Initiation à VBA pour Excel

Philippe Bernard
Ingénierie Economique et Financière
Université Paris-Dauphine

Septembre 2008

Chapitre 1

Références

- F. Riva *Applications financières sous Excel en Visual Basic*, Economica 2ème édition
  (ouvrage qui en 200 pages réussit la performance de présenter de manière très complète le langage VBA, puis de faire des applications à la gestion de portefeuille, à l'évaluation des options, à l'assurance de portefeuille et à l'efficience des marchés - les notes ci-dessous suivent souvent très étroitement la présentation de VBA de cet ouvrage)
- pour d'autres ouvrages ou références consultez les pages ressources en finance et en économie du master Ingénierie Economique et Financière

  http ://www.dauphine.fr/dfrea/master_ace/plan.html
- parmi les ouvrages très complets consacrés à VBA sous Excel, B. Jelen & T. Syrstad *VBA and macros for Microsoft Excel* (ou J. Walkenbach *VBA pour Excel 2003*)
- On peut se référer à l'ouvrage de C. Albright *VBA for modelers*, Thomson, 2nd édition

Chapitre 2

VBA un langage objet

Depuis *Excel 97*, un éditeur Visual Basic réside sous Excel et permet d'écrires des programmes complexes. En fait, cette propriété est vraie pour toutes les applications (*Word*, *Excel*, *Power Point*, *Access*) de *Microsoft Office*. *VBA*, *Visual Basic for Applications*, le langage utilisé peut être vu comme l'application d'un langage, *Visual Basic*, à un ensemble d'applications, une collection d'objets, qu'il manie. Il ne faut pas confondre a priori le langage *Visual Basic*, qui est un langage complexe opéramment indépendamment, de *VBA* en général, de *VBA pour Excel* en particulier.

2.1 Le VBE

Pour activer l'éditeur de Visual Basic depuis Excel, dans la barre de menu Excel :

Affichage → Barre d'outils → VisualBasic

et on clique ensuite sur l'icône Visual Basic Editor (*VBE*). La fenêtre apparaît alors par défaut disposée comme dans la figure 2.2.

On peut faire apparaître la fenêtre des codes en cliquant bouton droit sur This Workbook, situé dans la fenêtre Explorateur de projets (en haut à gauche), puis en cliquant en choisissant dans les menus Insertion puis Module. Dans la fenêtre de code (en haut à droite), on peut écrire le code mais aussi des commentaires. Ceux-ci sont le texte qui apparaît à droite d'un guillemet simple ′ .

Une organisation classique de la fenêtre de code du VBE sera de spécifier tout d'abord les options (par exemple ici l'option d'expliciter la nature de toutes les variables utilisées, la spécification du type de fonctions (procédure ou fonction proprement dite), la déclaration des variables à l'aide de la variable *dim*. Ci dessous on construit une fonction simpliste calculant le carré

FIG. 2.1 – La fenêtre du VBA Basic Editor (VBE)

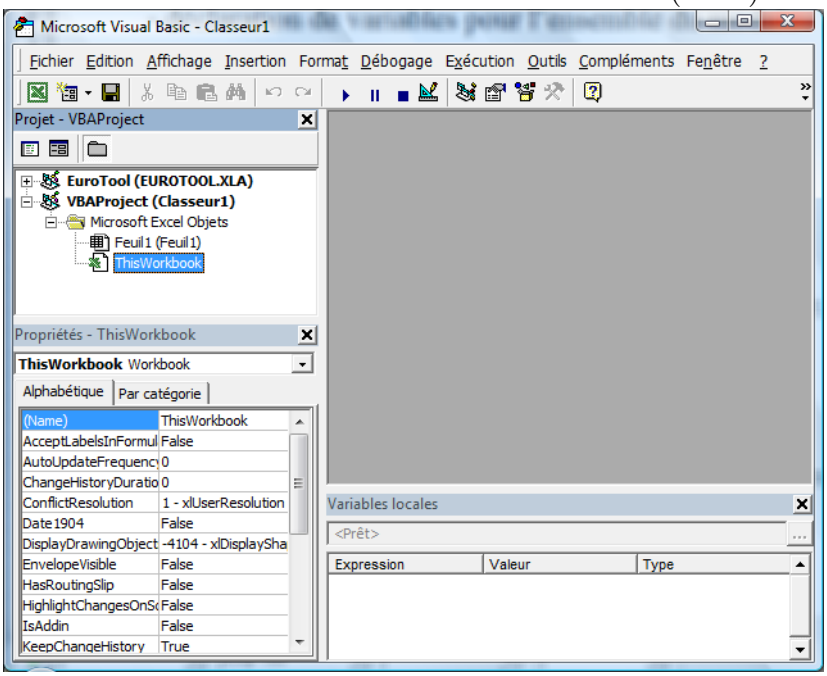
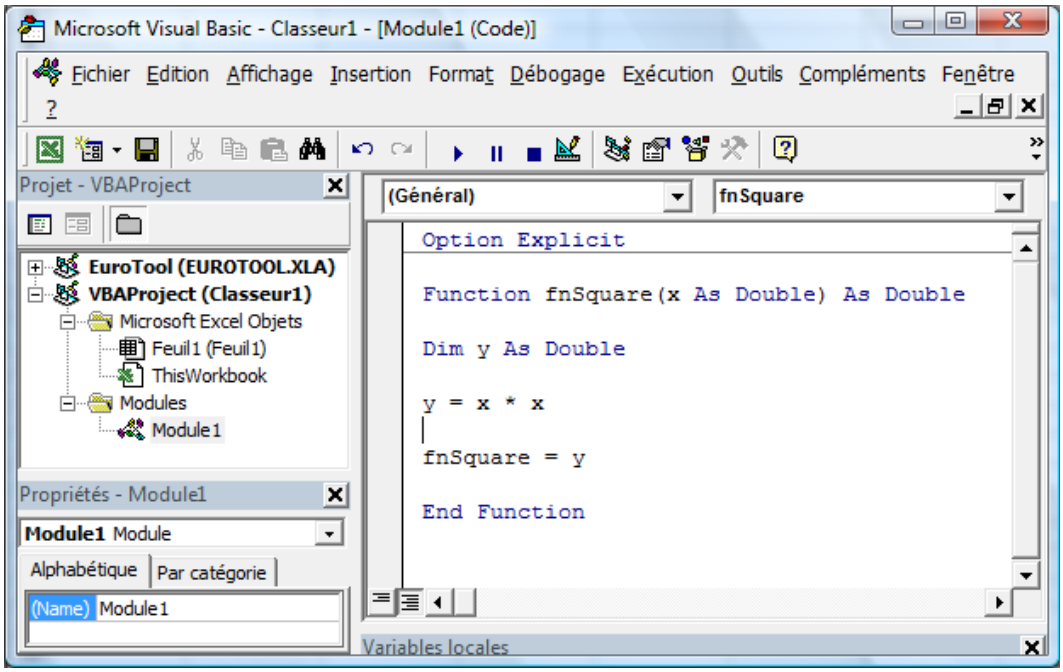


FIG. 2.2 – La fenêtre du Visual Basic Editor (VBE)

FIG. 2.3 – L'état du VBE pour la fonction simple d'élévation au carré

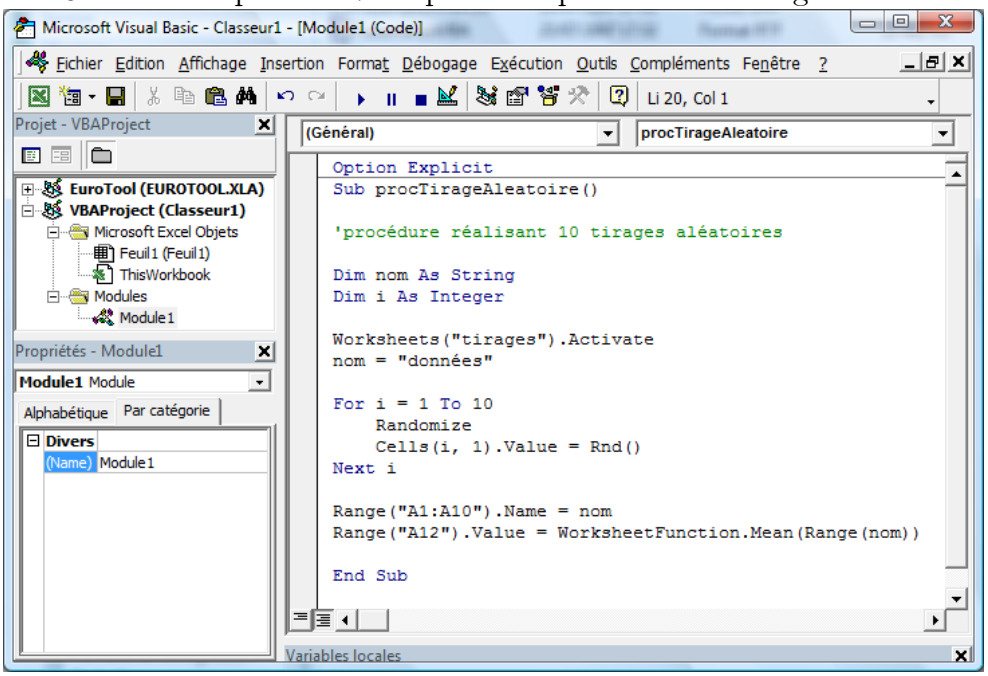


d'un nombre :

```
Option Explicit
Function fnSquare(x As Double) As Double

Dim y As Double

y=x*x

fnSquare=y

End Function
```

Dim y As Double est simplement la déclaration que la variable est de type Double, c'est-à-dire un réel. De même l'argument de la fonction *fnSquare*, *x*, est également un réel comme le souligne la déclaration *x As Double*. Le résultat de la fonction est lui aussi déclaré étant un réel par la syntaxe *Function fonction As Double*. La déclaration de la nature des variables, des arguments des fonctions ou du résultat de celles-ci n'est nullement obligatoire (en l'absence de l'option explicit) mais est généralement conseillé.

FIG. 2.4 – L'aspect du VBE pour une procédure de tirage aléatoire.



Un autre exemple de programme est celui de la procédure représentée sur la figure 2.4 :

L'option explicit est encore une fois spécifiée. La première ligne, après le trait horizon (automatiquement introduit par VBA) spécifie la nature de la fonction (une procédure de type Sub) et son nom (procTitrageAléatoire). La ligne suivante est en vert car elle constitue un commentaire (étant introduite par un guillemet). Puis viennent la déclaration des variables :

- la première variable, appelée *nom*, est une chaîne de caractères (et donc son type est *string*) ;
- la seconde variable, appelée *i*, est un entier naturel.

Puis on fait pointer l'ordinateur vers une feuille appelée *tirage* que l'on désigne comme étant la feuille *active*, c'est-à-dire celle que l'ordinateur doit utiliser par défaut. A la variable *nom* est affectée ensuite la chaîne de caractères *tirages* (la nature de chaîne de caractères étant attestée par les guillemets). On procède ensuite aux 10 tirages aléatoires (par l'usage de la boucle *For ... Next*, en utilisant la fonction aléatoire *Rnd*) et en stockant chaque résultat dans la cellule de la ligne i et de la 1ère colonne (*Cells(i,1)*). Puis à la plage (*Range*) ainsi définie, on affecte le nom contenu par la variable *nom* par l'écriture :

$$Range(\text{“}A1 : A10\text{”}).Name = nom$$

En utilisant la fonction Excel (*WorksheetFunction*) de la moyenne (*Average*), on calcule la valeur moyenne des tirages aléatoires stockées dans la plage *nom*, valeur que l'on affecte à la cellule de la ligne 12, colonne 1 :

$$Range(\text{“}A12\text{”}).Value = WorksheetFunction.Average(Range(nom))$$

Pour enclencher la procédure, il suffit alors de cliquer sur le premier bouton (triangle) du sous-menu :

tandis que le second bouton (double traits) est le bouton d'arrêt, le carré le bouton de réinitialisation.

2.2 Les objets de Visual Basic

VBA est un langage orienté objet. Son code ne part donc pas comme dans les langages procéduraux des actions (par exemple l'impression) pour pointer vers les objets (par exemple une équation). Au contraire, ce langage définit d'abord des *objets* pour ensuite soit pointer vers certaines de leurs *propriétés*, soit leurs appliquer des *actions*. Ainsi l'on désigne l'action de conduire une voiture, on n'écrira pas conduire(voiture), comme par exemple dans un langage comme le basic, mais on écrira voiture.conduire où voiture est l'objet auquel s'applique l'action de conduire. De même si l'on désire obtenir la propriété de la voiture qu'est sa couleur on écrira en VBA voiture.couleur.

Objets, *propriétés*, *actions* sont les principaux éléments que l'on manie dans ce langage.

2.2.1 Les objets et les collections

Le premier objet est Excel lui-même (appelé *Application*) puis viennent différents objets souvent organisés en collection et notamment :

Objet	**Nom**	**Collection**
fichier	workbook	workbooks
feuille de travail	worksheet	worksheets
feuille graphique	chart	charts
plage (de cellules)	range	
cellule	cell	cells
fonctions Excel	worksheetFunction	
macro complémentaire	addIn	addIns

Remarque 1 *Visual Basic ne fait pas de différence entre minuscule et majuscule. Les noms précédents peuvent donc être écrits de multiples manières selon que l'on recourt ou non aux majuscules.*

Une *collection* se définit comme un ensemble d'objets possédant les mêmes caractéristiques : la collection des fichiers (*workbooks*), la collection des feuilles de travail (*worksheets*), la collection des graphiques (*charts*), etc. La règle (comportant des exceptions) est que les collections se distinguent de leurs éléments par le *s* par lequel elles se terminent. L'élément d'une collection peut être notamment identifiée soit par un nombre ou par un nom. Ainsi les feuilles de travail sont à la fois numérotées et ont un nom. Si l'on veut pointer vers la 10e feuille du classeur dont le nom est "stat", on peut indifféremment taper :

worksheets(10) ou worksheets("stat")

les guillemets étant nécessaires pour que stat soit traité comme une chaîne de caractères et non comme une variable.

2.2.2 La hiérarchie des objets

Les objets sont organisés hiérarchiquement :
- au sommet réside Excel elle-même, l'objet *Application* ;
- puis vient notamment la collection des classeurs (*workbooks*) ;
- les feuilles de travail des classeurs (*worksheets*), la collection des graphiques (*charts*) ;
- ....

Les plages de cellules (*range*) constituent l'exception selon laquelle à chaque objet correspond une collection. Par contre à la cellule (*cell*) correspond la collection des cellules (*cells*). Le point . permet de spécifier la hiérarchie des objets. Ainsi si l'on veut faire pointer l'ordinateur vers la plage "A1 :B10" de la feuille appelée "travail" du classeur nommé "essai", on peut écrire :

Application.Workbooks("essai").Worksheets("travail").Range("A1 :B10")

Par défaut, l'ordinateur a une hiérarchie définie par défaut. Elle correspond au dernier fichier utilisé sous Excel, à la dernière feuille activée, etc. Aussi c'est cette hiérarchie qui sera utilisée si on utilise une spécification incomplète, par exemple si on ne spécifie dans notre exemple que la plage en écrivant seulement Range("A1 :B10"). Si le dernier fichier activé, la dernière feuille utilisée sont bien respectivement "essai" et "travail", cela n'aura donc pas de

conséquence. Par contre si cela n'est pas le cas, le programmeur aura des surprises ! Il est donc important d'être prudent dans les racourcis d'écriture. On pourra toujours faire l'économie du premier terme et donc écrire seulement :

Workbooks("essai").Worksheets("travail").Range("A1 :B10")

Mais pour les autres niveaux, tout dépend encore une fois des objets actifs sous Excel. Ainsi si fichier "essai" est le fichier par défaut (peut-être parce qu'il a été antérieurement activé par une commande : Workbooks("essai").Activate) alors la chaîne se réduira à :

Worksheets("travail").Range("A1 :B10")

Remarque 2 *Le point est également utilisé non seulement pour passer d'un objet vers un de ses subordonnés mais aussi pour pointer vers une propriété ou vers une action. Ainsi, si l'on veut colorier la plage "A1 :B10" en gris, on écrira :*

Range("A1 :B10").Interior.Color=vbGrey

où Interior est l'intérieur des cellules de la plage (un objet donc), Color est évidemment une propriété de l'intérieur, vbGrey est la constante désignant sous Visual Basic le gris (ou plus exactement une manière de définir un gris sous VBA). Si l'on veut aussi supprimer la feuille "travail", on utilisera aussi la syntaxe suivante :

Worksheets("travail").Delete

Dans ce dernier exemple, comme dans le précédent, on n'a pas explicité toute la hiérarchie des objets en supposant que les objets supérieurs étaient par défaut correctement définis - ce qui assure par exemple dans notre dernier exemple que notre écriture simplifiée est équivalente à

Application.Workbooks("essai").Worksheets("travail").Delete

2.3 Le modèle objet d'Excel

A chaque application d'*Office* (et *Office* lui-même) correspond un modèle objet, hiérarchie de collections d'objets. Comme ce modèle objet diffère par exemple entre Excel et Word, maîtriser VBA pour Excel n'assure pas de pouvoir sans difficulté programmer VBA sous Word, le langage commun s'appliquant à des objets parfois différents. Pour avoir un aperçu du modèle objet de VBA pour Excel, il suffit dans l'éditeur de VBA de solliciter l'aide

FIG. 2.5 – Le modèle objet d'Excel

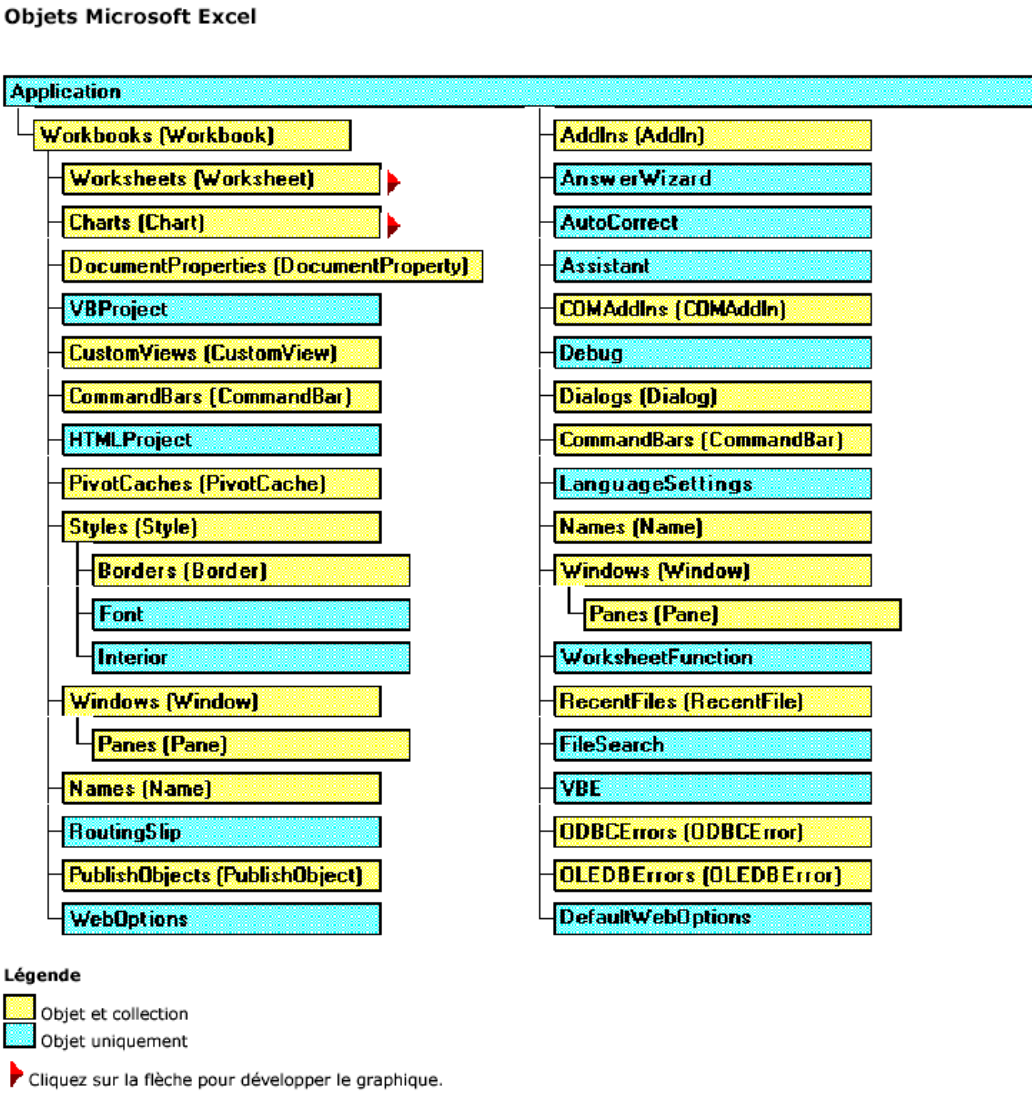


FIG. 2.6 – Le modèle objet d'Excel (suite)

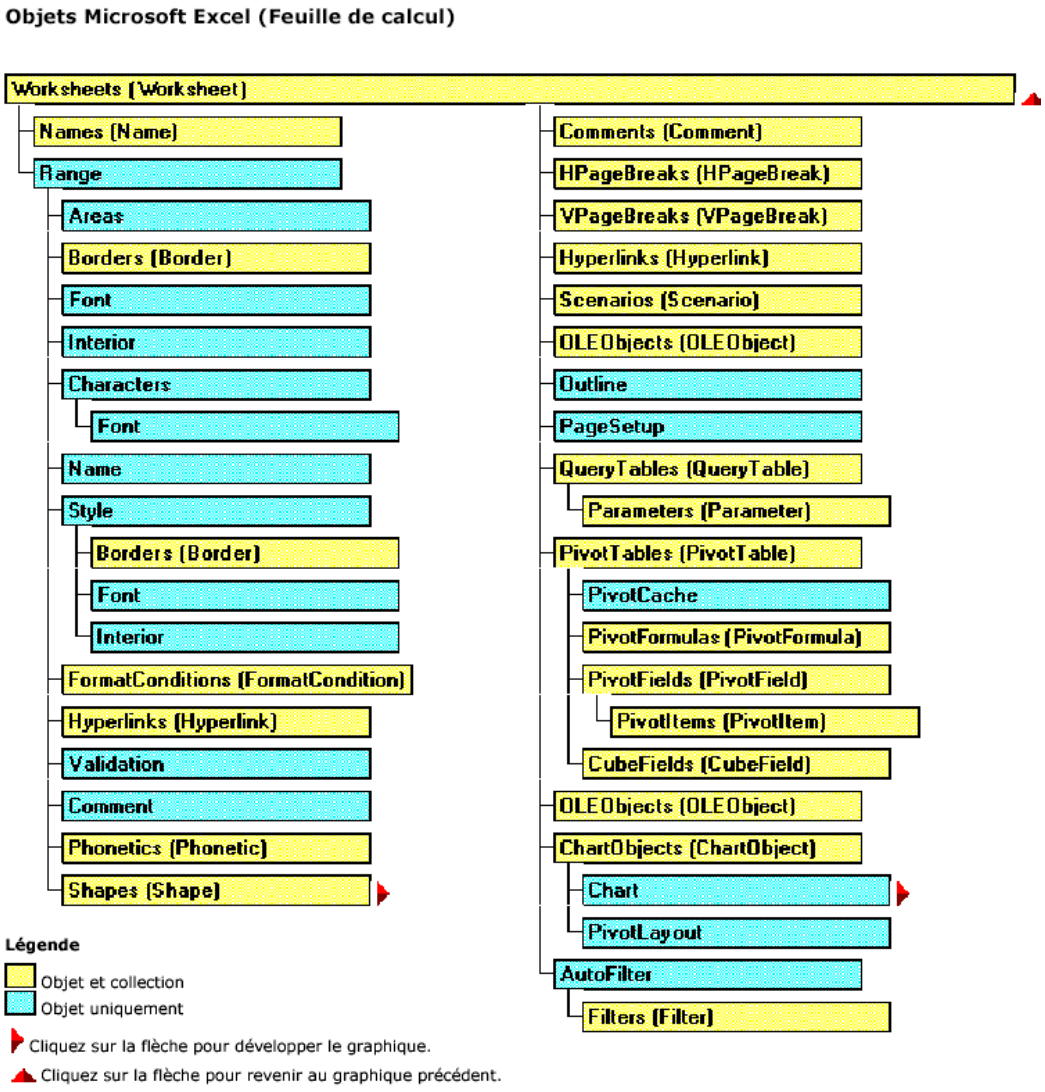


FIG. 2.7 – Le modèle objet d'Excel (fin)

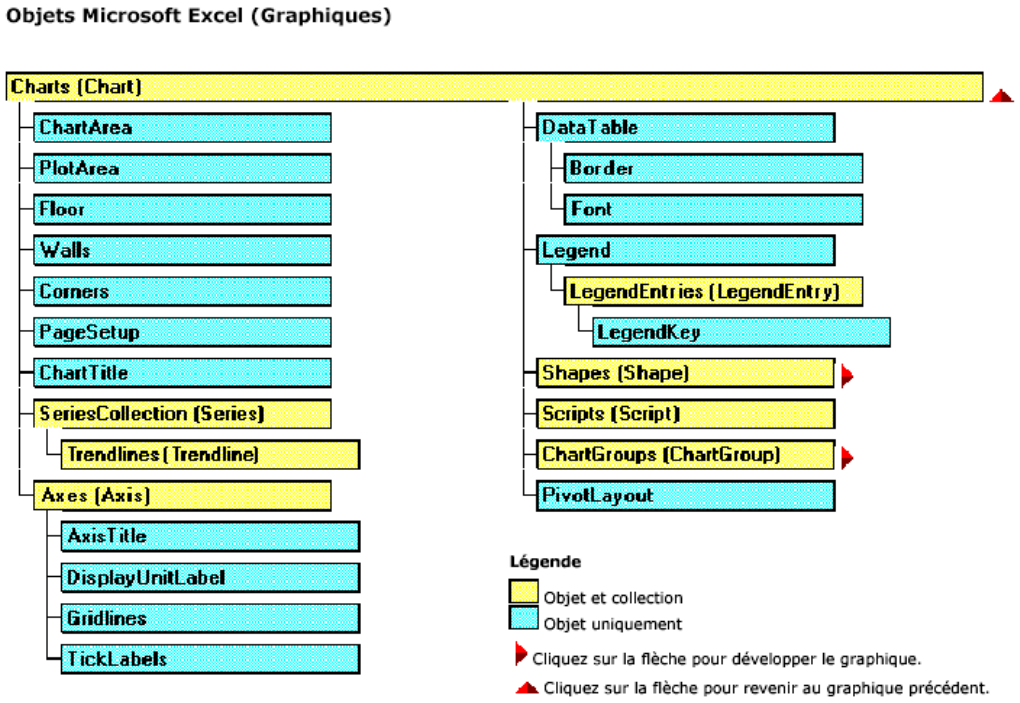
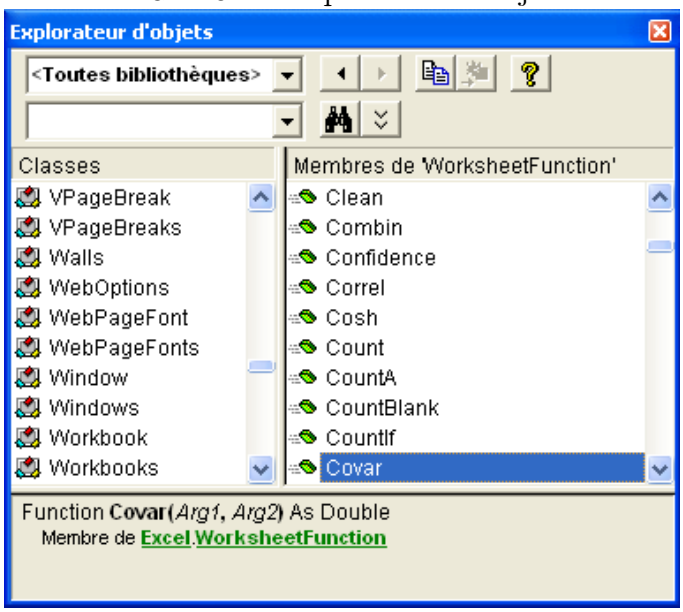


FIG. 2.8 – L'explorateur d'objets



en y tapant *Objets Microsoft Excel*. On y obtient l'aperçu représenté sur les figures 2.5, 2.6 et 2.7.

On retrouve sur ces graphiques certains objets déjà évoqués à côté de nombreux autres. Le modèle objet d'Excel est en effet riche et complexe. Connaître tous ces objets est naturellement impossible. D'autant que d'une version à l'autre, le modèle s'enrichit progressivement. Heureusement, sous le VBE, on a accès à un outil fort utile : *l'explorateur d'objets* que l'on peut obtenir soit via le menu *Edition*, soit en tapant *F2*. L'explorateur d'objets permet d'avoir accès à l'ensemble des libraries simultanément ou d'en cibler pour limiter la masse des objets à explorer. Les principales librairies sont celles d'Excel et de VBA.

Pour avoir de l'aide sur une fonction (par exemple Abs dans la figure ci-dessous), il suffit alors de pointer sur la fonction choisie, puis à cliquer sur l'icône de l'aide (le point d'intérrogation jaune)

Une autre source d'information très utile est celle de l'Info Express Automatique. Pour l'objet, sous le VBE, il faut aller dans le menu Outils → Options et cocher Info Express Automatique (cf figure 2.10).

FIG. 2.9 – L'aide sous l'explorateur d'objets

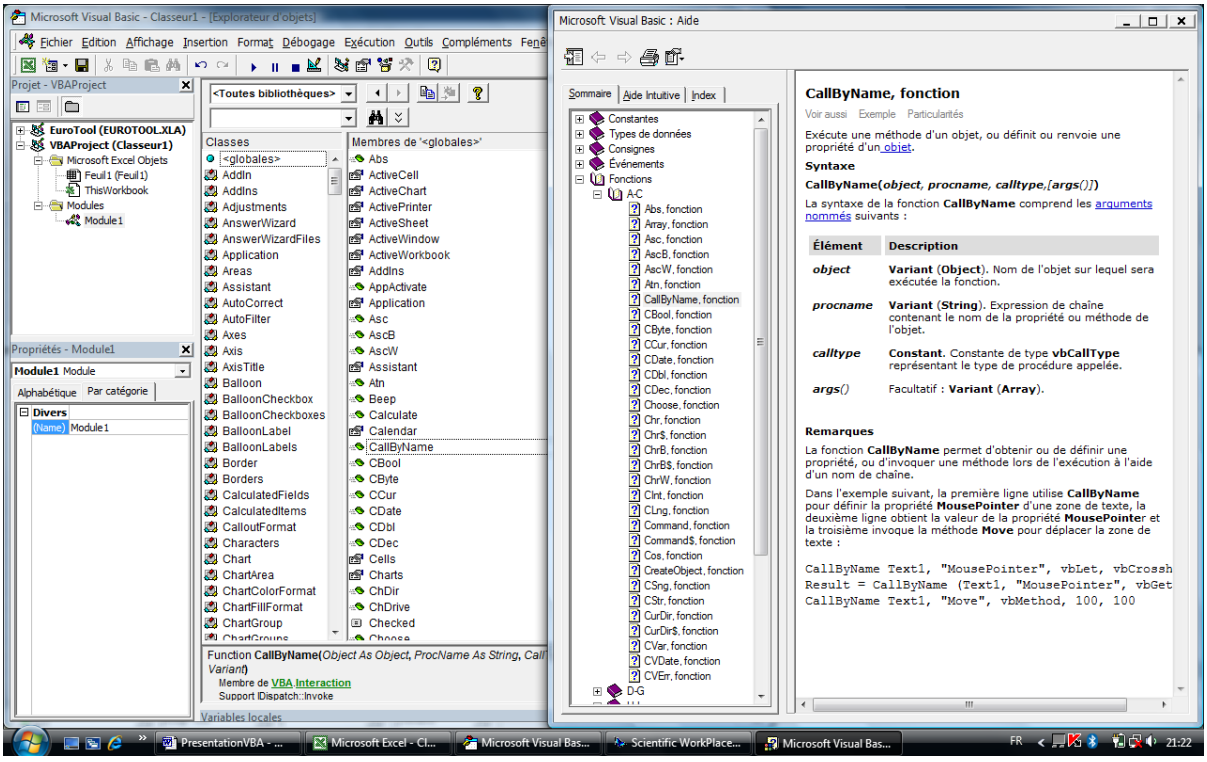
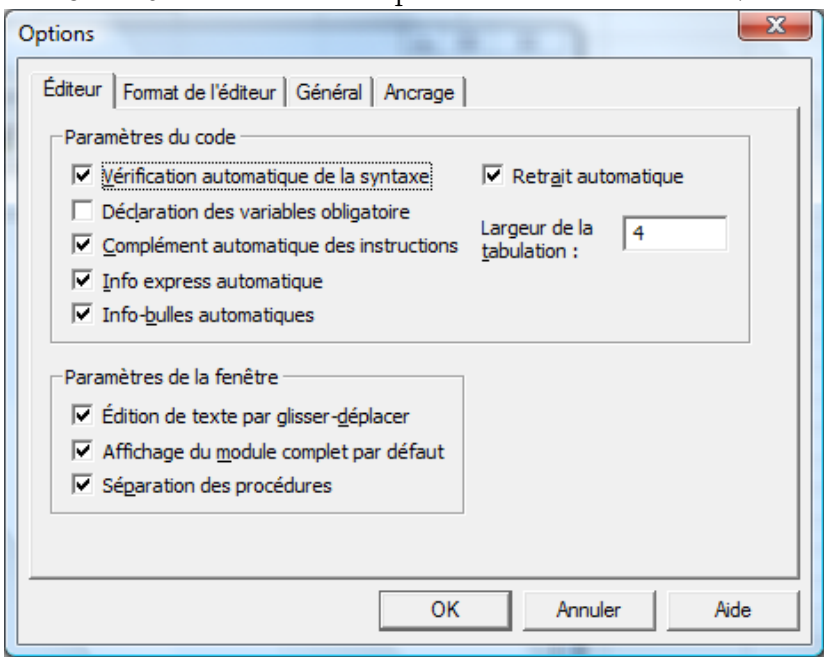


FIG. 2.10 – Le menu des options d'information du VBE



2.4 Propriétés et méthodes

Les propriétés d'un objet font références à ses différentes caractéristiques. Par exemple pour une cellule : sa couleur, sa hauteur, sa largeur, sa police, etc.

On peut :

– modifier les propriétés d'un objet en utilisant une syntaxe de la forme

Expression.Propriété=Valeur

Par exemple si l'on veut que la valeur de la cellule A1 soit 150, on utilisera l'instruction suivante

Cells(1,1).Value=150

ou de manière équivalente :

Range("A1").Value=150

– interroger un objet pour récupérer sa propriété, par exemple sa valeur ; ainsi on pourra stocker la valeur de la valeur de la cellule A1 dans une variable Contenu_A1 en tapant l'instruction suivante

Contenu_A1=Range("A1").Value

Les méthodes d'un objet sont les actions que l'on peut entreprendre sur un objet, par exemple le copier, le couper, le déplacer, le sélectionner, etc. De manière générale l'instruction est alors de la forme :

Expression.Méthode

Ainsi, si l'on veut copier le contenu la cellule A1, on écrira par exemple :

Range("A1").Copy

Ou si l'on veut la transférer son contenu à la cellule A2 on pourra écrire :

Range("A1").Cut Destination :=Range("A2")

Si l'on veut effacer le contenu d'une cellule ou d'une place on peut utiliser l'instruction ClearContents :

Range("A1").ClearContents

On peut aussi ajouter des éléments à la collection des feuilles (Worksheets) ou à celle des classeurs ouverts (Workbooks), les fermer, les sauver à l'aide des méthodes *Add*, *Close*, *Save* par des instructions du type :

$$Worksheets.Add$$

$$Workbooks.Add$$

$$Workbooks.Close$$

$$Workbooks.Save$$

On peut aussi bouger la position d'une feuille à l'intérieur du classeur, notamment après une autre feuille, soit en indiquant les indices des feuilles, soit leur noms par des instructions du type :

Worksheets(1).Move After :=Worksheets(3)

Worksheets("feuille1").Move After :=Worksheets("feuille3")

Chapitre 3

Premiers pas

3.1 L'enregistreur de macros

Excel comprend un enregistreur de macros susceptible d'enregistrer la plupart des actions que l'on exécute sous Excel et donc de générer des programmes. Pour déclencher l'enregistreur de macros, on exécute la séquence suivante dans le menu Excel :

Outils → Macro → Nouvelle Macro

La fenêtre de l'enregistreur (figure 3.1) permet de choisir le nom de la macro, le fichier dans lequel elle va résider (le fichier courant, un autre classeur ou un classeur caché appelé *classeur des macros personnelles*). Enfin, la fenêtre permet de décrire la procédure enregistrée.

Cette action crée si nécessaire un module où une procédure de type Sub est enregistrée. Dans l'interface Excel, vient se placer un bouton d'enregistrement de la macro sur lequel on doit cliquer pour arrêter le processus.

L'enregistreur de macros permet donc aux novices de VBA de disposer de procédures qu'il peuvent réutiliser plus tard, dans le fichier voire dans un autre fichier. En effet, il est possible d'enregister la macro dans le classeur caché dit de macros personnelles (cf figure 3.1). Ce fichier est disponible à tout instant dès lors qu'Excel est ouvert et ses procédures peuvent donc être sollicitées à partir de n'importe quel autre fichier.

A ces multiples avantages de l'enregistreur correspondent des inconvénients notoires souvent mis en avant par les programmeurs. Le code généré par cet outil est en effet souvent excessif par la taille. En effet, si par exemple, on change une propriété du format des cellules (par exemple la couleur de la bordure), l'enregisteur détaille l'ensemble des propriétés de celles-ci y compris celles qui n'ont pas été affectées.

FIG. 3.1 – La fenêtre de l'enregistreur de macros

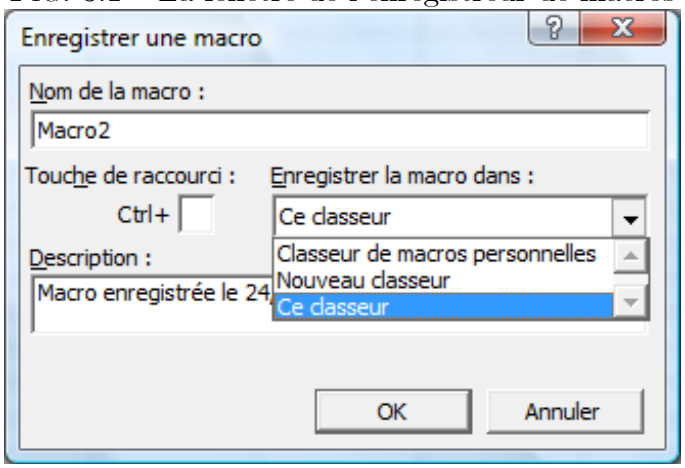
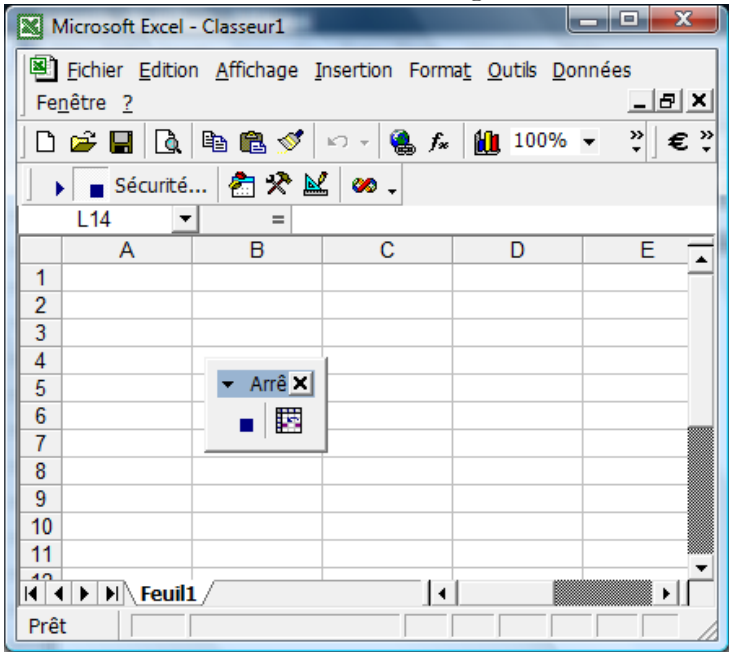


FIG. 3.2 – Le bouton d'arrêt de l'enregistrement de la macro



Ainsi le code généré pour une modification de la police de caractères pour pour qu'elle soit du type Times New Roman, sa taille 12 points, en gras, et en rouge :

```
With selection.Font
  .Name=''Roman''
  .Size=12
  .Strikethrough=False
  .SubScript=False
  .Outlinefront=False
  .Sahdow=False
  .Underline=xlNone
  .ColorIndex=xlAutomatic
End With
With Selection.Font
  .Name=''Times New Roman''
  .Size=12
  .Strikethrough=False
  .SubScript=False
  .Outlinefront=False
  .Shadow=False
  .Underline=xlNone
  .ColorIndex=xlAutomatic
End With
Selection.Font.Bold=True
Selection.Font.ColorIndex=3
```

Ceci répliqué sur des dizaines d'actions entraîne naturellement une explosion du nombre de lignes de code. Aussi est-il souvent conseillé si l'on veut utiliser l'enregistreur de nettoyer le code des lignes inutiles avant de l'enregistrer définitivement. Ainsi dans notre exemple, le code se simplifie drastiquement :

```
With selection.Font
  .Name=''Times New Roman''
  .Size=12
  .Bold=True
  .ColorIndex=3
End With
```

Remarque 3 *Dans le code ci-dessus, pour éviter de réécrire 4 fois selection.Font, on utilise la construction with. Avec celle-ci, on spécifie d'abord l'objet ou la propriété que l'on veut manipuler répétitivement (ici selection.font),*

puis aux lignes suivantes on précise les propriétés ou actions (aprèsl'opérateur .). La fin de la construction est signalée par End With . Cette construction est très souvent utile notamment dans la mise en forme des feuilles Excel.

Cependant même pour les développeurs, l'enregistreur de macros est utile, notamment comme système d'aide. Certains objets (comme par exemple les graphiques) sont en effet très complexes par leurs options. Plutôt que s'astreindre à connaître tous ces éléments, il est souvent plus facile de mettre en place manuellement un objet en déclenchant l'enregisteur, puis de retrouver l'ensemble des propriétés de l'objet recherché (le graphique en l'occurence), de copier le code obtenue en en faisant une correction.

3.2 Types de variables et procédures

VBA pour Excel utilise non seulement les ressources d'Excel (en maniant les cellules notamment) mais peut également, comme la plupart des logiciels, manier des variables qui lui sont propres pour effectuer des opérations sur celles-ci.

3.2.1 Les principaux types de variables

Outre les types usuels (chaînes de caractère, entiers naturels, réels, etc.), *VBA* utilise aussi un type de variables extrêmement souple : les variables de type *variant*. Naturellement la contrepartie d'une plus grande souplesse de la variable est l'espace qu'elle occupe dans la mémoire de l'ordinateur. Déclarer une variable permet de définir les actions et opérations qu'elle est susceptible de subir : impossible par exemple d'effectuer des opérations algébriques (addition, multiplication, etc.) sur des variables déclarées comme des chaînes de caractères. Déclarer une variable permet aussi d'éviter les doublons.

Les principaux types de variables utilisées sont donc les suivants :

- *String* - chaînes de caractères ;
- *Integer* - entiers relatifs compris entre $-32\,768$ et $32\,767$; une variable de type Integer occupe 2 octets de mémoire ;
- *Long* - entiers relatifs compris entre $-2\,147\,483\,648$ et $2\,147\,483\,647$; en contrepartie, elle occupe 4 octets de mémoire ;
- *Boolean* - variables de type booleenne, prenant comme valeurs soit $True$, soit $False$;
- *Double* - variables réelles à virgule flottante à double précision dont les valeurs peuvent aller de $1,79769313486231E308$ et $-4,94065645841247E-324$ pour les nombres négatifs et entre $4,94065645841247E-324$ et

$1,79769313486231E308$ pour les positifs ; en mémoire une variable de type *Double* occupe 8 octets de mémoire ;
- *Variant* - Variant est un type de données spécial pouvant contenir des données de toutes sortes ; sa contrepartie est d'être la variable la plus coûteuse en mémoire.

Sous VBA, la déclaration du type des variables n'est pas obligatoire : en l'absence de toute déclaration, la variable est réputée en effet de type *variant* (et dans ce cas tout se passe comme si l'ordinateur détectait si la variable est numérique, booléenne, ou autres). Evidemment ceci est coûteux en mémoire. Néanmoins, l'explosion de la puissance des ordinateurs a considérablement réduit cet inconvénient. Aujourd'hui l'intérêt de déclarer une variable réside essentiellement dans le fait qu'elle permet d'éviter des erreurs.

Pour déclarer les variables, la commande la plus communément utilisée est *Dim* (pour dimension). Pour que la variable x soit déclarée comme un entier relatif, on écrira ainsi :

```
Dim x As Integer
```

Si l'on a plusieurs variables du même type à déclarer, par exemple x et y, on peut naturellement soit les lister les unes derrières les autres (avec un saut de ligne) :

```
Dim x As Integer
Dim y As Integer
```

Evidemment la tentation est alors de racourcir la déclaration. Par exemple en tapant :

```
Dim x, y As Integer
```

Mais dans cette écriture, seule y sera déclarée comme un Integer, x sera considérée comme non déclarée et donc assimilée à une variable de type variant. La seule déclaration en ligne correcte suppose l'utilisation de la déclaration *As* pour chacune des variables :

```
Dim x As Integer, y As Integer
```

3.2.2 Les procédures

Les procédures principales utilisées sous Excel sont de deux types :
- les procédures de type *Sub*
- les procédures de type *Function*.

Chaque procédure est construite de la même manière avec une déclaration du type, du nom (et des arguments possibles), et une fin marquée par la présence des déclarations *End Sub* ou *End Function*. Ainsi la procédure Sub aura la structure suivante :

Sub nom(argument 1 As type de variable, ...)
< *déclaration des variables avec Dim* >
< *lignes d'instruction* >
End Sub

La déclaration d'arguments est facultative. Ainsi dans la procédure ci-dessous, aucune variable n'est déclarée. La procédure se contente de prendre la valeur (supposée numérique) des cellules *A1* et *A2*, de les multiplier et de reporter le résultat dans la cellule *B1* :

```
Sub procMult()
Dim x As Double, y As Double, z As Double
x=Range(''A1'').Value
y=Range(''A2'').Value
z=x*y
Range(''B1'').Value=z
End Sub
```

On remarque dans cette procédure simple, la déclaration séquentielle des variables x, y, z, puis l'affectation à x et y de la valeur des deux cellules A1 et A2. z est bien obtenue par le produit (symbole *) des deux variables. Enfin, on affecte à la cellule B1 la valeur de z. Naturellement, cette programmation n'est ni très rapide, ni très économe en mémoire. On aurait pu très directement écrire :

```
Sub procMult()
Range(''B1'').Value=Range(''A1'').Value*Range(''B1'').Value
End Sub
```

Les procédures de type *Function* sont très similaires aux procédures Sub dans leurs constructions. Elles n'en diffèrent que par le fait qu'elles sont supposés retournés directement un résultat (comme une fonction mathématique). En conséquence, le résultat de la fonction elle-même peut avoir un

type et la dernière ligne est l'affectation au nom de la fonction d'un résultat :

Function nom(argument 1 As type de variable, ...) As type de variable
< *déclaration des variables avec Dim* >
< *lignes d'instruction* >
nom = < *résultat* >
End Functiion

Ainsi la fonction *fnMult* combinant deux réels x et y pour en faire le produit s'écrira par exemple :

```
Function fnMult(x As Double, y As Double) As Double
Dim z As Double
z=x*y

fnMult=z
End Function
```

La fonction *fnMult* utilise une variable intermédiaire (de type *Double*) pour enregistrer le résultat de la multiplication des deux variables. Le résultat est déclaré de type *Double* comme le montre la première ligne. Evidemment, comme pour la procédure plus haut, il est possible de faire l'économie des variables intermédiaires en écrivant directement :

```
Function fnMult(x As Double, y As Double) As Double

fnMult=x*y
End Function
```

3.2.3 Les variables objet

A côté des variables précédentes, les objets du modèle Excel (notamment les classeurs, les feuilles de travail et les graphiques, les plages de cellules) peuvent être déclarés comme des variables. La déclaration se fait aussi avec *Dim*. Ainsi, si l'on écrit :

```
Dim ws As Worksheet
```

La variable *ws* sera interprété comme étant un objet worksheet. Si l'on veut que *ws* renvoie à une feuille particulière (par exemple une feuille dont le nom est "calcul") alors on pourra réaliser cette affectation par la commande suivante :

```
Set ws=Worksheets(‘‘calcul’’)
```

La même procédure (avec *Dim* puis *Set*) est également utilisable pour les plages (*range*), les graphiques (*chart*), les classeurs (*workbook*). Ceci est notamment très utile lorsque l'on manie des feuilles appartenant à des fichiers différents en allégeant considérablement l'écriture.

3.2.4 Les constantes et les constantes pré-définies

Au cours de l'exécution d'un programme, la valeur d'une variable peut être redéfinie. Si l'on veut maintenir constante la valeur d'un paramètre, il est alors souhaitable de la définir comme une *constante*. Elle se déclare à l'aide de la déclaration *Const*. Ainsi par exemple si le taux sans risque du marché est supposé avoir une valeur constante égale à 2%, on pourra au début du programme écrire par exemple :

```
Cst riskFree=0.02
```

Dans l'exécution du programme il sera alors impossible d'exécuter une nouvelle affectation de valeur à la variable *riskFree*.

Il existe aussi des constantes pré-définies. Selon qu'elles sont des constantes d'Excel, de VBA ou d'Office, elles comportent le préfixe *xl*, *vb* ou *mso*. Ainsi, sous VBA, les couleurs sont définies par les constantes *vbRed*, *vbGreen*, *vbGrey*, etc. . Sous Excel, les directions des déplacements verticaux vers le haut ou vers le bas sont définies par les constantes *xlUp* et *xlDown*, tandis que les directions horizontales vers la gauche et vers la droite sont définies par les constantes *xlToLeft* et *xlToRight*.

3.3 Outils d'interaction

Dans l'exécution d'un programme, il est parfois nécessaire que l'utilisateur fixe la valeur de certains paramètres. Pour réaliser cette interaction entre l'ordinateur et l'utilisateur, *Visual Basic sous Excel* met à la disposition plusieurs outils d'interaction dont les deux plus simples sont :

- la MsgBox ;
- l'InputBox.

3.3.1 La MsgBox

MsgBox est une boite de dialogue qui permet en réponse à une question d'apporter différentes réponses codées (oui, non, etc.). Les arguments sous VBA de la commande MsgBox sont notamment :

- le message lui-même (seul argument obligatoire) ;
- les boutons à afficher ;
- le titre de la boîte de dialogue ;

La disposition pour ces trois éléments est :

MsgBox(message,[boutons],[titre],[autresvariables])

où les variables entre crochets sont optionnelles. Le message est une variable de type string (chaîne de caractères) qui peut simplement être une question entre guillemets.

Les boutons sont codifiés et peuvent notamment prendre les valeurs suivantes :[1]

- vbYesOnly (la boîte de dialogue comprend un seul bouton) ;
- vbYesNo (la boîte de dialogue comprend deux boutons oui et non) ;
- vbYesNoCancel (la boîte de dialogue comprend trois boutons oui, non et annulation).

Le titre est une variable de type string à préciser.

Les différentes apparences de la boîte de dialogue selon les arguments de MsgBox sont :

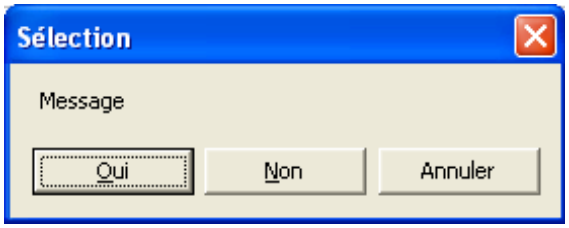
- MsgBox("message")

- MsgBox("message","sélection")

[1] Se référer à l'aide de VBA pour avoir plus d'information sur MsgBox.

– MsgBox("message",vbYesNo,"sélection")

– MsgBox("message ",vbYesNoCancel,"sélection")

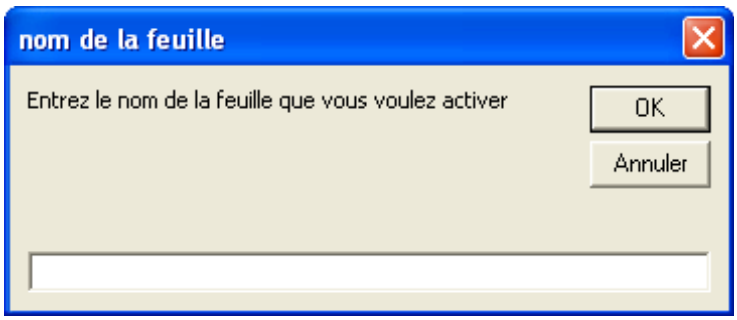


Lorsque les boutons comportent un choix, i.e. par exemple pour vbYesNo ou vbYesNoCancel, il est nécessaire de stocker la réponse dans un objet (soit une variable de type *string*, soit de type *variant*). La valeur stockée sera alors soit vbYes, soit vbNo, soit vbCancel et on peut alors utiliser cette valeur pour construire une boucle de contrôle conditionnellement à cette valeur dans le programme.

3.3.2 L'InputBox

MsgBox permet seulement des réponses codées binaires (oui/non) ou ternaires (oui/non/annulation). Parfois il est nécessaire de renvoyer une réponse qui soit un nombre ou un mot. Pour donner ces réponses, on peut utiliser l'autre boîte de dialogue qu'est l'inputBox (voir l'aide de VBA).

Un exemple utilisant l'InputBox est celui représenté sur la figure ci-dessous :



L'input comporte un titre (" nom de la feuille "), un contenu (" Entrez le nom de la feuille et que vous voulez activer "), une zone où l'on peut taper la réponse souhaitée, enfin deux boutons pour terminer l'utilisation de l'inputBox (avec les deux boutons OK, annuler). Même si d'autres arguments peuvent être définis, ceux cités peuvent suffire pour la plupart des applications. Pour obtenir l'inputBox de la feuille, on a tapé :

```
q=InputBox('' Entrez le nom de la feuille que vous voulez
activer '', '' nom de la feuille '')
```

q est la variable dans laquelle on va stocker la réponse. Elle peut être soit une variable de type variant, soit selon la nature de la réponse une variable de type string (pour les chaînes de caractère), de type integer, double pour les nombres, etc.

3.4 Fonctions Excel dans VBA

VBA comporte des fonctions financières et mathématiques, etc dont on peut trouver la liste dans l'explorateur d'objets, à l'intérieur de la librairie VBA.

Néanmoins comme le montre la figure 3.3 pour les fonctions mathématiques, le nombre de fonctions peut être très modeste. Ceci est évidemment la conséquence du fait que l'on a déjà à notre disposition l'ensemble des fonctions mathématiques, statistiques, financières, etc. d'Excel. Pour faire appel à ces fonctions sous VBA, il convient soit de taper *Application.WorksheetFunction* ou plus simplement *WorksheetFunction* en appliquant ceci soit à un nombre, soit à une variable, soit à une plage. Ainsi pour calculer la racine carrée de 25, on pourra si l'on utilise la fonction *Racine* (*Sqrt* en anglais) d'Excel soit taper directement :

```
WorksheetFunction.Sqrt(25)
```

soit si 25 est par exemple la valeur de la cellule $B4$:

```
WorksheetFunction.Sqrt(Range(‘‘B4’’))
```

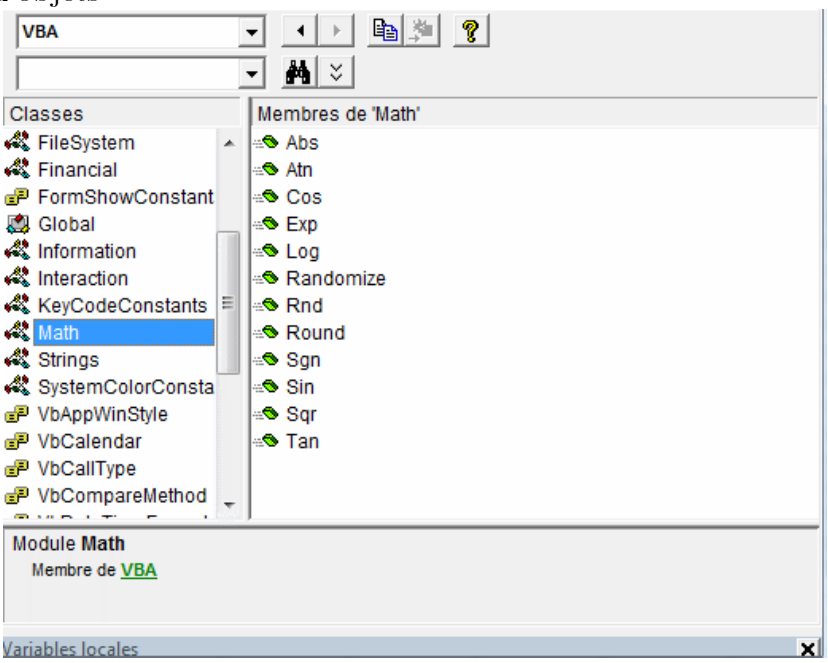
ou encore :

```
WorksheetFunction.Sqrt(Cells(4,2))
```

soit si 25 est la valeur affectée à une variable de type *Double* appelée x :

```
WorksheetFunction.Sqrt(x)
```

FIG. 3.3 – La collection des fonctions mathématiques de VBA dans l'explorateur d'objets



3.5 Quelques astuces

3.5.1 Mise en forme du projet VBA

Les programmes réalisés en VBA, dans le cadre de projets, ne sont pas nécessairement courts. La figure 3.4 donne ainsi un aperçu sur les fenêtres du VBE d'un projet visant à récupérer les informations relatives aux mutual funds américains sous Yahoo ! Finance.

Comme on peut le voir dans la fenêtre de l'explorateur de projet (en haut à gauche *VBA project*), le classeur *mutualFundsPerf.xls* comporte 7 feuilles de travail dont les noms sont *requete*, *intro*, etc.. Le répertoire des modules comporte lui-même 8 répertoires : *contenu*, *fnProcDivers*, etc. . Lorsque le projet devient suffisamment important il est alors conseillé tout d'abord de multiplier les commentaires au sein de chaque procédure, comme le montre la figure 3.5, pour rappeler la version du programme et rendre aussi lisible que possible les différentes étapes du programme.

Ensuite il est souvent souhaitable d'expliciter les différentes parties de chaque programme, voire de séparer le programme en plusieurs procédures. Enfin, si le projet comporte de multiples procédures, il est conseillé de les

FIG. 3.4 – Un exemple de projet comportant plusieurs modules et plusieurs procédures.

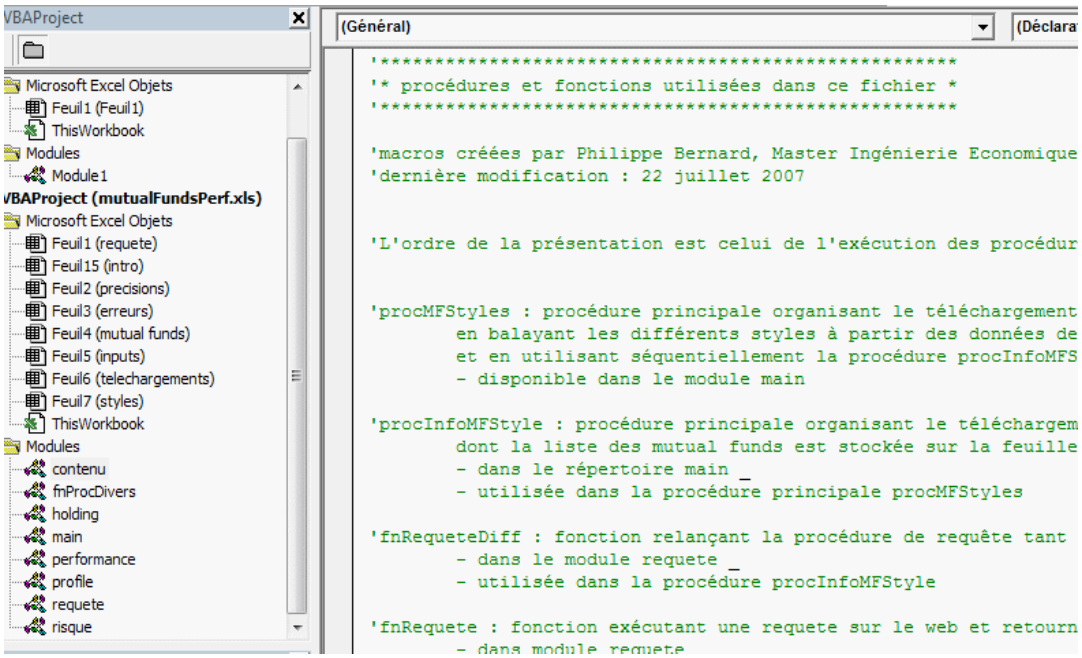


FIG. 3.5 – Un exemple de commentaires au sein d'un programme.

```
Sub procMFStyles()

'macro créée par Philippe Bernard, Master Ingénierie Economique et Financière, Universi
'dernière modification : 22 juillet 2007

'procédure principale organisant le téléchargement des données sur les mutual funds en
à partir des données de la feuille "mutual funds" de ce fichier et en utilisant séquent

'déclaration des variables
Dim wbS As Workbook, wbC As Workbook
Dim wsS As Worksheet, wsI As Worksheet, wsM As Worksheet, wsL As Worksheet
Dim rg As Range
Dim style As String, message As String, feuil As String, fich As String
Dim i As Integer, LL As Integer, n As Integer, k As Integer

'mise en garde sur la nécessité d'avoir mise à jour au préalable la feuille "styles"
message = "Avant de lancer cette procédure, vous devez mettre à jour la liste des style
message = MsgBox(message, vbYesNo)

'arrêt du programme si la mise à jour n'a pas été faite
If message = vbNo Then Exit Sub

'*************************************************************************************
'* I. Définition des fichiers, des feuilles, initialisation *
'***********************************************************
```

lister dans un module (appelé par exemple contenu) en y précisant l'objet de chaque procédure et sa place dans la hiérarchie du programme, comme le montre la figure 3.6.

3.5.2 L'exécution du programme

La vitesse d'exécution n'est pas un des points forts de *VBA pour Excel* comparé à d'autres logiciels que l'on peut utilisé en finance (par exemple *Matlab*, *Gauss*). Encore une fois, l'intérêt de ce logiciel vient de la facilité de son langage et surtout de l'utilisation quasi-universelle du tableur *Excel*.

Pour gérer la lenteur d'exécution de VBA, au surplus, il est possible d'augmenter substantiellement celle-ci en empêchant l'incessant rafraichissement de l'écran d'Excel par la commande :

```
Application.ScreenUpdating=False
```

à laquelle correspondra à la fin du programme la commande inverse :

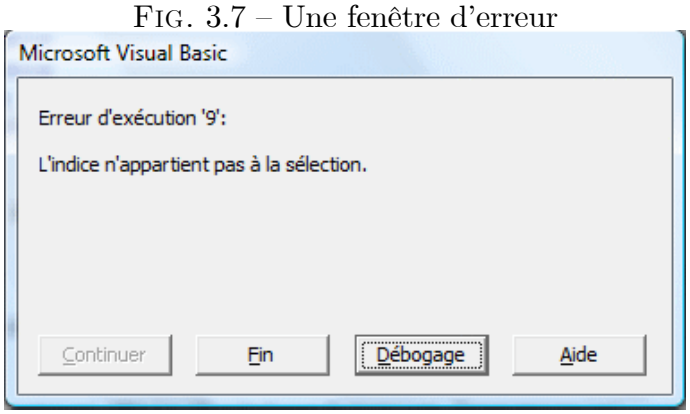
```
Application.ScreenUpdating=True
```

Comme ce gel de l'image peut enlever à l'utilisateur des repères au cours de l'exécution, il convient parfois de susbstituer à l'image des messages apparaissant sur la barre de statut d'Excel en utilisant des syntaxes du genre :

```
'*****************************************************
'* procédures et fonctions utilisées dans ce fichier *
'*****************************************************

'macros créées par Philippe Bernard, Master Ingénierie Economique et Financière,
'dernière modification : 22 juillet 2007


'L'ordre de la présentation est celui de l'exécution des procédures et des fonctions


'procMFStyles : procédure principale organisant le téléchargement des données sur
        en balayant les différents styles à partir des données de la feuille "mut
        et en utilisant séquentiellement la procédure procInfoMFStyle() _
        - disponible dans le module main

'procInfoMFStyle : procédure principale organisant le téléchargement des données
        dont la liste des mutual funds est stockée sur la feuille input _
        - dans le répertoire main _
        - utilisée dans la procédure principale procMFStyles

'fnRequeteDiff : fonction relançant la procédure de requête tant qu'elle échoue j
        - dans le module requete _
        - utilisée dans la procédure procInfoMFStyle

'fnRequete : fonction exécutant une requete sur le web et retournant vrai ou faux
        - dans module requete _
        - utilisée dans la fonction fnRequeteDiff

'fnDateYahoo : fonction calculant en fonction de la date d'aujourd'hui la date à
        sur Yahoo! Finance ont été calculées (fin de mois) _
        - dans le module fnProcDivers _
        - utilisée dans le module procInfoMFStyle

'fnDateY : fonction récupérant une date dans une chaîne de caractère sur la page
        - dans le module fnProcDivers _
```

FIG. 3.6 – Un exemple de module détaillant les fonctions et procédures d'un projet

FIG. 3.7 – Une fenêtre d'erreur




```
    Application.StatusBar=‘‘début de la première partie
du programme’’
```

Il est nécessaire à la fin de l'exécution de rendre la main à l'ordinateur sur la barre d'état en écrivant à la fin du programme :

```
    Application.StatusBar=False
```

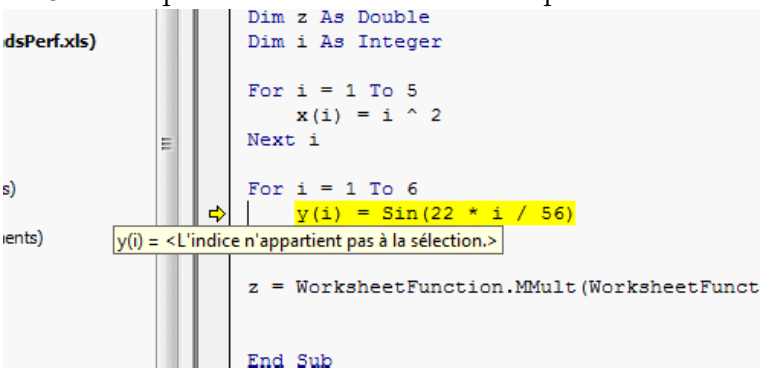
Enfin, pour gérer la possible absence de la barre d'état à l'écran, il peut être utile de la faire apparaître en codant au début du programme :

```
    Application.DisplayStatusBar=True
```

3.5.3 Debuggage

Sauf miracle, la rédaction de chaque programme comportera des erreurs qui provoqueront des erreurs lors des premières exécutions, et l'affichage d'agaçantes fenêtres comme celle de la figure 3.7. Comme on peut le voir sur cet exemple, elle offre la possibilité d'un debugage. L'activation de cette option conduira l'ordinateur à faire pointer une flèche jaune sur la ligne comportant l'erreur (figure 3.8). Si l'on pointe sur les éléments de cette ligne, la bulle information donnera alors soit la valeur des éléments, soit l'erreur à l'origine du bug. Ainsi dans l'exemple, la variable y est un vecteur colonne comportant 5 composantes alors que la boucle *For...Next* itère sur 6 composantes. Aussi pour $i = 6$, il est impossible d'affecter $\sin(22 * i/56)$ au vecteur y. D'où le message : <L'indice n'appartient à la sélection>, qui était également le message de la fenêtre d'erreur initiale (mais sans avoir l'information qu'elle s'appliquait à la variable y.

FIG. 3.8 – Un exemple d'information lors de la procédure de debuggage



Pour accélérer la détection des bugs, il est cependant conseillé avant tout lancement du programme de procéder à un debugage. Le menu de debugage (figure 3.9) offre la possibilité entre plusieurs options.

Le debugage *pas à pas détaillé* permet d'itérer le programme ligne par ligne. En utilisant le racourci de la touche *F8*, on passe ainsi de ligne en ligne, en vérifiant que le code s'exécute correctement ou non. Si le programme est trop long, ou s'il comporte des boucles longues, il convient alors d'introduire des points d'arrêt qui permettront l'exécution normale jusqu'au point d'arrêt. Ainsi sur la figure 3.10, en cliquant sur le bord de la fenêtre de code au niveau du début de la seconde boucle, on a introduit un point d'arrêt (en rouge). Si l'on déclenche normalement le programme il s'exécutera (sauf présence d'erreurs) jusqu'à ce point puis s'arrêtera on pourra alors soit le relancer normalement, soit en cliquant la touche *F8* passer en debugage au pas par pas détaillé.

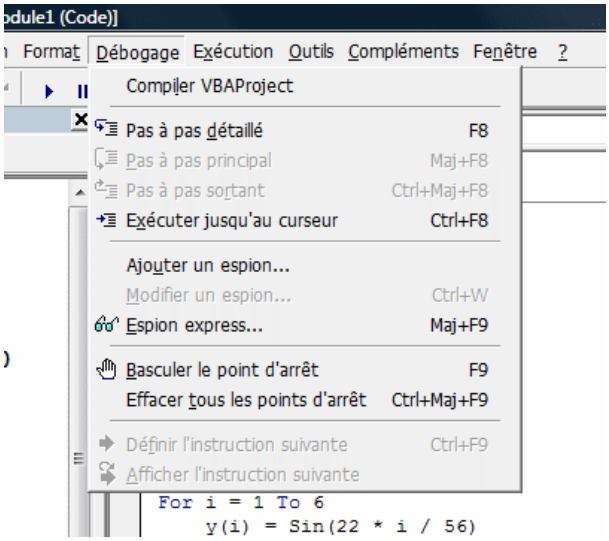


FIG. 3.9 – Le menu de debugage du VBE

FIG. 3.10 – Un exemple de point d'arrêt