Conception d'un Simulateur de Butterfly Obligataires sous Gambas 3.4

Écrit par Frédéric Leroy (aka Fly06)

6 Octobre 2013

Un Butterfly Obligataire est une stratégie de taux visant à exploiter une incohérence de prix entre trois obligations d'Etat de maturités différentes. Typiquement, la position consiste à être long des deux obligations adjacentes (barbell) et short de l'obligation centrale (bullet).

Dans ce contexte, notre « Simulateur de Butterfly Obligataires » est un outil à vocation pédagogique permettant de [1] :

1. Définir les obligations constitutives du butterfly en terme de maturité, de taux de coupon et de prix de marché
2. Structurer un butterfly obligataire en fonction de contraintes de couvertures (risques et/ou trésorerie) et de calculer un certain nombre d'indicateurs (spread, risques, ...)
3. Simuler et analyser l'évolution du P/L et du spread de la position à un horizon donné en fonction d'un scénario d'évolution des taux actuariels des trois obligations

Ce document décrit les spécifications techniques et fonctionnelles du projet ainsi que la façon dont ces spécifications seront mises en oeuvre dans Gambas 3.4 [2].

Le programme est classiquement constitué d'une partie « graphique » (interface utilisateur) et d'une partie « code » (traitements événementiels). L'architecture de la partie « code » est de type « architecture 3-tiers » constituée d'un « global manager » (controlleur), d'une partie « data management » minimaliste et d'une partie « business objects » plus complexe (cf. graphique ci-dessous).

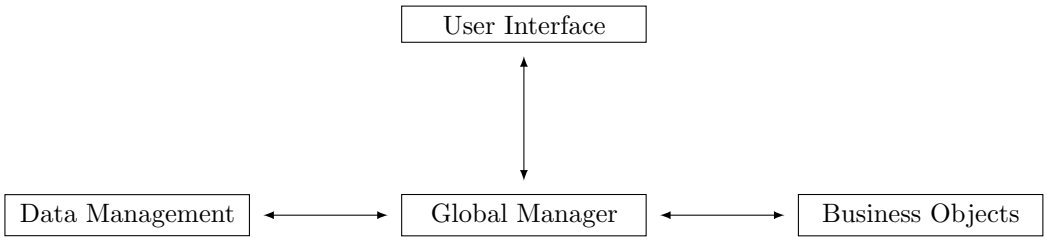

FIGURE 0.1 – Architecture 3-Tiers

1. Pour plus d'information sur les formules implémentées dans ce calculateur et sur les logiques financières associées, merci de vous reporter au Chapitre 4 du polycopié du « Cours sur les Produits et Stratégies de Taux » téléchargeable sur le site http ://www.tradingtaux.com/

2. Gambas est un « Visual Basic pour Linux » développé sous licence GNU/GPL v2 par Benoît Minisini et d'autres contributeurs. Pour des informations complémentaires, merci de vous reportez au site officiel du projet : http ://gambas.sourceforge.net/

Le contenu de ce document est divisé en trois sections :

1. Typologie des variables
2. Architecture du code
3. Autres aspects (hors calcul)

La partie principale du programme (calculateur) fait l'objet des deux premières parties tandis que les autres composantes du programme (initialisation, notification, ...) sont décrits dans la troisième partie.

1 Typologie des Variables

Rappelons que l'objectif est de réaliser un calculateur dédié aux stratégies de butterfly obligataires.

Ce calculateur impliquera deux types de variables :

1. Les paramètres (inputs) qui seront saisis
2. Les indicateurs (outputs) qui seront calculés et affichés

Dans ce contexte, on appelle « scénario » un n-uplet de valeurs admissibles pour les inputs du calculateur.

Ces différentes variables seront regroupées dans trois catégories distinctes :

1. Les variables obligataires
2. Les variables de butterfly
3. Les variables de simulation

Le fonctionnement de ce calculateur consistera simplement à mettre à jour automatiquement les outputs lorsqu'un input est modifié par l'utilisateur.

L'exposé des différentes variables est essentiellement énumératif et utilise le format suivant :

[Type du Control] Nom de la Variable (Unité)

1.1 Variables Obligataires

Les trois obligations constitutives du butterfly seront appelées Left, Center et Right et vérifieront :

$$Maturity_{Left} < Maturity_{Center} < Maturity_{Right}$$

Cette contrainte sur les maturités des obligations devra toujours être respectée sans que l'utilisateur n'ai à intervenir.

Pour chacune de ces trois obligations, les variables (obligataires) seront :
- Inputs :
  - [ComboBox] Maturité de l'obligation (années)

  - [MaskBox] Taux de coupon de l'obligation (pourcentage)
  - [MaskBox] Prix de l'obligation (pourcentage)
- Outputs :
  - [Label] Taux actuariel (pourcentage)
  - [Label] Duration (années)
  - [Label] Delta (Euros/bp)
  - [Label] Gamma (Euros/bp)
  - [Label] Theta (Euros/An)

Notons enfin que pour simplifier, on ne se placera systématiquement en date d'émission des obligations de sorte que les maturités seront des multiples entiers d'années [3].

1.2 Variables de Butterfly

Les variables de butterfly seront :
- Inputs :
  - [RadioButton] Type du butterfly (« Duration/Cash » vs « Shift/Twist »)
  - [MaskBox] Beta [4] ($0 < \text{beta} < 1$)
  - [MaskBox] Volatilité du 1er facteur actuariel (pourcentage)
- Outputs :
  - [Label] Montants nominaux des obligations Left et Right pour 1 Euro de nominal de l'obligation Center (pourcentages)
  - [Label] Taux de rendement actuariel (approché) du barbell (pourcentage)
  - [Label] Spread de taux entre le bullet et le barbell (points de base)
  - [Label] Forme de la courbe des taux (pente/convexité)
  - [Label] Biais de convexité (points de base)

Par ailleurs, les outputs obligataires seront dupliqués pour le barbell (calculs approchés).

1.3 Variables de Simulation

Les variables de simulations seront :
- Inputs :
  - [ComboBox] Horizon de la position (durée)
  - [MaskBox] Montant nominal de l'obligation Center (Millions d'Euros)
  - [MaskBox] Variation des taux actuariels des 3 obligations sur la période (en points de base)
  - [MaskBox] Taux de repo des 3 obligations sur la période (pourcentage)
- Outputs :
  - Analyse du P/L par composants :
    - [Label] Left/Center/Right/Total (Euros)
    - [Label] Obligation/Repo/Total (Euros)
    - [Label] Grand Total (Euros)
  - Analyse du P/L (obligataire uniquement) par « facteurs » dans le développement limité du P/L formel par rapport au taux actuariel et au temps :
    - [Label] P/L_Delta : Impact du facteur d'ordre 1 / Taux actuariel (Euros)
    - [Label] P/L_Gamma : Impact du facteur d'ordre 2/ Taux actuariel (Euros)
    - [Label] P/L_Theta : Impact du facteur d'ordre 1 / Temps (Euros)

3. 2A, 3A, 4A, 5A, 7A, 10A, 15A, 20A et 30A
4. Paramètre numérique spécifique au butterfly de type « Shift/Twist »

- Analyse du spread de convexité :
  - [Label] Taux de rendement actuariel (approché) du barbell à l'horizon (pourcentage)
  - [Label] Spread entre le bullet et le barbell à l'horizon (points de base)
  - [Label] Forme de la courbe des taux à l'horizon (pente/convexité)
  - [Label] Variation du spread de convexité sur la période (points de base)

Sur un plan purement graphique, ces différents contrôles pourront être groupés soit par nature de variables (obligations vs butterfly vs simulation) soit par type (inputs vs outputs). La façon dont les contrôles sont affichés sur la fenêtre principale (frmButterfly) n'ayant aucun impact au niveau du code.

2 Architecture du Code

Nous décrivons ici la partie interne du calculateur (architecture du code).

On va distinguer les aspects suivants :

1. Architecture globale
2. Modèle objet
3. Modèle événementiel

2.1 Architecture Globale

Le code de la partie « Business Object » de notre architecture 3-tiers sera modularisé selon un axe « Applicatif » et un axe « Objet ».

L'axe « Applicatif » sera spécifique au domaine traité (calculs financiers) :

1. Obligations
2. Butterfly
3. Simulation

L'axe « Objet » sera spécifique au type de langage utilisé (modèle objet Gambas) :

1. Classes :
   (a) Déclaration des variables privées (Inputs/Outputs)
   (b) Constructeur
   (c) Méthode publique de calcul
   (d) Méthodes privées de calcul spécifiques par output
   (e) Méthodes publiques d'accès aux variables (getter/setter)
2. Modules :
   (a) Déclaration des objets et/ou collections d'objets
   (b) Déclaration de raccourcis vers les contrôles du formulaire
   (c) Méthode publique de création des objets et/ou collection d'objets
   (d) Méthode publique d'initialisation des objets et/ou collection d'objets
   (e) Méthode publique de mise-à-jour (update) des objets, collections d'objets et contrôles associés

(f) Méthodes publiques d'accès à certaines méthodes publiques des objets (raccourcis)

(g) Méthodes privées (outils)

3. Formulaire :

   (a) Initialisation du calculateur par appel successif des méthodes de création et d'initialisation des objets et collections d'objets

   (b) Déclaration et création des groupes de contrôles et des évènements associés

   (c) Gestionnaires d'évènements (contrôles et objets de calcul)

Le tableau 1 ci-dessous donne les noms des différents fichiers par type de fichiers (objet) et types de calcul (applicatif).

	Obligations	Butterfly	Simulation
Classes	clsOblig	clsButterfly	clsObligSimul clsButterflySimul
Modules	modOblig	modButterfly	modSimul
Formulaire	frmButterfly		

TABLE 1 – Modularisation du Code

2.2 Modèle Objet

Nous décrivons ici les objets de calcul créés à partir des quatre classes précédentes (cf. tableau 1) :
– clsOblig : Calculs obligataires de base (taux actuariel, duration, ...)
– clsButterfly : Calculs de la structure du Butterfly (poids) et des indicateurs (spread, ...)
– clsObligSimul : Calcul de P/L par obligation sur une période donnée
– clsButterflySimul : Calcul du P/L du Butterfly sur une période donnée
Par « modèle objet » on entend à la fois :
– Les objets de calcul ainsi que les collections d'objets de calcul déclarés et créés dans les modules
– La façon dont ces objets sont reliés les uns aux autres (héritage d'objets)
Le tableau 2 ci-dessous fait le lien entre les objets de calcul ainsi que les collections d'objets et les modules.

Modules	Objets	Collections d'Objets
modOblig		colOblig
modButterfly	Butterfly	
modSimul	ButterflySimul	colObligSimul

TABLE 2 – Objets et Collections d'Objets

Rappelons qu'un objet est déclaré comme tel (myClass) dans son module et les méthodes publique de l'objet sont accessibles directement à partir du nom de l'objet déclaré.

Par exemple :

```
Dim myObject As MyClass
```

```
myObject = New MyClass(arg1, arg2, ..., argK)
myObject.DoThis()
```

Une collection d'objets est déclarée comme telle (Collection) et l'accès aux objets constitutifs de la collection est réalisé en précisant l'index de l'objet.

Par exemple :

```
Dim myObjectCol As New Collection
Dim myObjectColIndex As String[] = ["index1", "index2", "index3"]
Dim Index As String
For Each Index In myObjectColIndex
    myObjectCol[Index] = New MyClass(arg1, arg2, ..., argK)
Next
myObjectCol["index2"].DoThis()
```

Ces différents objets de calcul et collections d'objets de calcul sont reliés les uns aux autres via une relation de type « héritage d'objets ».

Le graphique ci-dessous décrit la hiérarchie des objets et collections d'objets.

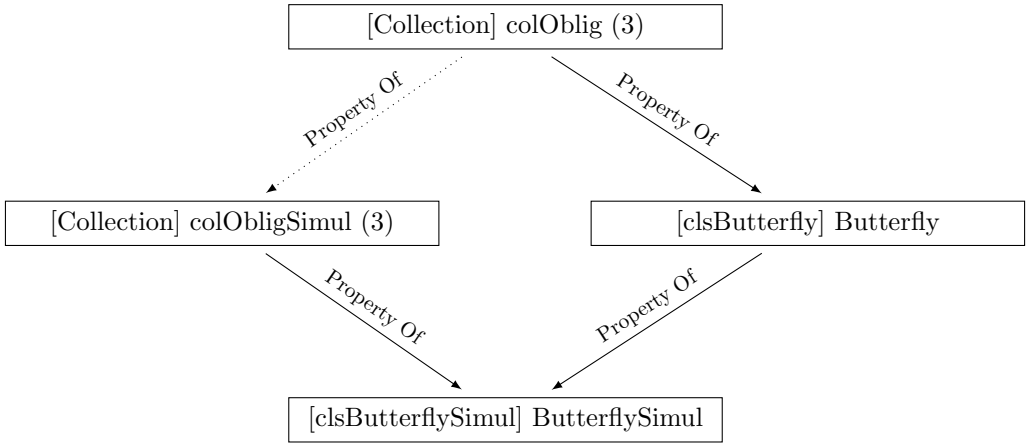

FIGURE 2.1 – Hiérarchie et Héritage entre les Objets et/ou Collections d'Objets

Dans ce graphique :
- Les boites représentent les objets ([Classe] Nom) ou collections d'objets ([Collection] Nom (Taille de la Collection))
- Les flèches représentent les relations de « propriété » :
  - Une flèche en trait plein allant d'un objet ou d'une collection d'objets A vers un objet B signifie que l'objet ou la collection d'objets A est une propriété de l'objet B
  - Une flèche en trait pointillé allant d'une collection d'objets A vers une collection d'objets B signifie que la relation de propriété s'applique aux objets individuels des deux collections et non aux collections elles-mêmes

Le modèle hiérarchique implémenté n'est pas un modèle de type « hiérarchie/héritage de classes » mais un modèle de type « hiérarchie/héritage d'objets ».

Supposons que l'on ait la situation fictive suivante :
- Classes :
  - myClass_A qui n'hérite d'aucune classe
  - myClass_B qui hérite de myClass_A
- Objets :
  - myObject_A de type myClass_A
  - myObject_B de type myClass_B

Dans l'hypothèse où les propriétés héritées de myClass_A ont les mêmes valeurs dans myObject_A et myObject_B, on ne peut pas dire pour autant que myObject_B hérite de myObject_A car ces deux objets ont une existence propre en RAM. En d'autres termes, la partie de myObject_B qui correspond à myClass_A et myObject_A sont deux entités « physiques » différentes.

Si l'on souhaite éviter la duplication « physique » et les éventuelles incohérences qui pourraient en résulter, il faut implémenter l'héritage d'objet.

Dans ce cas, la situation fictive précédente devient :
- Classes :
  - myClass_A et myClass_B qui n'ont pas de relation d'héritage entre elles
  - myClass_B contient une propriété myObjectProperty_A de type myClass_A instanciée dans le constructeur de myClass_B
- Objets :
  - myObject_A de type myClass_A
  - myObject_B de type myClass_B dont la propriété myObjectProperty_A « pointe » vers myObject_A

L'intérêt du modèle d'héritage d'objets est qu'il permet de créer les objets par étape.

On peut même simuler l'héritage de classe dans l'héritage d'objet en utilisant la méthode spéciale _unknown() de façon à accéder dans myObject_B aux méthodes de l'objet myObject_A comme si elles étaient des méthodes de myClass_B héritées de myClass_A.

Par exemple :

```
Public Function _unknown(...) As Variant
    Select Param.Name
        Case "getThisA1"
            Return myObjectProperty_A.getThisA1()
        Case "getThisA2"
            Return myObjectProperty_A.getThisA2()
    End Select
End
```

Les méthodes getThisA1() et getThisA2() sont des méthodes de l'objet myObject_A (myClass_A) qui deviennent ainsi des méthodes de l'objet myObject_B (myClass_B) comme si myClass_B héritait de myClass_A.

2.3 Modèle Événementiel

Parallèlement à l'aspect « objet » qui défini l'organisation du code, la partie « événementielle » permet elle de préciser sa dynamique à travers la définition des :
– Évènements (auxquels on souhaite que le programme réagisse)
– Gestionnaires d'évènements (code de réponse par évènement)
On distingue deux types d'évènements selon qu'ils sont émis par :

1. Les objets graphiques ou groupes d'objets graphiques (controls)
2. Les objets de calculs (et les classes associées)

On distingue par contre trois types de gestionnaires d'évènements selon le « type de communication » qu'il permet d'établir :

1. Répercuter les évènements provenants de l'interface utilisateur (UI) vers les objets de calcul (BO)
2. Répercuter les évènements provenants des objets de calcul (BO) vers l'interface utilisateur (UI)
3. Répercuter les évènements provenants des objets de calcul en amont (BO) vers les objets de calcul (BO) en aval (de la hiérarchie d'objets)

C'est le rôle du « Global Manager » (dans Gambas, le fichier de code associé à la forme principale) que de gérer les deux premiers types d'évènements. Par contre, les évènements de troisième type sont gérés directement dans les objets via des observateurs définis au sein des classes.

Le tableau 2.2 ci-dessous résume la situation :

Communication	Objets	Localisation	Observateur
UI -> BO	Graphiques	Global Manager	Non
BO -> UI	Calculs	Global Manager	Non
BO -> BO	Calculs	Classes de Calculs	Oui

FIGURE 2.2 – Typologie des Gestionnaires d'Événements

Dans le cas classique d'un objet myObject ou d'un control myControl ayant un évènement myEvent, le gestionnaire d'évènement sera défini dans le fichier de code du Global Manager.

Par exemple :

```
Private Sub myControl_myEvent()
    myModule1.UpdateThis()
End
```

Dans le cas où l'on souhaite qu'un objet aval (myObject2 de type myClass2) puisse intercepter un évènement émis par un objet amont (myObject1 de type myClass1), il est nécessaire de passer par un observateur d'évènement implémenté dans le code de myClass2.

Par exemple :

```
' Code myClass1 (Amont)
Event Recalculated
Public Sub Calculate() As Boolean
```

```
    Me.Recaculate()
    Raise Recalculated
End
' Code myClass2 (Aval)
Private myObject1 As myClass1
Private myObject1Obs As Observer
Public Sub _new(myObject10 As myClass1, ...)
    myObject1 = myObject10
    myObject1Obs = New Observer(myObject10, True) As "Obs"
...
End
Public Sub Obs_myEvent1()
    Me.DoThis()
End
```

Le graphique 2.3 ci-dessous résume la dynamique événementielle du calculateur :

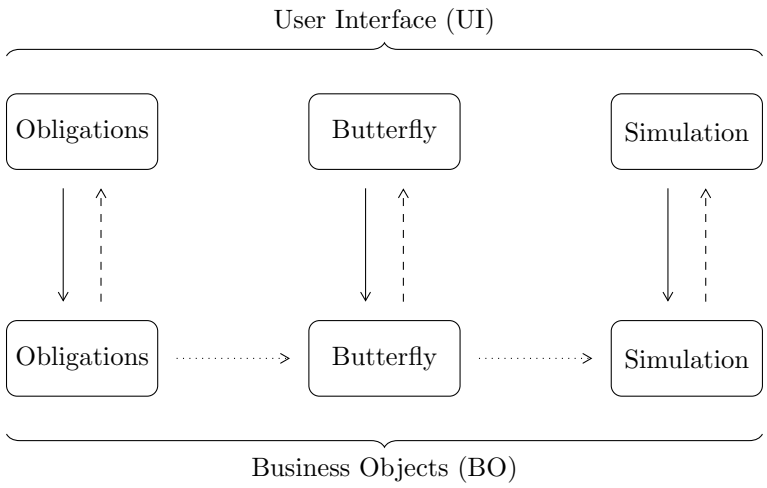

FIGURE 2.3 – Modèle Événementiel

Dans ce graphique :
- Les flèches en traits pleins représentent des évènements provenants de l'interface utilisateur (UI) vers les objets de calcul (BO)
- Les flèches en traits pleins discontinus représentent des évènements provenants des objets de calcul (BO) vers l'interface utilisateur (UI)
- Les flèches en traits pointillés représentent des évènements provenants des objets de calcul (BO) en amont vers les objets de calcul (BO) en aval

3 Autres Aspects (Hors Calcul)

Le calculateur sera aussi doté de trois autres blocs d'éléments graphiques :

– Un bloc de gestion du scénario de base
– Un bloc de notification (messages)
– Un bloc graphique (courbe des taux et position de butterfly)
Enfin, le calculateur sera doté d'un fichier de configuration.

3.1 Gestion du Scénario de Base

Rappelons qu'un « scénario » est un n-uplet de valeurs admissibles pour les inputs du calculateur.

On distingue trois scénarios :

1. Le scénario « courant » actuellement pris en compte par le calculateur (RAM)
2. Le scénario de « base - run time » (RAM)
3. Le scénario de « base - initialisation » (ROM/RAM)

A l'ouverture du calculateur (initialisation) ces trois scénarios sont identiques.

Le bloc de gestion du scénario de base est un ensemble de quatre contrôles :
– [Button] Sauver le scénario « courant » comme scénario de « base - run time »
– [Button] Revenir au scénario de « base - run time »
– [Button] Revenir au scénario de « base - initialisation »
– [Checkbox] Sauver le scénario de « base - run time » comme scénario de « base - initialisation » à la fermeture du calculateur

3.2 Notifications

Un bloc de notification permettra d'indiquer l'action réalisée par l'utilisateur via l'un des contrôles actifs de l'interface (inputs). Il s'agit d'une zone de texte (Label) dans laquelle un message explicite s'affichera au moment de l'action permettant de confirmer que l'action a bien été prise en compte et d'en préciser la nature de façon explicite.

Par exemple, si l'utilisateur fait passer le prix de l'obligation Left de 100.00% à 101.00%, le message sera :

Hausse de 100.00% à 101.00% du prix de l'obligation Left

Le message sera affiché un certain temps puis disparaîtra avec un effet de « fading ».

La technique utilisée consiste à utiliser deux Timer :

1. Le premier contrôle l'affichage du message
2. Le second contrôle l'effet de « fading »

Le premier timer activant le second.

Pour plus d'information, merci de vous reporter à cette discussion :

http ://www.gambasforge.org/sujet-4294-colorrgb-gbsdl-page-1.html

3.3 Graphique

Un bloc graphique (DrawingArea/Paint) permettra de donner une représentation graphique de la courbe des taux actuariels constituée par les 3 obligations ainsi que de la position de butterfly en distinguant la position réelle (Left/Center/Right) de sa représentation synthétique (Bullet/Barbell).

Le graphique sera réactualisé automatiquement lors d'une modification des inputs des blocs Obligations et Butterfly.

Au niveau du code, ce graphique impliquera :
- Une classe (clsChart) qui contiendra une méthode Draw() permettant de (re-)dessiner le graphique lorsque c'est nécessaire. Le constructeur de cette classe permettra d'initialiser les principales variables nécessaires à la création du graphique ainsi que la zone (DrawingArea) dans laquelle le dessin sera affiché. Cette classe utilisera la classe native Paint pour réaliser les différentes actions nécessaires à la création du dessin
- Un module (modChart) qui contiendra une méthode de création de l'objet graphique basé sur clsChart (à l'ouverture du calculateur) et une méthode de mise-à-jour (UpdateChart()) du graphique lorsque nécessaire (à partir de l'objet de dessin existant)

La création de l'objet de dessin et l'appel de la méthode UpdateChart() seront réalisés dans le fichier frmButterfly.class (Global Manager).

3.4 Fichier de Configuration

Ce fichier de configuration contiendra deux types d'informations :

1. Le scénario de « base - initialisation » qui permet l'initialisation du calculateur lors de son lancement
2. Les constantes de langues correspondants aux différents messages à afficher dans le bloc de notification

Seul le scénario de « base - initialisation » sera modifiable via l'interface graphique (bloc de gestion du scénario de base).

Le code source est téléchargeable sur le site http ://www.gambasforge.org/ à cette adresse : http ://www.gambasforge.org/code-89-butterfly-pricer.html