Guide d'utilisation de « LangageGraphique » et de la barre d'outils « Progiciels » dans *Calc* (le tableur de la suite *Open Office*)

André Boileau
Section didactique, Département de mathématiques, UQAM

Présentation de LangageGraphique

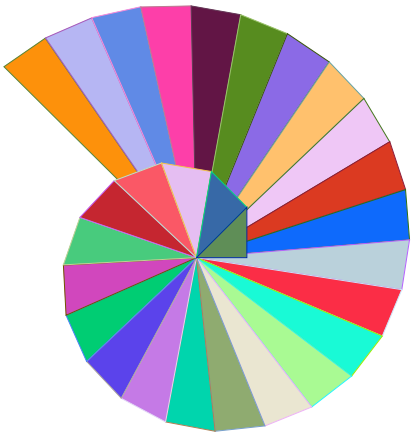
« ***LangageGraphique*** » est le nom donné à un ensemble de macros pour *Calc*, le tableur de la suite *Open Office*, qui permet de réaliser des figures mathématiques à partir de descriptions textuelles. L'utilisation de « ***LangageGraphique*** » passe par une barre d'outil

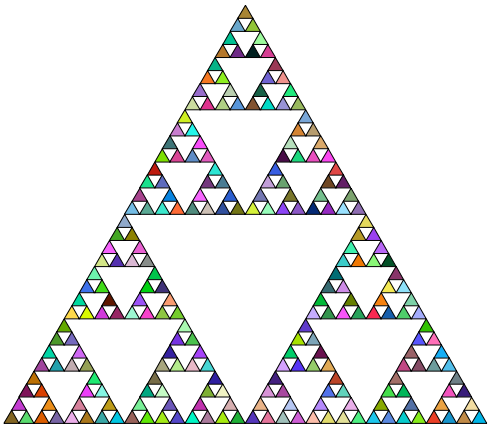
Agrandissement de la barre d'outils *Progiciels* .

permettant d'accéder à divers outils utiles pour créer des objets mathématiques dans *Calc*[1]. De gauche à droite, nous retrouvons les icônes permettant

- d'ouvrir l'éditeur d'équations
- d'insérer une matrice (après en avoir spécifié les dimensions)
- d'ouvrir l'éditeur de programmes *Libre Office Basic*
- de créer un dessin « ***LangageGraphique*** » dans un document *Calc*
- d'insérer une zone de texte (transparente et sans bordures) dans un document *Calc*
- d'effacer les dessins dans un document *Calc*.

Avec la version initiale (non modifiée) de « ***LangageGraphique*** », vous pouvez déclencher une démonstration de ses possibilités en cliquant sur le « bonhomme sourire » de la barre d'outils *Progiciels*. Par la suite, vous pourrez modifier l'effet de ce bouton pour qu'il trace vos propres figures, que vous pourrez redimensionner pour les copier puis les coller dans un document texte, comme illustré ci-dessous.



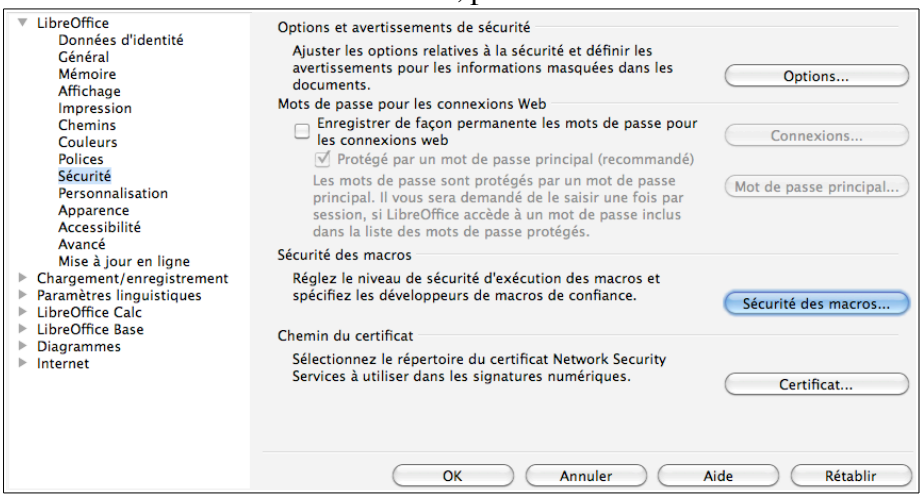


[1] Dans la version « installée », les quatre icônes de gauche sont aussi disponibles dans le traitement de textes *Writer*. Mais les figures se tracent alors dans une feuille de calcul créée spécialement pour l'occasion.

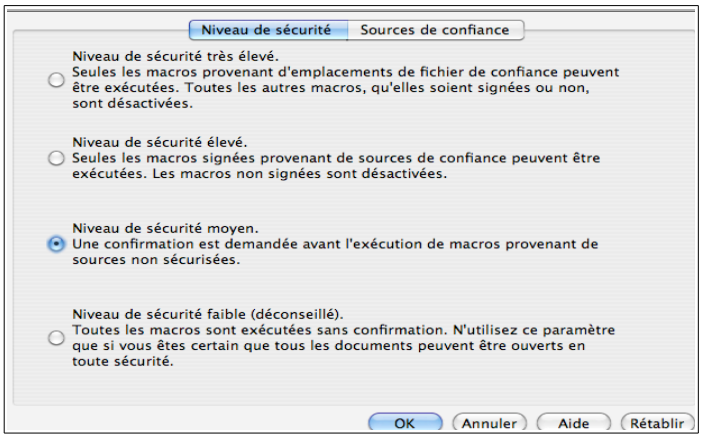
Utilisation de « *LangageGraphique* »

Pour utiliser « ***LangageGraphique*** », il suffit d'ouvrir le fichier[2] *LangageGraphique.ods*. Comme ce document contient des macros, il faudra

- avant l'ouverture du document, avoir choisi un niveau de sécurité « moyen » pour l'ouverture des macros. Pour se faire, il suffit d'aller dans les préférences de *Libre Office* et de choisir successivement
  - l'item « Sécurité » de « LibreOffice », puis le bouton « Sécurité des macros... »



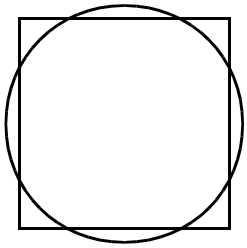
  - le « Niveau de sécurité moyen », puis confirmer le tout (boutons « Ok »)



- à l'ouverture du document, permettre l'utilisation des macros, par un clic sur le bouton « Activer les macros ».

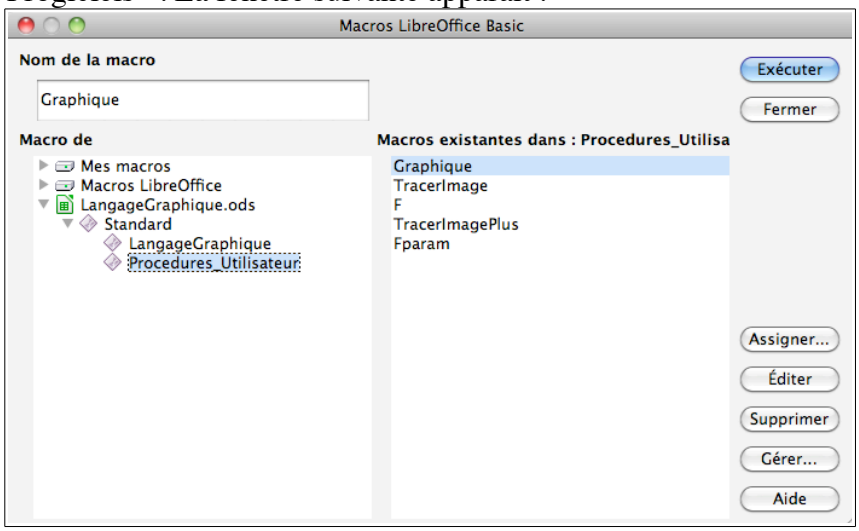
[2] Les utilisateurs plus avancés peuvent aussi choisir d'installer « ***LangageGraphique*** » directement dans *LibreOffice*, de façon à pouvoir utiliser « ***LangageGraphique*** » en tout temps, sans avoir recours au fichier *LangageGraphique.ods*.

Premiers pas avec « LangageGraphique »



Pour illustrer le fonctionnement de « LangageGraphique », nous allons l'utiliser pour tracer le dessin ci-contre, qui montre un carré et un cercle de même aire. Si *r* est la longueur du rayon du cercle, la longueur du côté *c* du carré devra donc être de $r\sqrt{\pi}$.

Pour entrer les instructions nécessaires, nous devons tout d'abord accéder à l'Éditeur OpenOffice Basic : ceci se fait par un clic sur l'icône  de la barre d'outils « Progiciels ». La fenêtre suivante apparaît :



Vous devrez peut-être cliquer sur les triangles à la gauche des noms de la colonne de gauche, puis ensuite sur le nom « Procedures_Utilisateur », pour arriver au résultat ci-dessus. Vous cliquez ensuite sur le bouton « Éditer » pour obtenir une fenêtre d'édition contenant un texte débutant par les lignes suivantes

```
Sub Graphique()  'Placez vos commandes graphiques dans cette procédure
    Demo
End Sub
```

Nous remarquons que la procédure « Graphique » comporte une seule instruction, « Demo », qui a provoqué tout à l'heure l'exécution de la démonstration. Nous devons donc remplacer cette instruction par des instructions pour tracer un cercle et un carré de même aire. Si nous choisissons que notre cercle soit centré à l'origine et de rayon 100, nous obtenons les instructions suivantes :

```
Sub Graphique()  'Placez vos commandes graphiques dans cette procédure
   r = 80
   c = r * Sqr(pi)
  Cercle 0 , 0 , r
  Rectangle  - c/2 ,  - c/2 ,   c/2 ,   c/2
End Sub
```

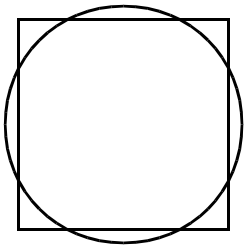
Notons que nous aurions aussi pu taper directement

```
Sub Graphique()
  Cercle  0 , 0 , 80
  Rectangle  -80*Sqr(pi) / 2,  -80*Sqr(pi) / 2 ,  80*Sqr(pi) / 2,  80*Sqr(pi) / 2
End Sub
```

L'important est de donner des valeurs numériques de façon à ce que le cercle et le rectangle puissent bien être tracés. Notons en passant que l'origine du système d'axes utilisé est située à peu près au milieu d'une page *Calc* (lors de l'impression au format lettre) et que l'unité de longueur utilisée est le mm. Mais ceci a relativement peu d'importance puisqu'il nous sera facile de changer ***gestuellement*** la position et les dimensions du graphique ainsi obtenu.

Quand nous avons fini d'entrer nos instructions, nous retournons à notre page *Calc* en cliquant sur celle-ci. On pourrait aussi fermer la fenêtre de l'éditeur Basic, mais il nous faudrait alors la réouvrir à notre prochaine visite.

Pour obtenir (enfin!) la figure désirée, nous devons maintenant faire un **clic** sur le bouton  . Si tout se passe bien, nos efforts seront récompensés par l'obtention de la figure ci-contre, que nous pourrons modifier (par exemple en changeant ses dimensions) avant de la **copier** et de la **coller** dans le document de notre choix. Notez en passant que notre tâche sera facilitée par le fait que « LangageGraphique » regroupe automatiquement tous les éléments tracés, ce qui facilite tant les redimensionnements que les copies.



Si *LibreOffice* nous signale une erreur, nous serons amenés à la ligne où l'erreur a été détectée. Malheureusement, cette ligne ne correspond pas nécessairement à la position de l'erreur initiale, qui est parfois détectée plus tard. Pour l'instant, contentez-vous de vérifier que vous avez bien tapé les instructions exactement comme elles apparaissent ci-dessus, en n'oubliant pas les espaces, virgules et parenthèses. Corrigez au besoin la ou les coquilles, retournez à *Calc* et redemandez le tracé.

Un autre exemple d'utilisation de *LangageGraphique*

Après avoir vu comment réaliser le cercle et le carré de même aire, nous allons nous attacher à réaliser le tracé d'un polygone régulier à 7 côtés parce que ce sera l'occasion d'introduire plusieurs autres possibilités de « LangageGraphique »**.**

Nous devrons donc tracer 7 segments, dont les extrémités correspondront aux 7 sommets du polygone. Dans le cas d'un polygone inscrit dans un cercle centré en (0,0) et de rayon r, disposé de façon telle qu'un des sommets repose sur l'axe des x positifs, on détermine que les coordonnées des sommets seront

$$(x_k, y_k) = \left(r\cos\left(\frac{2\pi}{7}k\right), r\sin\left(\frac{2\pi}{7}k\right)\right)$$, où k est un entier variant de 0 à 6.

Nous pourrions donc réaliser ce polygone via une série de commandes (une pour chaque segment à tracer) comme suit :

Segment $x_0\ y_0\ x_1\ y_1$
Segment $x_1\ y_1\ x_2\ y_2$
...
Segment $x_6\ y_6\ x_0\ y_0$

Mais nous pouvons profiter que nous travaillons dans le cadre d'un langage de programmation disposant de commandes pour réaliser des répétitions, et utiliser plutôt les instructions suivantes :

```
Sub Graphique()
    r = 0   '  r est la mesure du rayon du cercle circonscrit au polygone
    n = 7     '  n est le nombre de côtés du polygone
    For k = 0 To n-1
       Segment r*cos(k*2*pi/n), r*sin(k*2*pi/n), r*cos((k+1)*2*pi/n), r*sin((k+1)*2*pi/n)
    Next k
End Sub
```

Nous avons utilisé ici la structure de contrôle

```
For variable = valeur_initiale To valeur_finale
   liste_d'instructions_pouvant_contenir_la_variable
Next variable
```

qui, successivement,

- exécute la *liste_d'instructions* en donnant à la *variable* la valeur *valeur_initiale*
- exécute la *liste_d'instructions* en donnant à la *variable* la valeur *valeur_initiale* + 1
- exécute la *liste_d'instructions* en donnant à la *variable* la valeur *valeur_initiale* + 2
- ...
- exécute la *liste_d'instructions* en donnant à la *variable* la valeur *valeur_finale* [car *valeur_finale* = *valeur_initiale* +(*valeur_finale* - *valeur_initiale*)].

En se rappelant que $\frac{2\pi}{n}(k+1)$ équivaut à un angle de zéro quand $k = n-1$, on voit donc que l'on obtient des instructions en 5 lignes qui font le même travail que nos 7 instructions initiales. Mais ce n'est pas tout : en changeant la valeur de n, on peut tracer tous les polygones réguliers, quel que soit leur nombre de côtés. Ceci justifie amplement le choix d'utiliser des variables au lieu d'utiliser directement des valeurs, un autre avantage étant de rendre nos instructions plus lisibles.

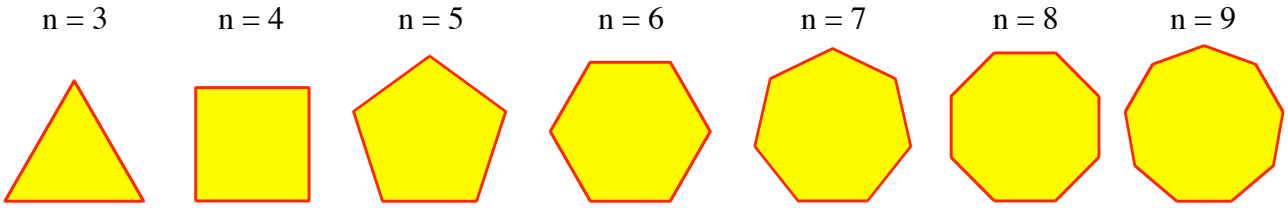
Notons que, jusqu'à présent, nous avons utilisé le paradigme de la *géométrie analytique* pour tracer nos graphiques. Nous pourrions tout aussi bien utiliser le paradigme de la *géométrie de la tortue* cher aux adeptes de Logo, car « LangageGraphique » peut piloter une tortue avec les instructions *Avance*, *Droite*, *FixePosition*, *FixeCap*, *LeveCrayon*, *BaisseCrayon* et *VideEcran*.

Dans ce contexte, nous aurions pu utiliser les instructions

```
Sub Graphique()   ' tout texte suivant une apostrophe est un commentaire
    c = 40     ' c est la mesure d'un côté du polygone
    n = 7     ' n est le nombre de côtés du polygone
    For k = 0 To n-1
       Avance c
       Droite 360/n   '  pour la tortue, les angles sont en degrés
    Next k
End Sub
```

et nous aurions obtenu un polygone régulier à 7 côtés dont l'inclinaison et la mesure des côtés auraient été quelque peu différentes. (Exercice : quelle valeur aurait-on dû donner au côté c pour que le polygone soit inscrit dans un cercle de rayon r?)

Nous pouvons maintenant utiliser une des deux versions précédentes, disons celle utilisant le paradigme de la tortue, pour produire la figure suivante



Cette figure a été réalisée à l'aide des instructions suivantes, exécutées 7 fois en donnant successivement à n des valeurs de 3 à 9.

```
Sub Graphique()
    n = 3      ' n est le nombre de côtés du polygone
    r = 30     ' r est le rayon du cercle circonscrit
    c = 2*r*sin(pi/n)     ' c est la mesure d'un côté du polygone
    CouleurCrayon 255,0,0  ' composante rouge au maximum
    CouleurRemplissage 255, 255, 0 ' composantes rouge et verte au maximum
    DebutRemplir   ' la figure fermée formée par les segments suivants sera remplie
    Droite -90   ' pour que la tortue pointe vers la gauche (base horizontale)
    For k = 0 To n-1
       Avance c
       Droite 360/n
    Next k
    FinRemplir  ' remplir la figure amorcée avec DebutRemplir
End Sub
```

Après chaque exécution, on a copié la figure obtenue pour la coller ensuite dans le présent document. Les annotations « n = ... » ont été réalisées au préalable, en faisant apparaître une zone de texte (via un clic sur le bouton ) que l'on complète et l'on dispose ensuite gestuellement.

Notons au passage le tandem d'instructions **DebutRemplir ... FinRemplir** donnant l'instruction de remplir une figure fermée formée de segments, ainsi que les instructions **CouleurCrayon** et **CouleurRemplissage** permettant de spécifier la couleur des traits et des remplissages en utilisant la codification rouge-vert-bleu pour décrire les couleurs.

Les primitives graphiques de *LangageGraphique*

Après ces quelques exemples d'utilisation de ***LangageGraphique***, voici maintenant une description des principales commandes disponibles. Dans la colonne **Procédure**, vous trouverez la forme générale des commandes avec, le cas échéant, un exemple d'utilisation *en italique*. La colonne **Alt.** donne, le cas échéant, la forme abrégée de la commande : par exemple, on peut utiliser **VE** au lieu de **ViderEcran**. Je vous rappelle que :

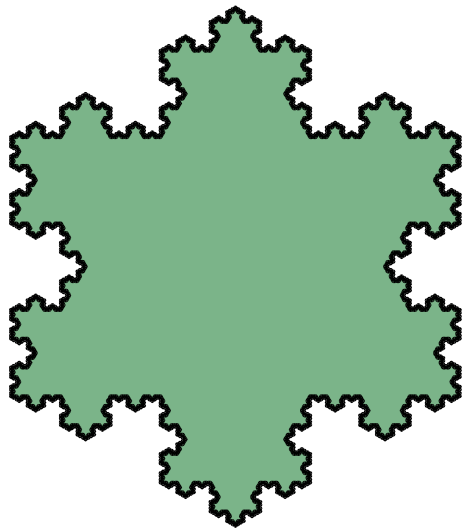
- l'origine du système de coordonnées sous-jacent se trouve approximativement au centre de la page 1
- au départ, la tortue est à l'origine du système de coordonnées et « regarde » vers le haut
- les distances et les coordonnées, qui peuvent être des nombres décimaux (exprimés avec des ***points décimaux*** et non des *virgules décimales*), sont exprimées en mm
- les angles sont exprimés en degrés. (Attention cependant aux fonctions trigonométriques, qui supposent que leurs arguments sont en radians, **cosD** et **sinD** faisant exception.)

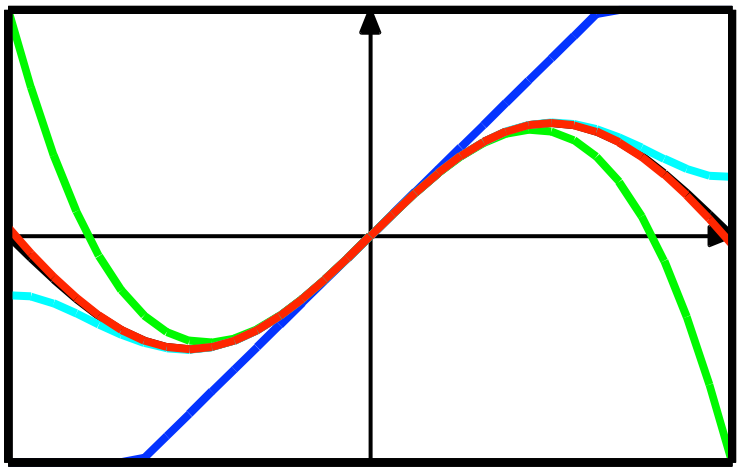
Commandes pour tracer de formes décrites avec des coordonnées

Procédure	Alt.	Commentaires
Segment x1, y1, x2, y2 *Exemple : Segment 0,0,100,50*		Trace un segment joignant deux points dont on donne les coordonnées.
Rectangle x1, y1, x2, y2 *Exemple : Rectangle 0,0,100,50*		Dessine un rectangle (sans remplissage) dont on spécifie les coordonnées d'une diagonale.
RectanglePlein x1, y1, x2, y2 *Exemple : RectanglePlein 0,0,100,50*		Dessine un rectangle rempli avec la couleur de remplissage.
Ellipse x1, y1, x2, y2, Remplir, Angle *Exemple : Ellipse 0,0,100,50,true,45*		Trace l'ellipse déterminée par le rectangle, et la tourne ensuite. Remplir peut prendre les valeurs « true » ou « false ».
Cercle x, y, R *Exemple : Cercle 0,0,100*		Trace le cercle de centre et rayon donnés
Disque x, y, R *Exemple : Disque 0,0,100*		Trace le disque (cercle plein) de centre et rayon donnés
Arc x, y, rayon, depart, arrivee *Exemple : Arc 0,0,100,45,135*		Trace un arc déterminé par un cercle, un angle de départ et un angle d'arrivée (sens +).
Grille x, y, dx, dy, nx, ny *Exemple : Grille 0,0,10,10,20,10*		Dessine une grille à la position (x,y) • nx colonnes de dimension dx • ny lignes de dimension dy

Commandes pour tracer de formes décrites avec des mouvements de la tortue

Procédure	Alt.	Commentaires
Avance distance *Exemple : Avance 30*	Av	Fait avancer la tortue d'un certain nombre de pas (points). Pour reculer, utiliser un nombre de pas négatif.
Droite angle *Exemple : Droite 90*	Dr	Fait tourner la tortue vers la droite d'un certain nombre de degrés. Pour tourner à gauche, utiliser un angle négatif..
FixePosition x, y *Exemple : FixePosition 0,0*	FPos	Fixe la position de la tortue.
FixeCap angle *Exemple : FixeCap 45*	FCap	Fixe le cap de la tortue.
LeveCrayon		Lève le crayon de la tortue. Si la tortue se déplace, elle ne laissera pas de trace.
BaisseCrayon		Baissele crayon de la tortue. Si la tortue se déplace, elle laissera une trace.





Commandes pour modifier les caractéristiques des formes

Procédure	Alt.	Commentaires
ViderEcran	VE	Efface tout le dessin
CouleurCrayon R, V, B *Exemple : CouleurCrayon 0,0,0*	CC	Détermine la couleur du crayon de la tortue. Le paramètres peuvent varier de 0 à 255.
TailleCrayon épaisseur *Exemple : TailleCrayon 2*	TC	Détermine l'épaisseur du crayon de la tortue.
DebutRemplir		Débute un polygone à remplir.
FinRemplir		Fin du polygone à remplir.
CouleurRemplissage R, V, B *Exemple : CouleurRemplissage 255,255,0*	CR	Détermine la couleur des remplissages. Le paramètres peuvent varier de 0 à 255.
Dessous Dessus		L'élément sélectionné est placé *au desssous* ou *au dessus* de tous les autres.
Hasard (min, max) *Exemple : CC Hasard (0,256),0,0*		Fonction retournant un nombre x choisi au hasard tel que min ≤ x < max
Texte chaine, x, y, Police, Taille, Gras, Italique *Exemple : Texte "Bonjour",0,0, "Times",12,true,false*		Dessine une chaîne de caractères à la position et avec les caractéristiques spécifiées.

Commandes pour faire des conversions d'unités

Procédure	Alt.	Commentaires
sinD (x) *Exemple : sinD (45)*		Fonction sinus où l'angle est exprimé en degrés. La fonction sin(x) suppose que x est en radians.
cosD (x) *Exemple : cosD (45)*		Fonction cosinus où l'angle est exprimé en degrés. La fonction cos(x) suppose que x est en radians.
degresAradians (x) *Exemple : degresAradians (45)*		Conversion en radians de x degrés.
radiansAdegres (x) *Exemple : radiansAdegres (π/4)*		Conversion en degrés de x radians.

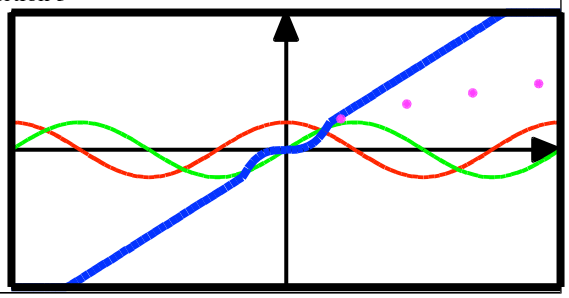
Commandes pour tracer des courbes de fonctions

Procédure	Alt.	Commentaires
RegionsFonctions xmin, xmax, ymin, ymax *Exemple :RegionsFonctions -2*pi, 2*pi, -2, 2*		Spécifie quelle région du plan mathématique sera utilisée pour visualiser les fonctions. Par défaut, on déclare implicitement RegionsFonctions -10, 10, -10, 10
FenetreFonctions gauche,droite,haut,bas *Exemple :FenetresFonctions 0, 300, 0, 200*		Spécifie quelle région du plan informatique (le document Excel) sera utilisée pour représenter les fonctions. Par défaut, on déclare implicitement FenetreFonctions -250, 250, -200, 200
TracerAxes		Noter que les axes tracés se limitent (pour l'instant) à deux segments avec flèches mais sans graduations.
TracerFonction numéro *Exemple TracerFonction 3*		Commande le tracé de la fonction dont le numéro est spécifié.
PasFonctions x *Exemple : PasFonctions 2*		Par défaut, la fonction est calculée à chaque 8 *unité*. On peut augmenter (x<8) ou diminuer (x>8) le nombre de points utilisés.
PointsRelies PointsNonRelies		Tout programme de tracé de fonctions n'évalue la fonction qu'en un nombre fini de points. On peut spécifier si ces points seront réliés entre eux ou pas via ces commandes.

Quelques mots d'explication sont nécessaires pour ce groupe de commandes. On doit tout d'abord spécifier les fonctions que nous voulons tracer. Ceci se fait dans la fonction **F**, qui suit immédiatement la procédure **Graphique**. On doit entrer la définition de la *k* ième fonction ($1 \le k \le 10$) immédiatement après la déclaration **Case *k*** correspondante et, surtout, ne pas modifier le reste de la fonction. Dans l'exemple ci-dessous, on peut voir (écrit en *italique*) qu'on a défini les fonctions 1 à 4.

```
Function F (n, x)
            F = "ND"
            On Error GoTo Erreur
            Select Case n
               Case 1
                  F = Cos(x)
               Case 2
                  F = Sin(x)
               Case 3
                  F = Sqr(x) ' Fonction racine carrée
               Case 4
                  If Abs(x) < 1 Then
                     F = x ^ 3
                  Else:
                     F = x
                  End If
               Case 5
               Case 6
               Case 7
               Case 8
               Case 9
               Case 10
               Case Else
            End Select
         Erreur:
End Function
```

```
Sub Graphique()
   RegionFonctions -2 * pi, 2 * pi, -5, 5
   ' FenetreFonctions -50, 50, -25, 25 déclarée implicitement
   TracerAxes ' Après les commandes RegionFonctions
              '  et FenetreFonctions (ici implicite)
   CouleurCrayon 255, 0, 0
   TracerFonction 1
   CouleurCrayon 0, 255, 0
   TracerFonction 2
   CouleurCrayon 0, 0, 255
   TailleCrayon 2
   TracerFonction 4
   PointsNonRelies
   PasFonctions 12
   TailleCrayon 1
   CouleurCrayon 255, 0, 255
   CouleurRemplissage 255, 0, 255
   TracerFonction 3
End Sub
```



Notez en passant la quatrième fonction, dont la définition utilise la structure conditionnelle

If ... Then ... Else ... End If.

Puis nous entrons, comme précédemment, dans la procédure **Graphique**, les commandes que nous souhaitons voir exécuter. Notons ici que l'ordre des commandes est important : par exemple, on ne peut tracer correctement les axes tant que la région et la fenêtre utilisées n'est pas connue. De même, on commande le tracé de la fonction 4 après celui de la fonction 2 : en cas d'intersection, la couleur de la fonction 4 prévaudra donc. Remarquons que cette lourdeur de notation s'accompagne d'une certaine versatilité : si le cœur nous en dit, on peut tracer, sur un même graphique, la même fonction représentée dans des régions différentes (et donc possiblement avec des échelles différentes). Veuillez noter que les choix effectués via les commandes « FenetreFonctions » et « PasFonctions » on un impact sur le temps nécessaire pour obtenir nos graphes cartésiens, tout particulièrement sur les machines plus lentes.

En guise de conclusion

Disons-le clairement : ***LangageGraphique*** n'est pas un environnement graphique qui est toujours optimal. Si vous avez la chance d'avoir accès à des logiciels tels *GeoGebra*, *Maple*, *POV-Ray*, ou d'autres logiciels mathématiques avec possibilités graphiques, il arrivera souvent que vous pourrez obtenir *plus simplement* de merveilleux graphiques en choisissant l'outil le plus approprié.

Rappelons simplement les caractéristiques qui font la force de ***LangageGraphique*** et qui peuvent nous guider sur les circonstances où il est avantageux de l'employer.

1. Il est *libre* et *gratuit*.
   En fait, il est placé sous licence publique générale GNU (GPL)[3]. Ceci permet éventuellement de l'*adapter à ses besoins*, que ce soit en le modifiant à sa guise ou en lui ajoutant des possibilités qui ne s'y trouvent pas actuellement.
2. Il produit des *dessins vectoriels*.
   Rappelons que, contrairement aux dessins matriciels qui sont essentiellement composés d'une grille rectangulaire de points de couleurs, les dessins *vectoriels* « retiennent » la description des objets dont ils sont composés. Ceci permet de les modifier de façon précise, que ce soit pour les redimensionner, leur appliquer une transformation, ou les rendre à une résolution différente (lors d'une impression, par exemple).
   Notons aussi que, bien que plusieurs logiciels tels *Cabri-géomètre* et *Maple* produisent eux-aussi des dessins vectoriels, ceux-ci subissent parfois certaines « pertes » lors d'un copier-coller dans *Writer*. Ce n'est pas le cas avec ***LangageGraphique*** puisqu'on demeure toujours dans *LibreOffice*.
3. ***LangageGraphique*** combine une approche à la fois ***gestuelle*** et ***textuelle*** dans la production de graphiques.
   Ceci n'est pas fréquent. *Cabri-géomètre*, par exemple, est presque exclusivement gestuel (la calculatrice étant une exception notable)[4]. De son côté, *Maple* est presque exclusivement textuel (avec, comme exception intéressante, la possibilité de changer gestuellement de point de vue pour observer un graphique à trois dimensions).
   LangageGraphique, de son côté, permet de produire *textuellement* des graphiques pour ensuite leur apporter *gestuellement* des changements qui peuvent être considérables (transformations, groupements et dissociations, alignements divers, ajouts d'éléments, etc.).
4. ***LangageGraphique*** permet de produire des graphiques via une *programmation*.
   C'est aussi vrai de *Maple,* de *GeoGebra* et de *POV-Ray*, mais cette caractéristique manque cruellement dans *Cabri-géomètre*, qui ne permet ni les constructions conditionnelles générales, ni les constructions itérées automatiquement. Nous admettons cependant que c'est une possibilité assez intimidante pour plusieurs.

[3] Voir http://fr.wikipedia.org/wiki/Licence_publique_générale_GNU.
[4] Notons cependant que *GeoGebra* réalise un équilibre remarquable entre les approches gestuelle et textuelle.