CHAPITRE I

DEMARCHE ET MISE EN ŒUVRE

Le développement de fonctions mathématiques peut répondre à plusieurs critères ou objectifs tels que la vitesse d'exécution, la précision ou la pédagogie. VBA n'a pas été conçu pour rivaliser avec Fortran ou le langage C sur des calculs scientifiques lourds, et le parti pris pour la constitution de cette (petite) bibliothèque consiste à privilégier systématiquement les algorithmes basiques et la clarté du code.

L'objet n'est pas de compliquer l'écriture des programmes au profit de la vitesse d'exécution, même si des principes d'optimisation du code et du calcul seront mis en œuvre. On trouvera donc ici des procédures et fonctions limitées en taille pour en conserver une vue globale et compréhensible (c'est le souhait !), plutôt que des pages de code beaucoup plus performant, mais illisibles.

Il faut garder à l'esprit que le temps de développement d'une application informatique n'est pas négligeable. Les programmes courts permettent de réduire considérablement cette phase et ainsi d'obtenir un coût global satisfaisant, dans la mesure où les performances de calcul ne sont pas trop sacrifiées.

La présente bibliothèque est essentiellement constituée de procédures **Function()** qui manipulent des valeurs de tous types et beaucoup de tableaux, la plupart de ces valeurs étant transférées comme arguments. Afin de limiter les erreurs de types, et de faciliter l'utilisation, certains choix ont été effectués pour ce qui concerne la structure générale des procédures, les types de variables et la forme des valeurs renvoyées.

Ce premier chapitre présente donc les conventions d'écriture et les conditions pratiques de mise en oeuvre des fonctions proposées, en commençant par détailler les avantages et inconvénients du couple "Excel+VBA" pour la création de fonctions mathématiques.

1- EXCEL ET VBA POUR LE DOMAINE SCIENTIFIQUE

1.1- Excel dans le domaine scientifique et technique

Sur le critère du nombre d'utilisateurs, Excel est sans doute le logiciel scientifique le plus répandu dans le monde ! La pratique montre en effet qu'Excel est très utilisé dans les entreprises, les écoles et les universités pour des applications scientifiques et techniques. Une recherche même sommaire sur Internet permet de s'en convaincre, et si cela ne suffisait pas, on peut noter qu'un logiciel scientifique de référence tel que Matlab® propose des passerelles d'accès pour travailler directement avec Excel.

Dès que l'on souhaite développer des applications scientifiques, il faut disposer d'une bibliothèque mathématique et Excel dispose d'un arsenal conséquent de fonctions de calcul. De plus, Excel propose en standard des outils extrêmement efficaces pour le domaine scientifique et l'ingénierie tels que l'outil "*Valeur cible*" et les macros complémentaires "*Solveur*" ou "*Utilitaire d'Analyse*", sans oublier les remarquables possibilités de calcul autorisées par le dispositif des références circulaires volontaires. Ces atouts très efficaces sont souvent ignorés de beaucoup d'utilisateurs, et peu d'entre eux savent, par exemple, qu'Excel propose en version standard un outil qui réalise la transformée de Fourier rapide (FFT). Enfin de nombreuses bibliothèques publiques ou commerciales permettent encore d'augmenter ses capacités de calcul.

Pour la création d'applications scientifiques, il faut toutefois considérer plusieurs inconvénients qui limitent les capacités ou l'utilisation de ces fonctions et outils :

- Certaines fonctions ne sont pas documentées en ligne sur le plan de la méthode mathématique mise en oeuvre. Ainsi, la fonction d'inversion de matrice renvoie une matrice inverse, mais on ne sait pas comment Excel procède pour y arriver, ce qui ne permet pas de garantir le résultat selon le conditionnement du système.
- Beaucoup de fonctions d'Excel sont limitées par la taille des arguments qu'elles acceptent. Par exemple, on ne peut pas transposer une matrice de plus de 5460 éléments ou inverser une matrice de taille supérieure à 52.
- Même si l'ensemble des fonctions de feuilles d'Excel est relativement vaste, il reste principalement orienté vers les statistiques et la finance, et plusieurs domaines ne sont pas couverts comme, par exemple, le calcul d'intégrales ou le traitement des équations différentielles.
- En pratique, l'utilisation des formules ne permet pas de réaliser de dispositifs algorithmiques efficaces comme les boucles ou la manipulation de tableaux.

1.2- VBA dans le domaine scientifique et technique

On admet que les langages destinés au calcul scientifique sont principalement le Fortran et le C, même s'il existe encore beaucoup d'utilitaires en Basic ou Pascal. Si l'on sort du domaine des logiciels de calcul généralistes fonctionnant sur stations de travail et que l'on rencontre dans la modélisation de structures ou d'écoulements (codes éléments finis associés à des mailleurs), il existe une foule d'applications pour les entreprises, la recherche ou l'enseignement, qui relèvent parfaitement de VBA sous Excel. Sans rechercher des micro-ordinateurs particulièrement puissants, cette association est adaptée à la simulation de systèmes industriels, aux calculs normalisés (DTU, Euro-Codes, ...) ou à l'enseignement de la physique, des mathématiques numériques et de l'ingénierie.

En fait, VBA permet de réaliser les calculs comme n'importe quel autre langage et la traduction du code scientifique d'un langage à un autre est assez facile, car les instructions purement calculatoires sont pratiquement identiques (affectations, opérations élémentaires, boucles, branchements conditionnels, …). Il y a donc peu d'efforts à faire de ce côté, et une procédure VBA pour résoudre un système d'équations, ressemblera énormément à sa cousine en Fortran ou en Pascal.

En tant que langage de programmation scientifique, VBA présente toutefois plusieurs limitations ou inconvénients :

- Vitesse d'exécution : Cet inconvénient est intrinsèque par rapport aux langages compilés. Les programmes VBA nécessiteront donc une machine assez performante pour fonctionner confortablement.
- Absence de bibliothèque mathématique en standard : VBA n'en possède pas directement et se trouve même assez pauvre en fonctions mathématiques de base.
- Portabilité réduite : VBA ne peut pas créer de version exécutable qui fonctionnerait seule sous un système d'exploitation tel que Windows®, DOS® ou Unix®. En pratique, les programmes VBA ne pourront pas être utilisés sans l'application sous laquelle ils ont été créés, c'est à dire Excel pour les fonctions proposées ici.
- Avenir incertain : VBA dépend de MS-Office, et comme tous les langages de Microsoft®, son avenir est plus ou moins dégagé, surtout face aux langages en liaison avec Internet. Depuis la version 2007, les barres de menu d'Excel ne sont plus programmables directement par VBA, et VBA lui-même n'est plus intégré à Office pour Mac®, même si un retour semble possible dans une future version ...

1.3- Le couple Excel+VBA dans le domaine scientifique et technique

En tant qu'environnement de programmation scientifique, l'association de VBA et d'Excel présente de très nombreux avantages :

- Facilité d'apprentissage : Le principal avantage de VBA est sa simplicité. Il permet au non-spécialiste, après un rapide apprentissage, de réaliser des applications finalement assez évoluées, tout en restant dans un environnement familier puisque "tout le monde a Excel sur son PC". Malgré ses limitations, c'est un excellent tremplin vers un langage plus performant.
- Bibliothèque mathématique : Cité comme inconvénient pour VBA en tant que langage isolé, l'association de VBA avec Excel permet d'utiliser dans le code les fonctions de feuilles de calcul, et elles sont nombreuses ! Par ailleurs, et c'est l'objet de cet ouvrage, on peut assez facilement fabriquer des fonctions de calcul personnalisées. Enfin, le développement de VBA comme langage scientifique a suscité la création de plusieurs bibliothèques dans divers domaines scientifiques ou d'ingénierie (mathématiques, chimie, génie climatique, finances, thermodynamique, mécanique, ...).
- Possibilités graphiques : VBA bénéficie des outils graphiques d'Excel pour convertir immédiatement les résultats de calculs en graphiques divers avec des outils d'analyse intégrés, sans programmer une seule ligne de code.
- Utilisation des feuilles de calcul d'Excel : Associées à VBA, les feuilles de calcul sont des endroits privilégiés et disponibles pour saisir des données, récupérer et stocker des résultats, jouer le rôle de base de données, sans la lourdeur d'une gestion de fichiers (fichier texte, délimité ou non, séquentiel, …).
- Création d'interfaces utilisateurs : VBA permet d'écrire des interfaces à l'aspect "professionnel" avec une très grande économie de moyens et présente un avantage décisif à ce niveau sur tous les autres concurrents au vu de l'importance

de la saisie et de la gestion des données pour un logiciel de calcul. Il est très facile de créer des masques de saisie avec listes déroulantes, boutons d'option ou compteurs bornés. Jusqu'à la version 2005, et avec très peu d'instructions, on peut créer des barres d'outils personnalisées pour les différents utilisateurs.

- Coût global de programmation : Une application scientifique n'est pas constituée que de code calculatoire et des critères comme le temps de développement ou la gestion de l'interface utilisateur sont souvent prépondérants. Dans ce cadre, VBA offre l'avantage de débarrasser le programmeur de presque toutes les contraintes "hardware", puisque Excel s'en charge. On peut alors se concentrer sur les aspects scientifiques et techniques sans être pollué, par exemple, par le contrôle de l'imprimante ou l'écriture d'un module de gestion graphique.
- Gratuité du langage : Il n'y a aucun achat supplémentaire, car le langage VBA est fourni en standard avec Excel, et il n'y a aucun droit à reverser en cas de distribution éventuelle de l'application réalisée.
- Enregistreur de macros : L'enregistreur de macros est d'abord un excellent professeur pour apprendre à programmer les objets d'Excel. Il enregistre les actions de l'utilisateur et les convertit en procédures VBA, c'est-à-dire en blocs de code que l'on peut ensuite consulter et analyser pour comprendre le mode de programmation des objets, et utiliser ensuite. Dans beaucoup de cas, il permet une grande économie de temps dans la programmation des interfaces.
- Communauté d'utilisateurs : Une recherche rapide sur Internet permet de constater que VBA est très utilisé par les entreprises et beaucoup d'universités du monde entier qui proposent des cours de mathématiques, de physique ou d'ingénierie en se basant sur VBA et Excel. Des sites de développeurs professionnels ou d'utilisateurs passionnés proposent des fonctions, trucs, astuces et foires aux questions, qui témoignent d'un activité régulière et soutenue (voir en partie A, le chapitre X).

***Remarque 1** : Il est important de noter qu'une bonne maîtrise des fonctionnalités d'Excel est souhaitable pour programmer en VBA sous Excel, sans quoi on risque de perdre beaucoup de temps à programmer des dispositifs déjà existants.*

***Remarque 2** : Une règle absolue respectée pour la conception de toutes les fonctions créées et utilisées dans cet ouvrage, est que seules sont utilisées les possibilités standard d'Excel 2000, à l'exclusion de tout utilitaire ou macro complémentaire (add-in) externe.*

2- CONVENTIONS D'ECRITURE DE LA BIBLIOTHEQUE

2.1- Types de variables et de procédures

Même s'il occupe plus de mémoire que les types numériques traditionnels, l'usage du type **Variant** est à peu près généralisé pour les variables accessibles à l'utilisateur, en particulier pour ce qui est passé entre les fonctions ou procédures.

Ce type est en effet pratiquement obligatoire pour passer des tableaux en arguments à des fonctions et surtout pour les récupérer avec ou sans modification des dimensions.

Lorsqu'une fonction renvoie un tableau ou un vecteur, la procédure fournie en illustration écrit le plus souvent ce tableau ou vecteur sur une feuille de calcul Excel, et le type Variant permet alors le transfert en une seule instruction, ce qui favorise également la clarté du code. Comme le type Variant peut contenir n'importe quel autre type de variable, les erreurs de transferts d'arguments sont limitées, et l'utilisateur peut plus facilement utiliser un réel de type simple ou double sans risquer de conflit de type.

Par contre les arguments et paramètres d'utilisation typique ou évidente seront totalement définis : La taille d'un système d'équations sera un nombre entier de type "Integer", alors qu'un critère de convergence sera un réel de type "Double". De leur côté, les variables secondaires et variables intrinsèques aux fonctions comme les compteurs de boucles par exemple, sont toujours définies avec un type donné.

L'option "Explicit" qui impose la déclaration explicite des variables est systématiquement activée. En effet, elle favorise la clarté du code et élimine les erreurs générées par toute nouvelle variable qui résulterait d'une faute de frappe.

A part quelques exceptions à but pédagogique ou démonstratif, le type de procédure **Function()** est privilégié dans toute la bibliothèque.

Toutes les procédures de type **Function()** proposées dans cette bibliothèque mathématique sont conçues pour être utilisables indifféremment comme des fonctions de feuilles sur des plages de cellules sur des feuilles de calcul Excel et comme des fonctions à l'intérieur de programmes VBA.

A cause de cette importante caractéristique, les arguments tableaux reçus et/ou renvoyés par les fonctions peuvent être des tableaux VBA ou des plages. Le type Variant est donc utilisé dans le code des fonctions pour manipuler ces arguments en évitant au maximum les conflits de types.

2.2- Notations

Les noms de variables sont autant que possible signifiants, et par exemple, un nombre limite d'itérations sera supporté par une variable nommée "**NbLimiteItér**".

Dans toutes les procédures, les noms de variables commencent par des majuscules. Les noms des arguments passés sont toujours explicites et précédés de la lettre minuscule "**p**" pour indiquer leur statut de paramètre (argument). Dans une fonction, un argument qui représente un nombre limite d'itérations sera alors nommé "**pNbLimiteItér**".

De façon à distinguer les variables secondaires intrinsèques dans les fonctions de celles des procédures appelantes, une appellation différente est utilisée. Les compteurs de boucles des procédures appelantes sont souvent appelés **i** et **j**, alors que dans les fonctions, ils seront appelés par exemple **ii** et **jj**.

2.3- Plages et tableaux manipulés par les fonctions

Afin de pouvoir travailler de façon transparente avec les mêmes outils dans Excel et dans VBA, les fonctions personnalisées proposées ici acceptent de la même façon les plages de cellules et les tableaux VBA. De plus, à la manière d'Excel, la taille d'un tableau passé à une fonction n'est jamais précisée dans la liste des arguments. Bien qu'atypique en programmation scientifique, cette démarche est systématiquement choisie pour toute la bibliothèque mathématique.

En contrepartie d'un certain confort d'utilisation, ceci conduit à allonger un peu plus le code des fonctions, car elles doivent analyser les arguments reçus (type et taille) avant de les manipuler et d'effectuer les calculs.

Pour éviter les confusions et les erreurs de transfert, tous les tableaux passés en arguments aux fonctions ou attendus comme retours de fonctions doivent être déclarés en type "Variant" et redimensionnés en tableaux aux dimensions du problème traité.

Par exemple, la fonction personnalisée **MatInv**() pour l'inversion de matrice, ne demande qu'un seul argument qui est la matrice carrée à inverser sans précision de taille. Dans le programme d'appel, la matrice à inverser doit donc être déclarée en type Variant, puis redimensionnée avec l'instruction **ReDim** à la taille souhaitée :

```
' Déclaration de [A] et de son inverse [AInv] avec le type Variant
Dim A  As Variant, AInv  As Variant

' Les variants A et AInv sont redimensionnés en tableaux
' avec la taille nécessaire (ici 70*70)  et un type adapté (ici réels de type Double)
ReDim A(1 To 70, 1 To 70) As Double
ReDim AInv(1 To 70, 1 To 70) As Double

' Remplissage de A
A(1, 1) = …
< … >

' Appel de la fonction d'inversion et affectation de la solution
AInv = MatInv(A)

' Utilisation de l'inverse
< … >
```

La fonction **MatInv**() peut également s'utiliser manuellement dans Excel comme une fonction de feuille en sélectionnant une plage de même taille que la matrice carrée [A], et en tapant la formule suivante (ici, "PlageA" est l'adresse ou le nom de la plage de cellules qui contient la matrice à inverser) :

= **MatInv**(PlageA)
Valider avec la combinaison de touches [Ctrl+Maj+Entrée]

Lorsque les tableaux à passer en arguments sont des vecteurs contenant peu de valeurs, il est possible de transférer directement ces valeurs dans les formules. Par exemple, on veut passer un vecteur de quatre valeurs (0, 1, 3, 2) comme initialisation pour la

résolution d'un système de 4 équations non linéaires avec la méthode de Broyden n° 2 présentée au chapitre A.III. La démarche standard avec un tableau de valeurs initiales est la suivante selon que l'on travaille dans Excel ou dans VBA :

Dans Excel : Ecrire les 4 valeurs initiales dans une plage de 4 cellules en ligne ou en colonne. Sélectionner ensuite une plage de 4 cellules pour la solution, et taper la formule suivante :

= **SysNonLinBM2**("VectF"; PlageIni; 50; 0,0000001, PlageParam)
Valider avec la combinaison [Ctrl+Maj+Entrée]
VectF() est la fonction VBA qui renvoie le vecteur 1D des 4 équations
PlageIni est le nom ou l'adresse de la plage des valeurs initiales
50 est le maximum d'itérations et le critère de convergence est ε = 0,0000001
PlageParam est une plage de paramètres (argument optionnel)

Dans VBA :

```
Dim VXini As Variant                ' Type Variant
Redim Vxini(1 To 4) As Double       ' Variant redimensionné
Vxini(1) = 0
Vxini(2) = 1
Vxini(3) = 3
Vxini(4) = 2

' Appel de la fonction de Broyden n° 2   SysNonLinBM2()
VXsol = SysNonLinBM2("VectF", VXini, 50, 0.0000001, TableauParam)
```

Avec 4 valeurs seulement, il est possible d'éviter la manipulation d'une plage ou d'un tableau d'initialisation avec les syntaxes suivantes :

Dans Excel : Sélectionner une plage de 4 cellules pour la solution, et taper :

= **SysNonLinBM2**("VectF"; **{0.1.3.2}**; 50; 0,0000001, PlageParam)
Valider avec la combinaison [Ctrl+Maj+Entrée]

Si le séparateur décimal d'Excel est le point, cette syntaxe devient :

= **SysNonLinBM2**("VectF"; **{0\1\3\2}**; 50; 0,0000001, PlageParam)
Valider avec la combinaison [Ctrl+Maj+Entrée]

Dans VBA :

```
' Appel de la fonction de Broyden n° 2   SysNonLinBM2()
VXsol = SysNonLinBM2("VectF", Array(0, 1, 3, 2), MaxItér, Epsilon, TableauParam)
```

La même démarche peut également s'appliquer au tableau de paramètres. Dans Excel, il faut noter que cette syntaxe reste limitée au transfert de petits groupes de nombres, car une formule ne peut dépasser un total de 256 caractères.

Beaucoup de fonctions proposées dans cet ouvrage sont soumises à des limites de tailles de tableaux qui proviennent de différentes contraintes ou préoccupations :

- Certaines des fonctions personnalisées utilisent des fonctions de feuilles d'Excel. Par exemple, on trouve dans le chapitre A.III, une fonction **SysLinXL**() pour la résolution des systèmes d'équations linéaires, et qui est basée sur l'utilisation des fonctions d'Excel INVERSEMAT() et PRODUITMAT(). Or, une fonction de feuille d'Excel ne peut accepter un argument tableau VBA qui contient plus de 5458 valeurs (73^2 pour des matrices carrées), et toutes les fonctions personnalisées qui les utilisent seront soumises à cette contrainte. De plus, la fonction INVERSEMAT est une exception à cette règle avec une limite de 52^2. La fonction SysLinXL() ne pourra donc résoudre un système avec plus de 52 équations.

- Utilisée comme une fonction de feuille, une fonction personnalisée VBA ne peut pas renvoyer une plage qui contient plus de 5458 cellules. Par exemple, le chapitre A.III propose une fonction **MatInv**() pour l'inversion de matrice par la méthode de Jordan sans pivotage, qui n'utilise aucune fonction de feuille d'Excel. Théoriquement, elle n'est donc pas limitée en taille. Il est alors possible d'inverser une matrice de taille 200×200 si l'argument est un tableau VBA ou une plage Excel, **ET QUE** le résultat (un tableau VBA de taille 200×200) reste utilisé dans le code, même si son contenu est ensuite transféré sur une feuille de calcul Excel. Par contre, si la fonction MatInv() est appliquée comme une formule de tableau sur une plage de cellules sur une feuille Excel, comme montré dans le paragraphe précédent, la matrice inversée ne pourra pas dépasser l'ordre 73.

- Tous les calculs effectués dans un calculateur électronique sont soumis à une précision limitée à cause des erreurs de troncatures, même en utilisant des déclarations de type "double précision". Si l'on reprend l'exemple de la fonction MatInv() du chapitre A.III, il est théoriquement possible d'inverser une matrice de taille 500×500 ou plus, si le tableau résultat reste manipulé en mémoire par le code. Toutefois, outre un temps de calcul qui devient prohibitif, les erreurs d'arrondis peuvent conduire à un résultat "inattendu" selon le conditionnement de la matrice. Comme MatInv() ne fixe pas de limite arbitraire d'utilisation, il relève de l'entière responsabilité de l'utilisateur de restreindre l'application de cette fonction à l'inversion de matrices d'une taille raisonnable, et dans tous les cas, d'effectuer les contrôles nécessaires pour valider les résultats renvoyés.

2.4- Utilisation de la méthode Run dans VBA

Pour plusieurs des méthodes numériques abordées dans cet ouvrage, les fonctions proposées appellent une fonction externe F(x,{P}) parmi leurs arguments. C'est le cas par exemple pour la résolution de F(x,{P}) = C, l'intégration de F(x,{P}) de x_a à x_b ou la résolution de systèmes d'équations différentielles ordinaires.

Pour appeler cette fonction à résoudre ou à intégrer, deux possibilités existent :