Benjamin COLLAS
benjamin.collas@telaxo.com

Mi-101 **Mathematica** 2004-2005

Leçon 4 : Le graphisme

Objectifs :
- utiliser les différentes primitives graphiques
- utiliser la représentation complexe du plan affine

Grâce à *Mathematica*, vous pouvez définir des objets (points, lignes, cercles..) en précisant leurs caractéristiques (centre et rayon pour un cercle, coordonnées pour un point). Vous construisez ainsi une liste d'objets dont vous demandez l'affichage par les commandes *Show* et *Graphics* combinées.

Travail sur les points

La commande *Point[{x,y}]* permet de transformer le couple *{x,y}* en un point du plan.
Attention : ce n'est pas un objet graphique.
Attention (bis) : cette commande ne demande pas l'affichage.
Cette commande n'étant pas listable, on ne peut pas l'appliquer directement sur une liste de coordonnées que l'on veut transformer en points.

```
points = Table[Point[{Cos[i], Sin[j]}], {i, 1, 4, .2}, {j, 1, 4, .2}];
```

La transformation en objet graphique s'obtient par la commande *Graphics*, tandis que l'affichage se fait en utilisant la commande *Show*.

```
ptsGraph = Graphics[points];

Show[ptsGraph];
```

■ Exercice :

En utilisant *Random*, construisez plusieurs nuages de points aléatoires de deux manières différentes :
- en utilisant une manière similaire à celle-ci dessus.
- en créant une liste de coordonnées et en appliquant la fonction *Point* à chacun des éléments de la liste.

Vous pouvez aussi modifier la taille et la couleur de ces points : il suffit de faire précéder les points en question par une commande de couleur (*Hue* ou *RGBColor*) et de *PointSize*.


```
ptsColores = Table[{Hue[Random[Real]], PointSize[Random[Integer, 3] / 100],
    Point[{Random[Real], Random[Real]}]}, {i, 1, 150}];

ptsColoresOk = Flatten[ptsColores, 1];

Show[Graphics[ptsColoresOk]];
```

■ Exercice :

Faut-il obligatoirement utiliser *Flatten* ?
Générez un ensemble de points colorés de différentes tailles, généré par les coordonnées *{Cos[i]+k,Sin[i]+j}* pour $i \in [0,2\pi]$ avec un pas de *0.3*, *j* et $k \in [-0.5,0.5]$ avec un pas de *0.1*.

■ Les points et les complexes

En mathématiques, il est souvent plus aisé d'exprimer certaines transformations du plan par des fonctions complexes. Aussi, les points sont ils tout naturellement codés sous la forme de points complexes. Ceci nécessite une fonction *CtoR* qui permet de transformer un point complexe en coordonnées du plan réel.

```
CtoR[z_] := {Re[z], Im[z]}
```

■ Exercice :

Affichez un nuage de points aléatoires en utilisant *CtoR*.

■ Exercice :

Coder sous forme de complexe les transformations affines usuelles suivantes : rotation, translation et similitude. Appliquer chacune d'elle à *ptsColores* et stocker le résultat dans une autre variable. Afficher les deux graphes obtenus dans un même graphique.

Travail sur les lignes et les polygones

Pour tracer une ligne , on utilise la commande *Line[{A,B,C,D}]* qui tracera une ligne brisée entre les points A, B, C et D exprimés sous forme de coordonnées réelles.
Là encore il est souvent plus simple de coder les points sous forme complexe avant de les retraduire en coordonnées réelles pour l'affichage.

Voici un exemple (sans les complexes) qui trace une enveloppe de lignes :

```
lignes = Table[Line[{{0, 10 - i}, {i, 0}}], {i, 1, 10}];

graphLignes = Show[Graphics[lignes], AspectRatio → Automatic, Axes → True];
```

■ Polygone

Voici un exemple qui trace un polygone régulier à 6 côtés :

```
ptsC = Table[Exp[I * r], {r, 0, 2 * Pi, 2 Pi / 6}];
ptsR = Map[CtoR, ptsC];

graphPoly = Show[Graphics[Line[ptsR]], AspectRatio → Automatic, Axes → False];
```

On peut aussi utiliser la primitive *Rectangle* qui repère un rectangle par les coordonnées de ses sommets inférieur gauche et supérieur droit.

```
malevich = {RGBColor[.8, .2, .2], Rectangle[{-4.3, 1.3}, {9, 2.5}],
   RGBColor[.95, .3, 0], Rectangle[{-6, 2}, {7.7, 3}], RGBColor[0, .2, 1],
   Rectangle[{-.5, -15}, {.3, 7}], RGBColor[.9, .8, 0], Rectangle[{0, -10}, {.8, 10}]};

malevichGr = Show[Graphics[malevich], AspectRatio → Automatic];
```

■ Ligne et rotation

■ Image par une rotation d'un élément graphique de base

On crée tout d'abord un élément graphique de base.

```
lignAlea = Table[Random[Real], {i, 1, 5}, {j, 1, 2}];

Show[Graphics[Line[lignAlea]], AspectRatio → Automatic];
```

Appliquons lui maintenant une rotation.

```
rot[theta_][{x_, y_}] := CtoR[(x + I * y) * E^(I * theta)]

lignRot = Map[rot[Pi / 1.2], lignAlea];
Show[Graphics[{Map[Line, {lignAlea, lignRot}], Point[{0, 0}]}]];
```

■ Images successives par une rotation

Utilisons une nouvelle commande *NestList.*

```
ptA = Table[Random[Real], {2}];

grTot = Line[NestList[rot[Pi / 2], ptA, 4]];

Show[Graphics[grTot], AspectRatio → Automatic];
```

Exercice :
Quelle est l'utilisation de *NestList* ?
Quelle transformation géométrique permet-elle d'obtenir ici ?
Tracer un polygone convexe à 17 cotés.

■ Autre solution : FoldList

Il s'agit du même principe, mais on utilise dès le départ les nombres complexes ainsi qu'une nouvelle commande *FoldList.*

```
Attributes[CtoR] = {Listable};

ptsC1 = Table[Random[Complex], {i, 1, 5}];

pts = FoldList[Times, ptsC1, Table[E^(I * Pi / 6), {j, 1, 12}]];

ptsR1 = CtoR[pts];

Show[Graphics[Map[Line, ptsR]], AspectRatio → Automatic];
```

Exercice :
Quelle est l'utilisation de *FoldList* ?
Quelle transformation géométrique permet-elle d'obtenir ici ?

Quelques étoiles

Voici quelques modèles qui utilisent des notions déjà rencontrées. Etudiez les en recréant vos propres exemples.

■ Premier modèle

Création des points en complexe :

```
ptsC2 = ptsC / 5 * Exp[I * 2 * Pi / 12];
```

Transformation en coordonnées réelles :

```
ptsR2 = Map[CtoR, ptsC2];
```

Construction de l'étoile proprement dite en utilisant les points définis précédemment :

```
etoile = Flatten[Table[{ptsR[[i]], ptsR2[[i]]}, {i, 1, Length[ptsC]}], 1];

graphEtoile1 = Show[Graphics[Line[etoile]], AspectRatio → Automatic, Axes → False];
```

■ Second modèle [bonus]

Un autre type d'étoile basé sur le même processus de contruction.
Attention : Etudiez la fonction *Mod* en s'assurant de bien comprendre son action sur quelques exemples.

```
ptsC = Table[Exp[I * r], {r, 0, 2 * Pi - .1, 2 * Pi / 5}];

ptsR = Map[CtoR, ptsC];

etoile2 = Flatten[
   Table[{ptsR[[Mod[i, Length[ptsR - 1]] + 1]]}, {i, 1, 2 * Length[ptsR] + 1, 2}], 1];
```

```
graphEtoile2 = Show[Graphics[Line[etoile2]], AspectRatio → Automatic, Axes → False];

ptsC = Table[Exp[I * r], {r, 0, 2 * Pi - .1, 2 * Pi / 11}];

ptsR = Map[CtoR, ptsC];

etoile2 = Flatten[
   Table[{ptsR[[Mod[i, Length[ptsR - 1]] + 1]]}, {i, 1, 5 * Length[ptsR] + 1, 5}], 1];

graphEtoile3 = Show[Graphics[Line[etoile2]], AspectRatio → Automatic, Axes → False];
```

Faisceaux de cercles

La commande pour tracer un cercle est la commande *Circle[{x,y},r]* qui définit un cercle de centre *{x,y}* et de rayon *r*. Voici quelques exemples :

```
Show[Graphics[Table[Circle[{0, 0}, r], {r, 0, 1, .2}]], AspectRatio → Automatic];

Show[Graphics[Table[Circle[{r, 0}, 1], {r, 0, 1, .2}]], AspectRatio → Automatic];
```

On peut aussi donner des noms aux dessins tracés :

```
gr1 = Show[Graphics[Table[Circle[{r, 0}, 1 - r], {r, 0, 1, .2}]],
   AspectRatio → Automatic, Axes → False];

gr2 = Show[Graphics[Table[Circle[{r, 0}, r - 1], {r, 1, 2, .2}]],
   AspectRatio → Automatic, Axes → False];
```

Ceci permet ensuite de les recombiner lors d'un même affichage :

```
Show[{gr1, gr2}];
```

Présentation multi-graphiques

Lorsque vous désirez afficher plusieurs objets graphiques côte-à-côte après leur création, vous pouvez utiliser la commande *GraphicsArray* accompagnée de *Show*. Celle-ci permet l'affichage d'une liste d'objets graphiques sous forme de tableau.

Remarque : vous pouvez mèler objets 3D et objets 2D.

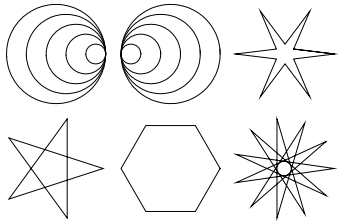
```
listeGraph = {gr1, gr2, graphEtoile1, graphEtoile2, graphPoly, graphEtoile3};

Show[GraphicsArray[listeGraph]];
```

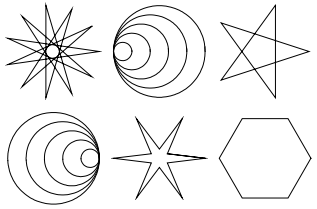
■ Exercice :

A l'aide des fonctions *Partition, Transpose* et *RotateLeft*, générez les affichages suivants :

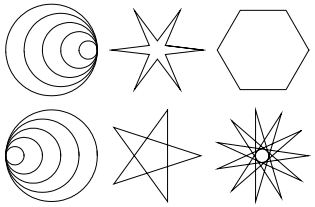
Affichage 1 :



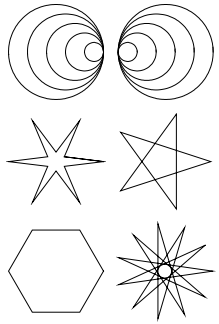
Affichage 2 :



Affichage 3 :



Affichage 4 :



Primitives dans l'espace [Bonus]

Les primitives qui ont un sens en dimension 3 (comme les lignes ou les points), sont réutilisables en utilisant trois coordonnées au lieu de deux.
La fonction qui commande l'affichage devient alors en toute logique *Graphics3D.*

```
pts3D =
  Table[Point[{Random[Real, {-6, 6}], Random[Real, {-6, 6}], Random[Real, {-6, 6}]}],
   {i, 1, 200}];

lignes3D = Line[Table[{Random[Real, {-6, 6}],
     Random[Real, {-6, 6}], Random[Real, {-6, 6}]}, {i, 1, 50}]];

Show[Graphics3D[{pts3D, lignes3D}]];
```

■ La bibliothèque Polyhedra

On peut ajouter des primitives 3D supplémentaires *via* la bibliothèque *Polyhedra*.

La commande suivante permet de charger en mémoire le contenu de la bibliothèque :

Attention aux pédéfinitions : si vous avez défini, avant de charger la bibliothèque, une fonction qui se trouve dans celle-ci, c'est votre première définition qui sera prise en compte et non celle du noyau.

```
<<Graphics`Polyhedra`
```

■ Exemple du cube

Les différentes primitives de la bibliothèque sont disponibles grâce à la commande *Polyhedron*. Le premier argument spécifie quelle primitive est requise, les arguments restants venant caractériser l'objet géométrique.

Pour le cube, on spécifie ainsi son centre et son coté.

```
cubeCube = Table[Polyhedron[Cube, {i, j, k}, .4], {i, 1, 3}, {j, 1, 3}, {k, 1, 3}];

cubeCubeGr = Show[cubeCube];
```

Se référer à l'aide *Add-on & Link/Polyhedra* pour la liste complète des primitives.