linecolor=red,RightAngleSize=0.2]{A}{B}{C}% codage angle droit
  \pstSegmentMark[SegmentSymbol=MarkHashh]{A}{B}%   segment AB et codage
  \pstSegmentMark[SegmentSymbol=MarkHashh]{B}{C}%   segment BC et codage
  \pstMarkAngle[Mark=MarkCros,LabelSep=0.7]{C}{A}{B}{$45\degres$}% codage CAB
  \pstMarkAngle[Mark=MarkCros,LabelSep=0.7]{B}{C}{A}{$45\degres$}% codage BCA
  \pstCircleAB[linecolor=red]{A}{C}%                cercle de diamètre AC
  \pstGeonode[PosAngle=-45](1,0){D}%                point D
  \pstRightAngle[linecolor=red,RightAngleSize=0.2]{A}{D}{C}% codage angle droit
  \pstLineAB{A}{D}  \pstLineAB{D}{C}%               segments AD et DC
  \pstMiddleAB{A}{C}{I}%                            milieu I de AC
\end{pspicture}
```

13.2 D'autres outils de base

13.2.1 Les polygones

Pour tracer des polygones de plus de 3 sommets, c'est très simple : on définit les points et on demande de les relier pour en faire un polygone. C'est la variable `CurveType` qu'il faut utiliser. Voici un court exemple :




```
\psset{unit=1cm,PointSymbol=none}
\def\xmin{-1} \def\xmax{4}
\def\ymin{-1} \def\ymax{4}
\begin{pspicture}(\xmin,\ymin)(\xmax,\ymax)
\psgrid[subgriddiv=1,gridcolor=lightgray]
\pstGeonode
   [PosAngle={-110,150,90,0},CurveType=polygon]
   (0,0){A}(0,2){B}(2,3){C}(3,1){D}
\end{pspicture}
```

On remarque le `PointSymbol=none` dans `\psset` pour ne pas afficher les points.

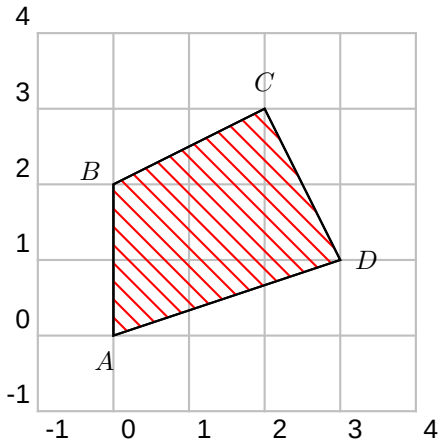
La construction est quand même rapide !
À part `polygon`, les autres options pour `CurveType` sont `polyline` (ligne brisée ouverte) et `curve` (courbe ouverte).
Comme on peut mélanger des instructions de `PsTricks` avec celles de `pst-eucl`, on va voir comment hachurer le polygone tracé précédemment. En `PsTricks`, c'est l'instruction `\pspolygon` qui permet de tracer un polygone dont les sommets sont définis par leurs coordonnées ; en utilisant conjointement `pst-all` (tous les packages de `PsTricks`) et `pst-eucl`, on peut tracer un polygone en utilisant ses sommets ; il suffit d'écrire :

```
\pspolygon(A)(B)(C)(D)
```

Attention ; les noms des points sont entourés de parenthèses (et pas d'accolades !).
On peut ainsi utiliser tous les outils de remplissage des polygones que permet `PsTricks` ; par exemple :

```
\pspolygon[fillstyle=vlines,hatchcolor=red](A)(B)(C)(D)
```

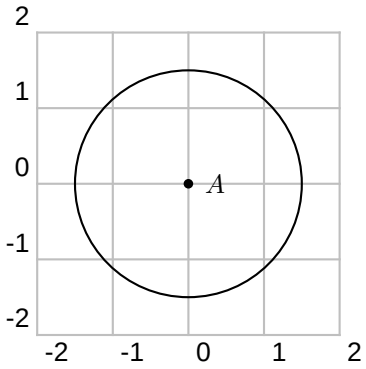
Ligne qui, ajoutée au code du polygone après la définition des points, donne :



13.2.2 D'autres cercles

On a vu comment tracer un cercle de diamètre donné dont on connaît les extrémités avec `pstCircleAB`, et on a vu comment tracer un cercle de centre un point donné et passant par un autre point avec `\pstCircleOA`.
Voici quatre exemples donnant d'autres façons de définir des cercles.

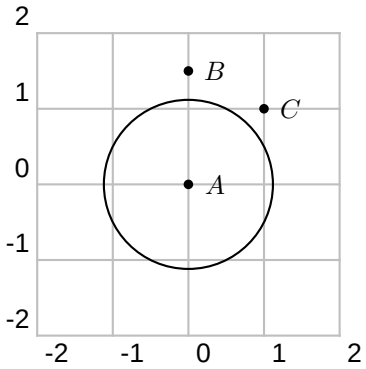
Cercle de centre A et de rayon k




```
\pstGeonode(0,0){A}
\pstCircleOA[Radius=\pstDistVal{1.5}]{A}{}
```

On définit le point A.
Comme le cercle est défini par son centre, c'est `\pstCircleOA` qu'il faut utiliser. Le rayon est défini par `Radius` et sa valeur est définie par `\pstDistVal`.
On laisse alors le second paramètre de `\pstCircleOA` vide.

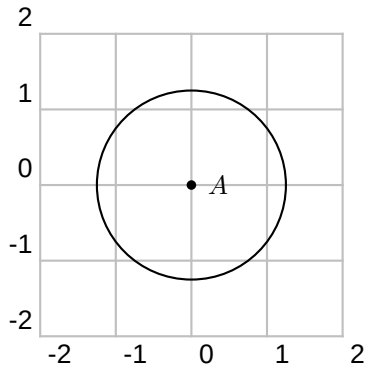
Cercle de centre A et de rayon BC




```
\pstGeonode(0,0){A}(0,1.5){B}(1,1){C}
\pstCircleOA[Radius=\pstDistAB{B}{C}]{A}{}
```

On définit les points A, B et C.
Le cercle est défini par son centre, donc on utilise `\pstCircleOA`.
Le rayon est défini par `Radius` et sa valeur est la distance BC, définie par `\pstDistAB`.
Là aussi le second paramètre de `\pstCircleOA` sera laissé vide.

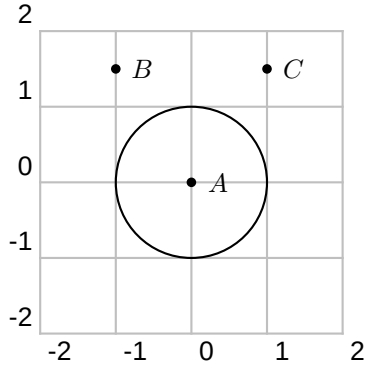
Cercle de centre A et de diamètre k




```
\pstGeonode(0,0){A}
\pstCircleOA[Diameter=\pstDistVal{2.5}]{A}{}
```

On définit le point A.
Comme le cercle est défini par son centre, c'est `\pstCircleOA` qu'il faut utiliser. Le diamètre est défini par `Diameter` et sa valeur est définie par `\pstDistVal`.
Le second paramètre de `\pstCircleOA` est encore vide.

Cercle de centre A et de diamètre BC




```
\pstGeonode(0,0){A}(-1,1.5){B}(1,1.5){C}
\pstCircleOA[Diameter=\pstDistAB{B}{C}]{A}{}
```

On définit les points A, B et C.
Le cercle est défini par son centre, donc on utilise `\pstCircleOA`.
Le diamètre est défini par `Diameter` et sa valeur est la distance BC, définie par `\pstDistAB`.
Et le second paramètre de `\pstCircleOA` sera encore laissé vide.