MASTER EN SCIENCES INFORMATIQUES

Systèmes experts en statistique

Winandy, Vincent

Award date:
1988

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX NAMUR

Institut d'Informatique

SYSTEMES EXPERTS

EN STATISTIQUE

Mémoire présenté par

WINANDY VINCENT

2e année de la
Maîtrise en Informatique

Année académique 1987-1988

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX NAMUR

Institut d'Informatique

Rue Grandgagnage, 21, B-5000 NAMUR, Tel (081)22.90.65

SYSTEMES EXPERTS EN STATISTIQUE : RESUME

Les systèmes informatiques en statistique sont de plus en plus utilisés par des personnes n'ayant qu'une faible formation en statistique. Pour éviter des utilisations erronées, des systèmes plus intelligents seraient souhaitables. Dans ces "Systèmes Experts" (S.E.) des connaissances statistiques sont incorporées, sous la forme de stratégies, pour fournir une assistance à l'utilisateur.

Nous présentons ici, une synthèse des idées actuelles à propos de l'utilité, la conception et les directions futures de ces S.E. en Statistique (S.E.S.) Nous avons également développé un S.E.S. capable d'aider un utilisateur lors de l'analyse d'une série chronologique selon la méthode BOX-JENKINS. Nous décrivons de manière détaillée les caractéristiques, la stratégie et l'architecture du système.

EXPERT SYSTEMS IN STATISTICS : ABSTRACT

Statistical computer programs are becoming increasingly accessible to people with limited statistical training. To prevent misuse, more intelligent software is clearly needed. In these, so called, Expert Systems (E.S.), statistical knowledge is embedded, in the form of strategies, to provide guidance, explanations and other features.

We present here a summary of current ideas about the usefulness, the design and the future directions of these Statistical E.S. (S.E.S.) We also built a S.E.S. able to help some user perform time series analysis by using BOX-JENKINS method. A detailed description of the characteristics, the strategy and the architecture of the system is given.

Promoteur du mémoire
Madame le Professeur
M. NOIRHOMME-FRAITURE

Mémoire présenté par
WINANDY Vincent
Deuxième année de la
Maîtrise en Informatique

Année Académique 1987-1988

AVANT - PROPOS

Je tiens à remercier Madame le Professeur **M. NOIRHOMME-FRAITURE**, Promoteur du mémoire, non seulement pour les conseils qu'elle m'a prodigués, mais aussi pour la confiance qu'elle a bien voulu me témoigner au cours de cette année.

J'exprime aussi toute ma gratitude aux personnes qui ont accepté de me recevoir et qui, par leurs compétences ou leurs avis, m'ont apporté une aide précieuse. Je pense notamment à Monsieur **G. LIBERT**, Docteur en Sciences Appliquées de la Faculté Polytechnique de Mons, dont les recherches ont été d'une grande utilité pour ce travail.

Enfin, je ne voudrais pas oublier tous ceux qui, aux cours de ces dernières années, ont, à quelque titre que ce soit, contribué à ma formation.

TABLE DES MATIERES

ANNEXES

A1. LA METHODE BOX - JENKINS

A2. IMPLEMENTATION DE LA STRATEGIE D'IDENTIFICATION

A3. DOSSIER CONCEPTUEL DU S.E.B.J.

REFERENCES BIBLIOGRAPHIQUES

INTRODUCTION

Ces quelques pages d'introduction présentent le sujet de mémoire et son traitement. Elles décrivent également le contenu du rapport écrit pour en faciliter la lecture.

1. LE SUJET DE MEMOIRE : LES SYSTEMES EXPERTS EN STATISTIQUE

"We need to make statistical strategy available to more people to prevent misuse of statistical packages." [1]

En effet, les statisticiens ont fréquemment fait remarqué que les systèmes statistiques actuels risquaient d'être mal utilisés. Ils constituent de bons outils numériques, mais laissent à l'utilisateur toute liberté pour choisir une méthode statistique, l'appliquer et interpréter les résultats. L'utilisation correcte de ces systèmes nécessite une stratégie statistique, c'est-à-dire une approche cohérente face à une tâche d'analyse de données.
Pour mécaniser une stratégie statistique, on peut penser recourir aux techniques d'Intelligence Artificielle (I.A.) et donner ainsi naissance à des Systèmes Experts en Statistique (S.E.S.).

A l'heure actuelle, des S.E.S. ont été développés dans le cadre de projets de recherche. Ils visent notamment à démontrer la possibilité d'aider un utilisateur à sélectionner une technique statistique appropriée, à vérifier les hypothèses sous-jacentes à une technique. Mais ces S.E.S ne font pas l'unanimité parmi les statisticiens. De plus, leur développement entraîne des difficultés liées à l'établissement d'une "bonne" stratégie statistique et à sa représentation. C'est pourquoi, des travaux de recherche sont entrepris pour apporter des solutions à ces deux problèmes.

2. LE TRAITEMENT DU SUJET

Ayant pour objectif général l'étude des S.E.S., nous avons procédé en deux étapes, l'une théorique et l'autre pratique.

Dans un premier temps, nous avons rassemblé le maximum de documents concernant les S.E.S. et nous en avons fait une synthèse. Cela nous a permis de comprendre ce qu'il fallait attendre de ces systèmes et d'étudier certaines réalisations. Mais, traitant un sujet nouveau et contreversé, ces documents ne nous ont pas toujours paru convaincants.

C'est pourquoi, dans un second temps, nous avons décidé de réaliser un S.E.S. Nous lui avons assigné un objectif limité : guider un utilisateur lors de l'analyse d'une série chronologique selon la méthode BOX - JENKINS (B&J). Nous l'avons baptisé S.E.B.J. Cette étape nous a permis d'avoir une opinion sur le sujet qui est à la fois plus nuancée et plus personnelle qu'au terme de la première étape. Elle nous a également permis d'acquérir une expérience supplémentaire dans le développement de logiciels.

3. LE CONTENU DU RAPPORT

Le rapport a été conçu de manière à refléter notre démarche de travail et à présenter l'information au lecteur selon plusieurs niveaux de détails.

Le corps du rapport comporte deux parties correspondant aux deux étapes décrites ci-dessus : la synthèse de l'étude théorique et la réalisation du S.E.B.J. Des annexes fournissent des compléments d'information de type théorique et méthodologique.

Le tableau suivant fournit un guide pour une lecture efficace du rapport.

Corps du rapport	Compléments d'information
PREMIERE PARTIE : ETUDE THEORIQUE Elle donne une synthèse de ce qui a été publié concernant l'opportunité de développer des S.E.S., leur faisabilité et les réalisations et travaux de recherche actuels. **DEUXIEME PARTIE : DEVELOPPEMENT D'UN S.E.B.J.** Après un bref rappel des principes de la méthode B&J, elle décrit chaque étape du développement du S.E.B.J. : les spécifications fonctionnelles, l'explicitation l'implémentation et la validation de la stratégie. **CONCLUSIONS**	**ANNEXE 1 : LA METHODE BOX - JENJINS** Elle fournit une synthèse détaillée sur les principes théoriques de la méthode B&J et sur son utilisation. **ANNEXE 2 : IMPLEMENTATION DE LA STRATEGIE** Elle précise un certain nombre de procédures clés pour la réalisation des fonctionnalités du S.E.B.J. **ANNEXE 3 : DOSSIER CONCEPTUEL** Elle donne les spécifications externes et internes de chaque module de l'architecture logicielle du S.E.B.J.

REFERENCES INTRODUCTION

[1] GALE W.A., Overview of Artificial Intelligence and Statistics, p 8

PREMIERE PARTIE

ETUDE THEORIQUE

INTRODUCTION

I. ANALYSE D'OPPORTUNITE

II. ANALYSE DE FAISABILITE

III. SUJETS DE RECHERCHE

INTRODUCTION

Nous avons consacré les premiers mois de travail à une recherche documentaire. Cette partie constitue une synthèse de nos investigations.

Cette recherche avait pour but de mieux cerner un sujet dit "à la mode" mais qui, aux yeux de beaucoup, reste mal défini. Nous voulions en particulier connaître les motivations et les attentes des personnes intéressées par le sujet, mais, en tant qu'informaticien, nous voulions surtout recueillir des informations d'ordre conceptuel : les fonctionnalités, les choix d'architecture et d'implémentation de ces Systèmes Experts en Statistique.

Nous tenons à souligner deux caractéristiques des documents consultés. D'abord, il existe de nombreuses divergences entre les auteurs, ensuite, les articles décrivant avec précision des réalisations sont extrêmement rares. Cela est compréhensible puisque le sujet est nouveau, mais cela ne facilite pas un travail de synthèse qui se veut objectif et cohérent.

Le lecteur trouvera dans cette partie trois chapitres :

- Le premier (I. Analyse d'opportunité) est une introduction aux S.E.S. En répondant à des questions générales telles que "Pourquoi un S.E.S. ?", "Pour qui ?", "Pour quoi faire ?", il permettra de définir ces systèmes.
- Le second (II. Analyse de faisabilité) aborde une question plus controversée "Comment réaliser un S.E.S. ?" Nous y répondrons en proposant une démarche de développement et en décrivant quelques réalisations intéressantes.
- Le troisième (III. Sujets de recherches) traite quelques projets de recherche en matière de développement de S.E.S.

I. ANALYSE D'OPPORTUNITE

Ce chapitre aborde quelques questions essentielles introduisant les S.E.S. :

- Pourquoi s'intéresser à ce sujet ?
- Quels sont les objectifs de ces systèmes ?
- A qui sont-ils destinés ?

Pour répondre à ces questions, nous commençons par identifier des besoins en analyse statistique, nous montrons ensuite que les systèmes actuels ne satisfont pas totalement ces besoins, ce qui nous conduit vers la description de systèmes "plus intelligents".

1. ETUDE DES BESOINS EN ANALYSE STATISTIQUE

Après avoir expliquer en quoi consiste une analyse statistique, nous dégageons la principale tendance constatée.

1.1. ANALYSE STATISTIQUE

Une analyse statistique peut généralement se décomposer en trois phases [1]:

- La conception : phase durant laquelle les objectifs sont établis, le problème formulé, une ou plusieurs méthodes statistiques choisies.
- La conduite : phase d'application de la (des) méthode(s) retenue(s) sur les données.
- L'interprétation : phase d'analyse des résultats.

Des retours en arrière sont possibles à l'issue de chaque phase.

Pour mener à bien ces trois étapes, des ressources humaines et informatiques sont nécessaires :

- Un expert en statistique intervient principalement lors de la conception de l'étude et lors de l'interprétation des résultats.
- Un système statistique est généralement utilisé pour conduire l'analyse.

1.2. TENDANCE

On constate un décalage entre le rythme de l'évolution des besoins en analyse statistique et le rythme d'accroissement des ressources humaines.

En effet, il apparaît clairement que des personnes provenant de domaines très variés (médecins, sociologues, ...) auront dans les prochaines années un volume d'informations à analyser sans cesse croissant. Cela est principalement dû à l'évolution des techniques d'acquisition de données digitales et à l'augmentation des capacités de stockage.

Parallèlement, le nombre d'experts en statistique semble rester constant.

Il devient dès lors nécessaire de donner à des personnes ayant peu de connaissances en statistique, la possibilité de concevoir, de conduire et/ou d'interpréter une analyse statistique avec une assistance humaine réduite.

Force est de constater toutefois, les dangers de cette mise à disposition :"Use of statistical techniques could be replaced by overuse and misuse"[2] car les systèmes actuels, comme nous allons le voir au point suivant, ne sont pas adaptés pour cette classe d'utilisateur.

2. ETUDE DES SYSTEMES STATISTIQUES ACTUELS

Après avoir défini ces sytèmes, nous en soulignons quelques défauts importants.

2.1. DEFINITION

Les systèmes actuels peuvent être définis comme une collection de programmes fournissant aux utilisateurs un ensemble de techniques ou méthodes statistiques. Ces programmes sont coordonnés et l'utilisateur spécifie des analyses à accomplir sur ses données, généralement à l'aide d'un langage de commandes [3].

Les composants et interfaces de ces systèmes peuvent être représentés de la manière suivante [4] :

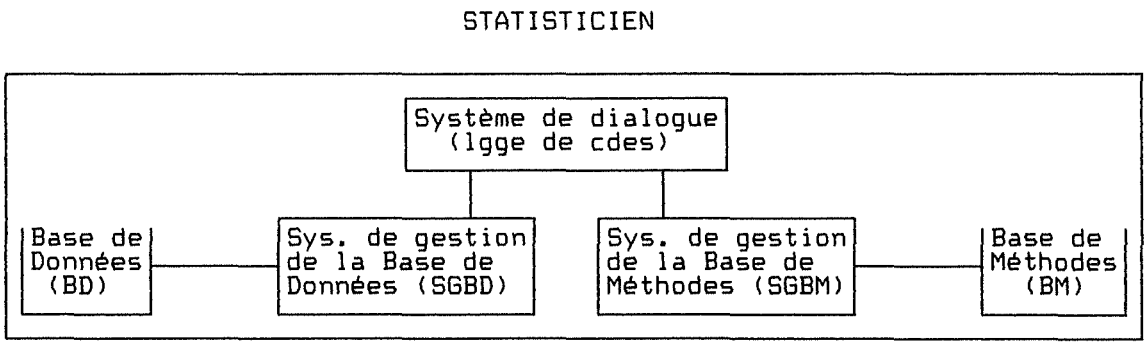


Figure I.1.: Composants et interfaces d'un système statistique actuel

Ces systèmes, dont les plus connus sont SAS, BMDP, SPSS, ... sont capables de réaliser des analyses très sophistiquées (la base de méthodes est fort développée). Ils remplacent avantageusement les deux générations de software qui les ont précédés : la première approche fut d'écrire de simples programmes ne réalisant qu'une technique statistique bien déterminée. Dans une seconde approche, l'utilisateur disposait d'une collection d'algorithmes qu'il lui fallait encore relier à l'aide d'un langage de programmation.

2.2. DEFAUTS

Durant des années, les statisticiens et programmeurs avaient comme principaux soucis d'améliorer la précision des calculs et de fournir le plus de méthodes statistiques possibles. Aujourd'hui, bien que les problèmes de précision soient toujours présents, d'autres préoccupations apparaissent. Outre des défauts

ergonomiques visibles, il semble que ces systèmes souffrent également de problèmes d'ordre architectural.

2.2.1. Les défauts ergonomiques

En matière d'ergonomie, certaines "règle d'or" sont maintenant généralement admises et des recommandations se dégagent [5]. Si l'on prend ces règles et recommandations comme critères d'évaluation ergonomique d'un système statistique, on relève essentiellement les défauts suivants :

1) Manque de cohérence

Chaque système propose un langage de commandes différent, ce qui est regrettable mais compréhensible. Par contre, des inconsistances au sein d'un même langage pourraient être évitées telles que, par exemple, des variations de signification d'une commande ou d'un terme d'une situation à une autre.

2) Pauvre gestion des erreurs

Les erreurs numériques (telles que l'inversion d'une matrice singulière) et syntaxiques (telles que l'omission d'un argument dans une commande du langage) sont détectées et signalées à l'utilisateur. Cette signalisation est généralement peu compréhensible et ne contient pas d'indication concernant les actions correctrices. Quant aux erreurs sémantiques, (telles que l'application d'un test dans des conditions violant les hypothèses propres à ce test, ou l'inversion d'une matrice presque singulière entraînant des problèmes de précision), elles ne sont pas détectées et donc pas signalées.

3) Importante charge de mémoire

La maîtrise d'un système nécessite un long apprentissage. Cela est dû au style de dialogue utilisé (langages de commandes), à la richesse de ces langages (nombre important de commandes, d'arguments et d'options) mais également au manque de cohérence signalé plus haut.
Heureusement, la plupart des systèmes offrent un système d'aide "online", mais celui-ci est souvent limité à la syntaxe du langage.

4) Autres défauts ergonomiques

Rares sont les systèmes évoluant vers des techniques récentes telles un dialogue par manipulation directe sur des objets interactifs (fenêtres, menus, icônes,...) à l'aide d'instruments de pointage (souris, ...).

De même, l'utilisation des représentations graphiques reste limitée.

Ces caractèristiques ergonomiques permettent de tracer un profil grossier de l'utilisateur typique d'un système actuel en fonction des connaissances requises [6].

Le tableau suivant donne pour trois types de connaissances (syntaxiques, sémantiques dans le domaine informatique et sémantiques dans le domaine statistique) le niveau requis. Il en ressort que les systèmes actuels sont destinés aux experts en statistique ayant aussi de bonnes connaissances en informatique.

		peu	beaucoup
SYNTAXE			X
SEMANTIQUE	Informatique		X
	Statistique		X

Tableau I.1. : Profil de l'utilisateur d'un système statistique actuel

Cette constatation se confirme lorsque l'on parcourt les articles décrivant ces systèmes. Ainsi, par exemple, la phrase "The latest version of ... contains many improvements to make it even more useful to statisticians. Enhancements include the calculation of ..."[7] montre bien à qui sont destinés les systèmes et dans quel sens vont les améliorations.

2.2.2. Problème d'architecture

Ces sytèmes sont de "gros logiciels", dont le développement nécessite des investissements lourds et dont la durée de vie est longue. Ils ont subi au cours de leur existence des modifications, essentiellement des extensions pour intégrer de nouvelles techniques statistiques.

S'il est vrai, comme le prétend CHAMBERS [8], que les concepteurs de systèmes statistiques ont montré scepticisme et ignorance envers certains principes de génie logiciel, alors on comprend le souhait de MULLER [9] de voir certains modules des architectures classiques profondément modifiés.

3. VERS DES SYSTEMES STATISTIQUES PLUS INTELLIGENTS

Tant que les systèmes statistiques étaient exclusivement utilisés par des experts en statistique, les défauts ergonomiques (et principalement l'incapacité de détecter les erreurs sémantiques) ne posaient pas trop de problèmes (voir 2.). Aujourd'hui ils sont fréquemment utilisés par des personnes ayant peu de connaissances en statistique (voir 1.). Des systèmes plus "intelligents" pourraient aider l'utilisateur afin de lui éviter certaines erreurs [10].

Partant d'un certain nombre d'améliorations souhaitables des sytèmes actuels, nous définissons, en termes de fonctions, une nouvelle génération de systèmes statistiques.

3.1. AMELIORATIONS SOUHAITABLES

Il faudrait bien sûr remédier aux défauts cités en 2.2. De plus, compte tenu des constats faits en 1.2., un effort de flexibilité nous semble également indispensable, d'autres améliorations sont souhaitables, nous les citerons rapidement.

3.1.1. Flexibilité

Un système sera flexible s'il offre plusieurs modes d'utilisation en fonction des caractéristiques de l'utilisateur. Partant d'une classification arbitraire des différents types d'utilisateurs, nous proposons dans le tableau suivant quelques fonctions à offrir à chacune des classes.

La classification retenue est la suivante :

- utilisateur naïf : ayant peu de connaissances syntaxiques à propos du système et peu de connaissances sémantiques tant en informatique qu'en statistique.
- expert intermittant : ayant des connaissances sémantiques mais peu de connaissances syntaxiques.
- expert fréquent : ayant tout type de connaissances.

UTILISATEUR	FONCTIONS OU CARACTERISTIQUES DU SYSTEME
Naïf	- Apprentissage : utilisation du système dans un mode tutoriel. - Assistance : système guidant l'utilisateur lors de son analyse, expliquant des concepts statistiques ou réalisant une analyse de manière automatique (voir 3.2.)
Expert intermittant	- Convivialité : réduction de la quantité de connaissances syntaxiques : menus de commandes, manipulation directe écrans d'aide.
Expert fréquent	- Puissance : large éventail de fonctions statistiques avec un langage de commandes riche. - Efficacité des algorithmes et de l'interface. - Souplesse : possibilité d'étendre le système en intégrant des routines que l'utilisateur aurait lui-même programmées, ou de choisir parmi plusieurs algorithmes

Tableau I.2.: Fonctions pour améliorer la flexibilité d'un système

3.1.2. Autres améliorations

Sans reprendre l'impressionnante liste de MULLER [11], on peut encore souhaiter :

- des améliorations communes à tout type de programmation telles que fiabilité, maintenabilité, portabilité, bonne utilisation des capacités des systèmes d'exploitation, ...
- des améliorations propres aux applications statistiques (qu'elles soient de nature répétitive ou exploratoire) telles que des facilités d'acquisition et de manipulation de données, des productions automatiques de rapports, des représentations graphiques, ...
- des améliorations concernant les configurations matérielles telles que des moyens de communication permettant une distribution des systèmes avec des postes de travail spécialisés.

3.2. DESCRIPTION DE SYSTEMES PLUS INTELLIGENTS

Face à la situation décrite aux points 1. et 2., UNE solution (parmi d'autres) POURRAIT être l'application des techniques d'Intelligence Artificielle pour développer des Systèmes Experts en Statistique. Les paragraphes qui suivent décrivent ces S.E.S. : une définition, des objectifs, des fonctionnalités et des contraintes sont proposées.

Un S.E.S. est un programme informatique capable de prendre en charge ou de supporter des activités qui sont spécifiques à un statisticien. Pour cela, il utilise des connaissances statistiques [12].

Les composants et interfaces de ces S.E.S. peuvent être représentés de la manière suivante [13] :

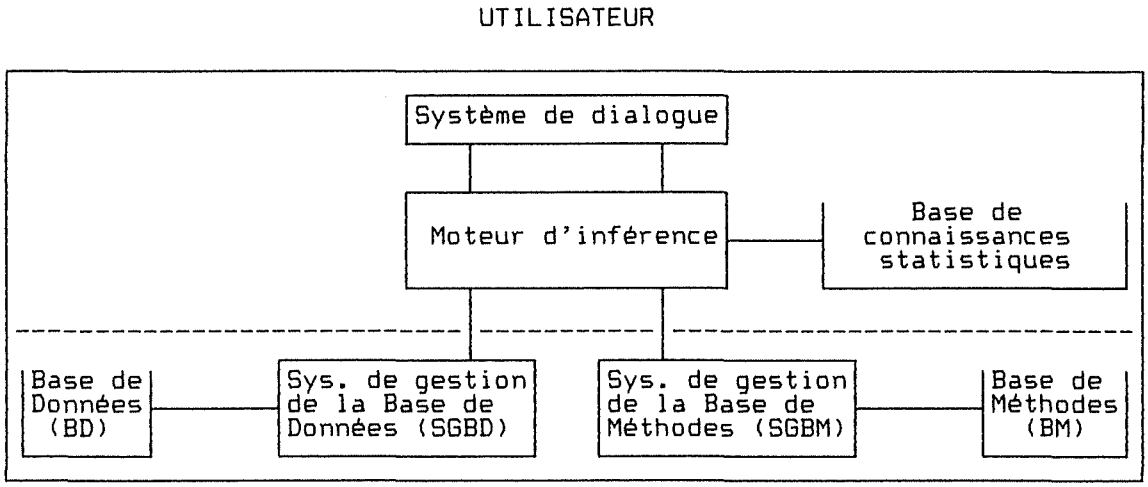


Figure I.2.: Composants et interfaces d'un S.E.S.

Comme pour tout autre système expert, les deux principales motivations des recherches en S.E.S. sont [14] :

- accumuler les connaissances d'un expert dans un système afin de les rendre disponibles à un utilisateur du système,
- acquérir et systèmatiser les connaissances d'un expert afin de mieux comprendre les mécanismes de raisonnement sous-jacents.

Les fonctionnalités possibles d'un S.E.S. peuvent être rangées en deux catégories correspondant aux deux objectifs mentionnés ci-dessus : des fonctions d'assistance, pour les différentes phases de l'analyse, et des fonctions d'acquisition de connaissances.

OBJECTIF	FONCTIONNALITES D'UN S.E.S.
Assistance	Phase de conception de l'analyse - Aider l'utilisateur à définir le problème, simuler une consultation statistique. - Choisir/suggérer un ou plusieurs modèles appropriés. - Choisir/suggérer une ou plusieurs méthodes statistiques Phase de conduite de l'analyse - Choisir/suggérer une suite d'actions à accomplir. - Choisir/suggérer des valeurs de paramètres. Phase d'interprétation des résultats - Expliquer la signification des résultats. - Choisir/suggérer des analyses complémentaires. Pour toutes les phases - Expliquer pourquoi et comment tel choix ou suggestion est fait. - Expliquer la signification d'un terme.
Acquisition de connaissances	- Observer le travail d'un statisticien et le systématiser - Construire à partir de cette observation une base de connaissances pour un système de consultation (voir la description du système STUDENT en III.2.)

Tableau I.3.: Fonctionnalités possibles d'un S.E.S.

Pour être véritablement une solution, un S.E.S. devra, en plus d'offrir les fonctionnalités citées, éviter les défauts des systèmes actuels (voir 3.1.) et notamment satisfaire les contraintes de flexibilité décrites en 3.1.1.

4. SYNTHESE

Les besoins croissants en analyse statistique ne pourront être satisfaits avec les ressources actuelles : les statisticiens restent peu nombreux et les software présentent certaines déficiences. Les S.E.S. constituent un nouveau type de ressources potentielles pour faire face à cette situation.

Ces S.E.S. seraient caractérisés par:

- l'objectif visé : assister un utilisateur et/ou acquérir des connaissances,
- les phases de l'analyse supportées : conception, conduite et/ou interprétation,
- l'étendue des méthodes statistiques considérées,
- les fonctionnalités proposées,
- le profil de l'utilisateur : naïf, expert intermittant, expert fréquent,
- les techniques d'implémentation.

REFERENCES CHAPITRE I

---

[1] HAND D.J. Patterns in Statistical Strategy
[2] SMITH A.M.R., LEE L.S., HAND D.J., Interactive User-friendly Interfaces to Statistical Packages, p 199
HOOKE R., Getting People to use Statistics properly
[3] CHAMBERS J.M., Statistical Computing : History and Trends, p 239
[4] WITTKOWSKI K.M., Generating and testing statistical hypotheses : strategies for knowledge engineering, p 142
[5] SCHNEIDERMAN B., Designing the User Interface
[6] Profil basé sur "The Syntaxic\Semantic Model of User Knowledge" décrit dans SCHNEIDERMAN B., Designing the User Interface, p 43
[7] LANE P.W., New Development in Genstat, p 263
[8] CHAMBERS J.M., Statistical Computing : History and Trends, p 240
[9] MULLER M.E., Aspects of Statistical Computing : What Packages for the 1980's Ought to Do
[10] WITTKOWSKI K.M., Generating and testing statistical hypotheses : strategies for knowledge engineering, p 140
[11] MULLER M.E., Aspects of Statistical Computing : What Packages for the 1980's Ought to Do, p 161
[12] HAUX R., Expert Systems in Statistics, p 3
STREITBERG B. A modestly intelligent system for identification, estimation and forecasting of univariate time series : A4 ARIMA, artificial intelligence, and APL2, p 184
[13] WITTKOWSKI K.M., Generating and testing statistical hypotheses : strategies for knowledge engineering, p 151
[14] OSTERMAN R. The use of expert systems in different fields of statistics : two examples, p 87

II. ANALYSE DE FAISABILITE

En supposant qu'il soit souhaitable de réaliser des S.E.S. tels que décrits au chapitre précédent, encore faut-il savoir s'il est possible de les réaliser et si oui, comment ?

On remarque souvent que les personnes impliquées dans un domaine de recherche peuvent se ranger dans l'une des catégories suivantes :

- les penseurs constructifs,
- les penseurs sceptiques ou destructifs,
- les réalisateurs.

Les recherches en S.E.S. n'échappent pas à cette règle. Par souci d'objectivité, nous avons reflété cette réalité dans la structure du chapitre : Nous commençons par donner quelques principes et recommandations concernant une démarche de développement, nous évoquons ensuite le débat pour ou contre opposant sceptiques et convaincus, nous terminons par la description et les enseignements à tirer de quelques réalisations intéressantes.

1. PRINCIPES ET RECOMMANDATIONS POUR DEVELOPPER UN S.E.S

Bien qu'aucune méthodologie n'existe, nous proposons une "démarche" de développement d'un S.E.S. en quatre grandes étapes : choix et spécifications, explicitation, implémentation et validation de la stratégie statistique.

1.1. CHOIX ET SPECIFICATIONS

Il est admis en I.A. qu'un système très général de résolution de problèmes n'est pas réalisable. Il faut restreindre son champ d'application, ce qui signifie pour un S.E.S. :

- Choisir quelle(s) phase(s) de l'analyse et quelle(s) méthode(s) seront supportées : dans le but d'éviter des conflits d'expertise, on suggère de rechercher celles où le degré de variabilité entre les approches des statisticiens est faible, ou encore celles pour lesquelles il existe une théorie solide, reconnue [1].

- Choisir un type d'utilisateur : malgré la contrainte de flexibilité (voir I.3.1.1.), la classe "utilisateur naïf" semble pour l'instant la plus intéressante. Il est cependant conseillé d'éviter les extrèmes afin de limiter la quantité de connaissances à considérer [2].

Ensuite, comme pour tout autre système logiciel, une étape importante consiste à spécifier les fonctionnalités du S.E.S. (voir I.3.2.).

1.2. EXPLICITATION DE LA STRATEGIE

1.2.1. Définition et objectifs

Une stratégie statistique peut être définie comme une description formelle des choix à faire, des actions à mener et des décisions à prendre pour utiliser des méthodes statistiques au cours d'une étude [3].
Plus généralement, la stratégie est le raisonnement utilisé par le statisticien [4].

La stratégie constitue une partie des connaissances de l'expert. Avant de les représenter, il est important de les expliciter. Selon les termes du statisticien J.TUKEY :"One just cannot build an expert system without thinking through a strategy"[5]. En I.A., cette phase est connue sous le nom "knowledge elicitation"[6].

Nous avons retenu trois approches pour décrire une stratégie, nous les avons baptisées respectivement "planification hiérarchique", "hiérarchie utilise" et "diagramme de transition" pour souligner leur similitude avec d'autres domaines connus en informatique [7].

1.2.2. Planification hiérarchique [8]

Si l'on considère la phase "conduite de l'analyse" pour des problèmes théoriques, alors celle-ci peut être vue comme un ensemble de tâches à effectuer, la plupart des tâches peuvent être décomposées en une séquence de sous-tâches, qui peuvent être à leur tour décomposées en séquences de sous-tâches, etc. Ce procédé conduit à une structure hiérarchique.

Ainsi par exemple, une représentation partielle de la tâche "régression linéaire" pourrait être [9] :

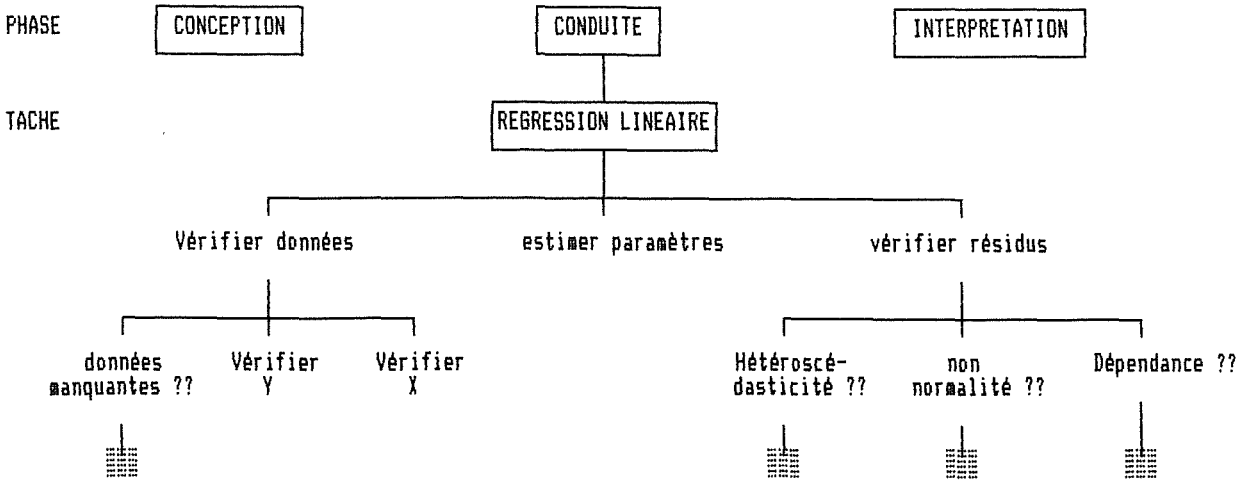


Figure II.1.: Explicitation de la stratégie : Planification hiérarchique

?? : Sous-tâche dont le développement est conditionné par le résultat d'une procédure d'évaluation. Ainsi par exemple, dépendance ?? sera développé en fonction des résultats d'un test d'autocorrélation sérielle (Durbin-Watson).

▒ : Noeud terminal indiquant ce qu'il faut faire dans cette situation :

- Soit l'expert peut donner une action systématique. Ainsi par exemple, si les résidus ne sont pas indépendants, l'expert indique qu'il faut ajouter une variable explicative au modèle (la variable dépendante "laggée").
- Soit il ne peut pas. Dans ce cas une aide humaine est nécessaire. On rejoint ainsi l'idée que tout S.E.S. devrait contenir la règle "SI ... ALORS appeler votre statisticien local" [10].

Signalons encore qu'il est conseillé de développer les sous-tâches de manière breadth-first et aussi loin que possible, c'est-à-dire jusqu'au moment où une sous-tâche décrite de manière déclarative, ne peut plus être décomposée que de manière procédurale.

1.2.3. Hiérarchie utilise [11]

Chacune des (sous-)tâches d'une analyse peut être réalisée selon une ou plusieurs (sous-)stratégies. Les auteurs de cette approche suggèrent de classer ces (sous-)stratégies en fonction de leur niveau opérationnel. Ce procédé conduit également à une organisation hiérarchique.

Ainsi par exemple, un ordre partiel des (sous-)stratégies utilisées par le statisticien lors d'une analyse de régression pourrait être :

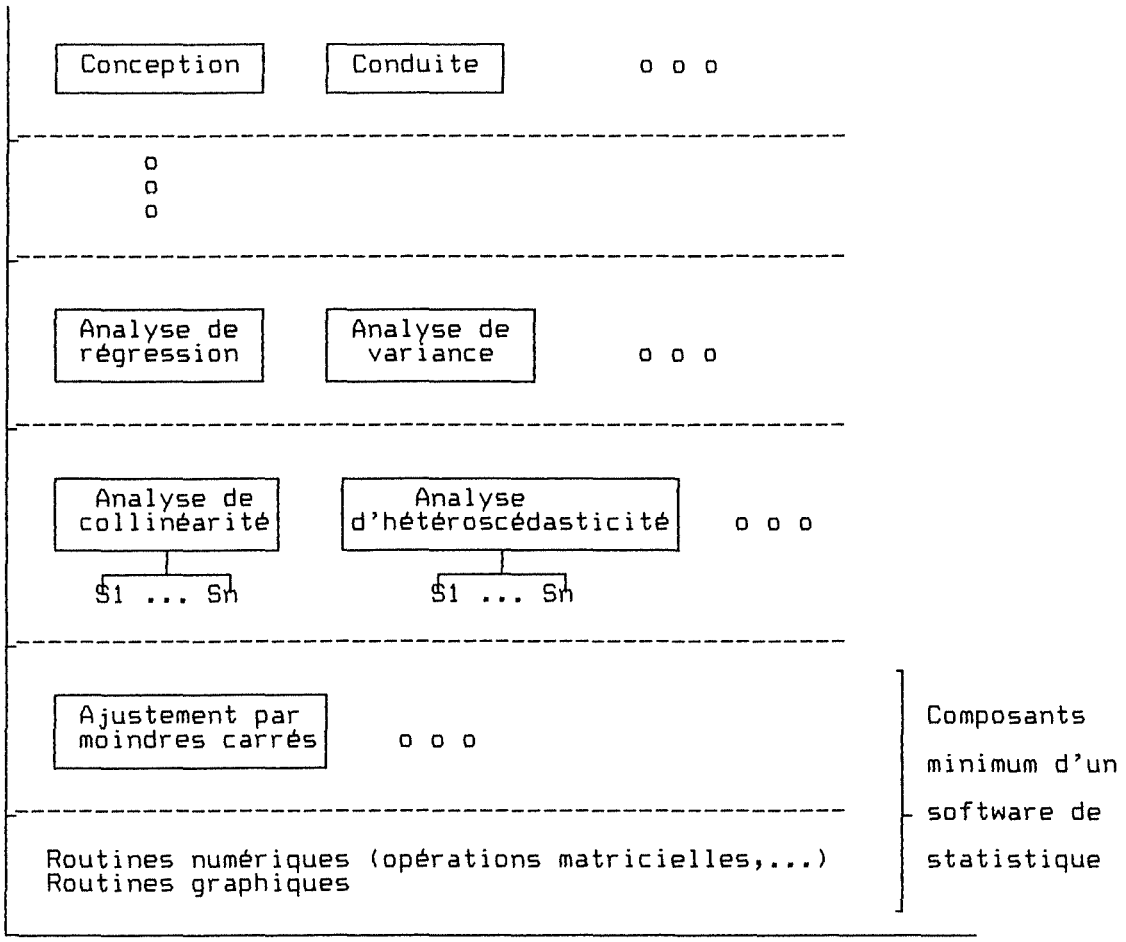


Figure II.2.: Explicitation de la stratégie : Hiérarchie utilise

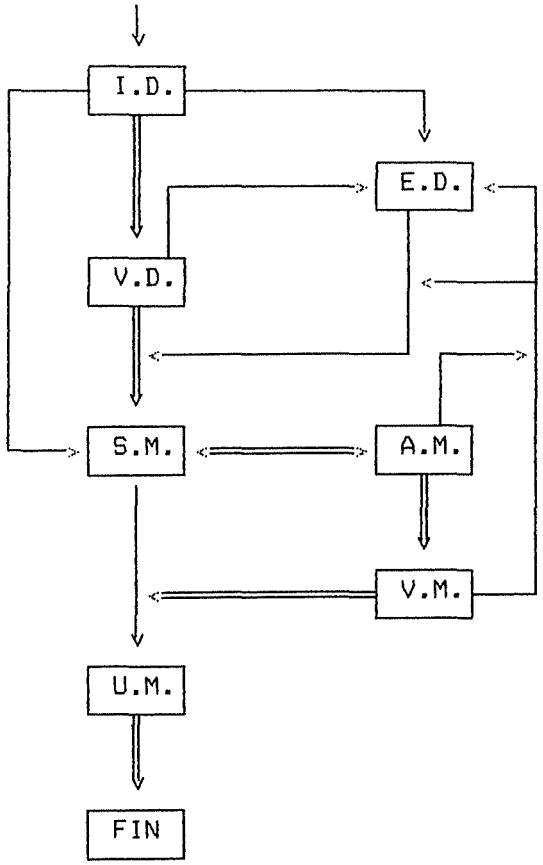
Les niveaux opérationnels sont représentés sur l'axe vertical.

Dans cette approche, chaque (sous-)stratégie d'analyse suppose l'existence de procédures aux niveaux inférieurs qu'elle peut utiliser pour atteindre son objectif.
Il est possible d'identifier plusieurs stratégies alternatives pour une même analyse, ce que nous avons indiqué par S1 ... Sn sur le schéma.
Les auteurs soulignent que les stratégies de bas niveau sont moins dépendantes du contexte du problème. Ceci signifie que la difficulté de représenter une stratégie croît avec son niveau opérationnel.

1.2.4. Diagramme de transition

Un processus analytique peut également être défini sous la forme d'un graphe orienté, dans lequel chaque noeud correspond à une (sous-)tâche de l'analyse, et chaque arc définit une transition possible d'une (sous-)tâche à une autre.

Ainsi par exemple, la tâche "construction d'un modèle" peut être décrite par le graphe suivant [12] :



Avec :

I.D. : Introduction des données
E.D. : Exploration des données
V.D. : Validation des données
S.M. : Sélection d'un modèle
A.M. : Affichage du modèle
V.M. : Validation du modèle
U.M. : Utilisation du modèle

Figure II.3. : Explicitation de la stratégie : Diagramme de transition

Une stratégie particulière est alors définie par un chemin dans le graphe. Par exemple le chemin ⟹

1.3. IMPLEMENTATION DE LA STRATEGIE

Il y a plusieurs techniques pour représenter explicitement les connaissances stratégiques dans un software, telles que règles de production, réseaux sémantiques ou frames.

Au cours d'un "workshop", la question "Quelles sont les techniques de représentation appropriées pour les connaissances statistiques"[13] a reçu des réponses aussi peu précises que :

- "Due to lack of experience, it seems to be too early to decide ..."[14],
- "This is not yet clear ..."[15],
- "I really would like to know the answer to this question ..."[16]

Pourtant, comme le fait remarqué THISTED [17], il semble clair qu'une forme de système de production couplé avec des frames serait un bon point de départ pour un S.E. en analyse de données. C'est d'ailleurs dans cette direction que vont les systèmes dévelopés par OLDFORD et PETERS [18] et par GALE et PREGIBON [19] (la description de REX en II.3.2. illustre un tel système).

Plusieurs raisons peuvent expliquer cette tendance vers les frames :

- d'abord, parce que les recherches progressent de manière prometteuse, et que ces objets structurés constituent un formalisme de représentation riche,
- ensuite, parce que les statisticiens semblent réellement organiser leurs analyses selon une structure hiérarchique, ce qui se prête bien à une description utilisant des règles de production et des frames,
- enfin, parce qu'un réseau de frames peut être utilisé pour générer des explications sémantiques (voir III.1.).

1.4. VALIDATION DE LA STRATEGIE

La seule manière de valider une stratégie est de la confronter avec de nouveaux problèmes d'analyse. Ils dévoileront certainement des limites qui nécessiteront de repenser la stratégie.

Ainsi la démarche proposée est essentiellement un cycle

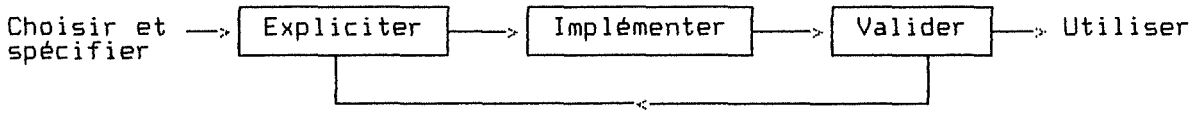


Figure II.4. : Démarche de développement d'un S.E.S.

Ce qui est une démarche fréquente pour développer un projet informatique de type exploratoire.

2. DEBAT POUR OU CONTRE

Nous proposons de donner ici un aperçu des divergences de vue entre les auteurs. Nous allons pour cela simuler un débat entre d'une part, les "sceptiques" et d'autre part, les "convaincus". Le tableau suivant se lit dans l'ordre des numéros 1a, 1b, 2a, 2b, ...

ARGUMENTS DES SCEPTIQUES	REPONSES DES CONVAINCUS
1a. Les SES constituent une menace pour le travail des statisticiens. "I do not welcome this feature as I believe [S.E.S.] will stifle individual innovative." [20]	1b. Le S.E.S. est une aide pour le statisticien. "The computer will in no way replace the function of the experienced consultant whose initiative ... often leads to important discoveries." [21] "S.E.S., if well designed, might filter some of the more mundane and less important problems and then direct the user back to the live statistician for the more challenging ones." [22]
2a. Il est difficile, voire impossible, d'expliciter une stratégie statistique. "The problem of knowledge acquisition is particularly difficult in the area os statistics [because] expertise can come from a number of areas." [23] "I have considerable doubts of letting the computer make any decisions [...] Very few statistical problems are so clear cut, that exactly one single course of actions is possible." [24]	2b. On peut commencer par un S.E.S. très spécialisé et l'étendre progressivement. "Developers of a S.E.S. should therefore initially concentrate on tapping the knowledge of a true expert in the theory and application of a specialized area." [25] "S.E.S. can be built sequentially, assuming that a general system structure has been established." [26]
3a. Les S.E.S. présentent des dangers : " - Inadequate statistical analysis, errors of the third kind (the right answer to the wrong problem). - Challenging problems may no longer reach the expert. - New and better methods may become hard to push. - Human sensitivity will be missing in the consulting process." [27]	3b. Ils présentent aussi des chances " - Improving the quality of statistical analysis. - Offering the possibility for statisticians to tackle more advanced problems by delegating routine work to S.E.S. - Standardizing some applications. - Letting non-statisticians learn about statistics."[28]

Tableau II.1. : Illustration du débat pour ou contre.

3. REALISATIONS INTERESSANTES

La plupart des réalisations actuelles sont construites comme des interfaces facilitant l'utilisation de systèmes statistiques existants. Nous présentons deux S.E.S. représentatifs de cette classe : GLIMPSE et REX.

- GLIMPSE nous permettra d'exposer les principes et intérêts de ces interfaces,
- REX nous donnera l'occasion d'étudier une architecture logicielle d'un S.E.S.

Ces deux systèmes poursuivent un objectif d'assistance (voir I.3.2.), nous décrivons dans le troisième chapitre un autre S.E.S. en cours de développement : STUDENT dont l'objectif est l'acquisition de connaissances.

3.1. GLIMPSE [29]

3.1.1. Présentation générale

GLIM est un système statistique conçu pour l'ajustement et la vérification de modèles linéaires généralisés.
GLIMPSE est un "Knowledge-Based Front-End" (K.B.F.E.) pour GLIM :

- F.E. car il offre une aide syntaxique sur l'utilisation de GLIM,
- K.B. car il propose également une aide sémantique.

GLIMPSE est écrit en Prolog avec APES [30] et fonctionne sous UNIX.

3.1.2. Structure conceptuelle d'un K.B.F.E.

La structure conceptuelle d'un K.B.F.E. comprend trois composants :

Un statisticien abstrait

Ce module incorpore des connaissances statistiques et des connaissances sur l'utilisation d'un système de statistique. Il est capable de donner des conseils en termes de tâches statistiques de haut niveau.

Un traducteur

Ce module connaît la syntaxe du langage de commandes d'un système statistique. Il est capable de traduire des tâches de haut niveau en commandes exécutables par ce système. Les tâches peuvent provenir soit du statisticien abstrait, soit directement de l'utilisateur.

Un système de statistique

Il prend en charge les fonctions de base (les deux niveaux inférieurs de la hiérarchie utilise présentées en 1.1.2.) nécessaires pour l'analyse.

Cette structure peut également se schématiser par une hiérarchie utilise :

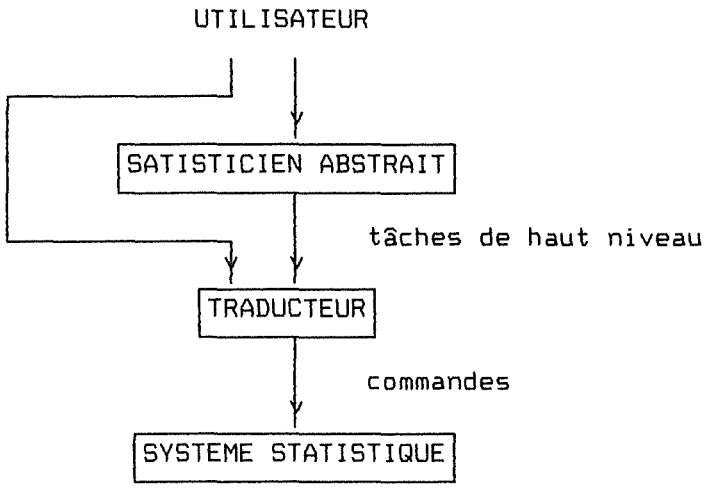


Figure II.5.: Structure conceptuelle d'un K.B.F.E.

3.1.3. spécifications de GLIMPSE

GLIMPSE doit offrir les fonctionnalités suivantes :

- donner des conseils sur les actions à entreprendre au cours d'une analyse (variable à inclure dans un modèle, ...) et sur les conclusions qui peuvent être tirées,
- expliquer les conseils donnés et les termes utilisés.

De plus, les contraintes suivantes doivent être respectées :

- les conseils donnés sont "non-autoritaires", l'utilisateur est libre de les accepter ou de les rejeter,
- il doit y avoir plusieurs niveaux d'utilisation pour permettre l'adaptation du système à divers types d'utilisateurs.

3.1.4. Explicitation de la stratégie

Sans détailler la stratégie incluse dans GLIMPSE signalons que la structure générale du processus d'analyse a été défini par un graphe orienté, qu'un chemin dans ce graphe détermine une stratégie d'analyse particulière, que les noeuds du graphe (Validation des données, Sélection de modèles, ...) correspondent à des "activités", c'est-à-dire des tâches à accomplir, et que pour chacune de ces activités, une stratégie plus fine a été identifiée.

3.1.5. Implémentation

Le programme principal présente à l'utilisateur un menu d'actions possibles : Changer d'activité , Demander des conseils , Introduire une commande

Si l'utilisateur demande des conseils (c'est une question au sens Prolog), le statisticien abstrait parcourt ses règles et deux cas peuvent se présenter :

1) Il est capable de donner les conseils

   Dans ce cas, l'utilisateur peut
   - accepter les conseils, ce qui provoque l'exécution des actions suggérées,
   - les refuser et introduire directement une commande,
   - demander une explication concernant les conseils.

2) Ses connaissances sont insuffisantes pour répondre à la question

   Dans ce cas, GLIMPSE suppose que les informations complémentaires sont présentes ailleurs et interroge soit la base de données du système statistique, soit l'utilisateur.

3.1.6. Enseignements à tirer

Le caractère interface de ces K.B.F.E. fait que les systèmes statistiques existants sont entièrement réutilisables.

La structure conceptuelle proposée est basée sur les principes de modularité et d'interface abstraite [31]. De ce fait, le statisticien abstrait pourrait être réutilisé avec un autre système statistique, il suffirait de modifier le traducteur.

Les connaissances sont réparties entre le statisticien abstrait, l'utilisateur et la base de donées du système statistique.

3.2. REX [32]

3.2.1. Présentation générale

REX est une interface pour le système S, il conseille un utilisateur naïf dans l'analyse de problème de régression.
Son architecture est inspirée de la structure Centaur [33], il est écrit en LISP et fonctionne sous UNIX.

3.2.2. Spécifications

REX doit être capable de

- guider l'analyse (en vérifiant des hypothèses de régression),
- suggérer des transformations (lorsque des hypothèses sont violées),
- justifier ses suggestions,
- interpréter des résultats intermédiaires et finals,
- expliquer des concepts statistiques.

De plus, l'ergonomie du système doit être soignée (utilisation de fenêtres, graphiques, ...)

3.2.3. Explicitation de la stratégie

La stratégie de régression développée consiste en un planification hiérarchique d'hypothèses à vérifier (voir 1.1.2.). Pour chaque violation d'hypothèse, une transformation des données ou du modèle est considérée. Ces transformations constituent donc les feuilles de l'arbre.

3.2.4. Implémentation de la stratégie

Un moteur d'inférence a été construit pour interpréter la stratégie. Nous allons en donner l'architecture logicielle et les principes de fonctionnement.

1) Architecture logicielle

Schématiquement, la structure modulaire de REX est la suivante :

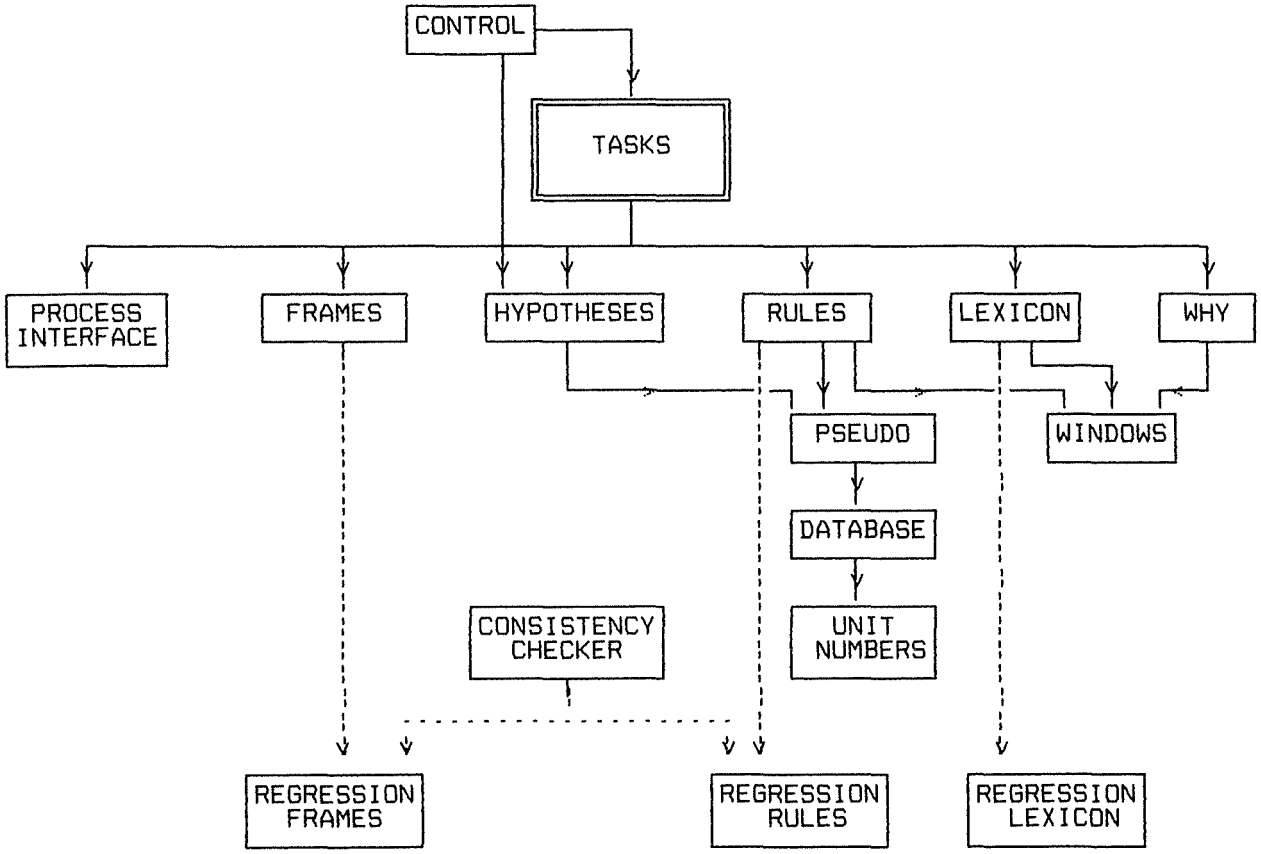


Figure II.6.: Architecture logicielle de REX

Module structure de données offrant des primitives d'accès à cette structure.

Module gérant les services fournis par les autres modules.

a) Concepts préalables

Avant de décrire chaque module, il est important de comprendre les concepts de base de l'architecture:

Une tâche est une combinaison d'opérations des modules inférieurs. Tout ce que le statisticien veut accomplir doit être codé comme une tâche. Il y aura par exemple des tâches pour :

- vérifier une hypothèse,
- présenter un message à l'utilisateur,
- préparer un appel au système S.

Au bas de la figure se trouvent des modules spécifiques à l'application régression. Ils ne font pas partie du moteur d'inférence, ce sont des structures de données représentant la stratégie sur lesquelles le moteur opère.

Un frame est une structure de données à quatre niveaux : il est composé de "slots", eux-mêmes composés de "facets" contenant des valeurs.
Des types de frame ont été identifiés pour collecter des informations sur

- les hypothèses à vérifier,
- les transformations à appliquer en cas de violation d'hypothèses,
- les tests à faire pour vérifier une hypothèse,
- les graphes à présenter à l'utilisateur.

Les slots peuvent contenir

- des noms de frames pour former la hiérarchie exprimant la stratégie,
- des noms de tâches,
- des noms de règles,
- des commandes à envoyer au système S,
- des phrases en langue naturelle à présenter à l'utilisateur.

Une règle est une instruction de la forme if <antécédants> then <conséquents>.
Trois types de règles ont été distingués :

- des règles de vérification d'hypothèses,
- des règles de sélection de nouvelles hypothèses à considérer,
- des règles de sélection d'un paramètre d'une transformation.

b) Description des modules

Control : Module gérant un agenda, qui est une pile de tâches à accomplir, et une file d'hypothèses à vérifier.

Tasks : Module cachant les modules inférieurs et présentant un ensemble d'opérations de haut niveau.

Frames : Module gérant le stockage et la consultation d'informations dans un frame.

Rules : Modules utilisant les règles pour fournir les deux mécanismes de déduction utiles au sytème : le chaînage arrière pour établir une hypothèse, le chaînage avant pour sélectionner d'autres hypothèses ou des actions spécifiques.

Lexicon : Module gérant un dictionnaire on-line permettant de présenter à l'utilisateur la définition de termes statistiques.

Why : Module gérant une liste d'événements prédéfinis et jugés intéressants. Ces événements sont enregistrés lorsqu'ils interviennent et permettent de reconstituer un texte d'explication.

Process Interface : Module gérant les appels au système S et à UNIX

Windows : Module utilitaire pour interagir avec l'utilisateur. Il gère l'entrée et la sortie d'informations.

Memory : Les fonctions de mémorisation sont fournies par trois modules : Pseudo produit des requêtes, Database stocke des relations n-aires sur des symboles qui sont vraies ou fausses, Unit Numbers stocke des valeurs numériques.

Consistency Checker : Module vérifiant la cohérence entre des ensembles de frames et de règles.

2) Fonctionnement du moteur

Le cycle de base du moteur est le suivant :

Initialisation : Placer l'hypothèse "une régression peut être faite" dans la file.

Cycle : Si l'agenda n'est pas vide
Retirer et exécuter la tâche du sommet
Sinon, si la file d'hypothèses n'est pas vide
Retirer la première hypothèse de la file,
Placer une tâche au sommet de l'agenda pour la vérifier.
Sinon terminer.

3.2.5. Enseignements à tirer

REX est un exemple d'implémentation d'une stratégie utilisant plusieurs techniques de représentation des connaissances : une hiérarchie de frames, des règles et des tâches. La simplicité relative des constructions laisse penser qu'un moteur semblable pourrait supporter des stratégies pour d'autres domaines d'analyse de données. Si cela est vrai, alors REX est une démonstration par construction de la faisabilité des systèmes de consultation en analyse de données.

Parmi les difficultés rencontrées par les auteurs, nous retiendrons :

- la difficulté de développer une stratégie robuste (acquisition de connaissances),
- la difficulté de concevoir et de réaliser un module d'explication satisfaisant.

Des recherches dans ces deux domaines seraient souhaitables. Des idées seront proposées au chapitre III.

REFERENCES CHAPITRE II.

[1] JOCKEL K.H., LENZ H.J., OSTERMAN R., et d'autres
[2] HAUX R., STREITBERG B., WITTKOWSKI K.M., et d'autres
[3] HAND D.J., Patterns in Statistical Strategy, p 356
[4] OLDFORD R.W., PETERS S.C., Implementation and Study of Statistical Strategy, p 337
[5] TUKEY J., Sunset Salvo
[6] BROOKING A.S., The Analysis Phase in Development of Knowledge Based Systems
[7] Concept de planification hiérarchique en intelligence artificielle.
Concept de hiérarchisation selon la relation "utilise" en génie logiciel.
[8] KLOSGEN W., EXLORA : an example of knowledge based data analysis
PREGIBON D., A DIY Guide to Statistical Strategy
[9] Inspiré de PREGIBON D., A DIY Guide to Statistical Strategy
[10] STREITBERG B., A modestly intelligent system for identification, estimation and forecasting of univariate time series : A4 ARIMA, artificial intelligence, and APL2, p 185
[11] OLDFORD R.W., PETERS S.C., Implementation and Study of Statistical Strategy
[12] WOLSTENHOLME D.E., NELDER J.A., A front end for GLIM, p 159
[13] HAUX R., Expert Systems in Statistics, p 179
[14] LENZ H.J., Knowledge engineering in statistical quality control, p 182
[15] NELDER J.A., A Statistical Expert System : Some Experiences in constructing a Knowledge-Based Front-End for GLIM, p 183
[16] STREITBERG B., A modestly intelligent system for identification, estimation and forecasting of univariate time series : A4 ARIMA, artificial intelligence, and APL2, p 185
[17] THISTED R.A., Representing Statistical Knowledge for Expert Data Analysis Systems, p 272
[18] OLDFORD R.W., PETERS S.C., Implementation and Study of Statistical Strategy
[19] GALE W.A., REX Review
[20] Auteur inconnu cité par HAUX R.
[21] JONES B. The Computer as a Statistical Consultant, pp 180, 193
[22] HAHN G., More Intelligent Statistical Software and Statistical Expert Systems : Future Directions, p 6
[23] BELL E., WATTS P., THESEUS : An Expert Statistical Consultant
[24] SCHACH S., Computer support for the design and analysis of survey samples, p 108
[25] HAHN G., More Intelligent Statistical Software and Statistical Expert Systems : Future Directions, p 6
[26] HAHN G., More Intelligent Statistical Software and Statistical Expert Systems : Future Directions, p 5
[27] JOCKEL K.H., Statistical expert systems and the statistical consultant - considerations about the planning stage of clinical studies, p 179
[28] JOCKEL K.H., Statistical expert systems and the statistical consultant - considerations about the planning stage of clinical studies, p 180
[29] NELDER J.A., A Statistical Expert System : Some Experiences in constructing a Knowledge-Based Front-End for GLIM
WOLSTENHOLME D.E., NELDER J.A., A front end for GLIM
[30] HAMMOND P., SERBOT M.J., APES : Augmented Prolog for Expert Systems
[31] PARNAS D.L., Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems
[32] GALE W.A., REX Review,
PREGIBON D., A DIY Guide to Statistical Strategy
[33] AIKINS J.S. Prototypical Knowledge for Expert System

III. SUJETS DE RECHERCHES

Le développement de S.E.S. ne se fait pas sans problèmes. Parmi ceux-ci, nous avons relevé la difficulté d'acquérir une stratégie robuste, la difficulé de représenter les connaissances, et la difficulté de réaliser un module d'explication.

Ces sujets font l'objet de recherches que nous allons présenter dans ce chapitre. Le premier sujet permet d'améliorer la qualité des explications en représentant explicitement les calculs statistiques au moyen d'objets structurés. Le second permet d'automatiser l'acquisition et l'implémentation d'une stratégie.

1. AMELIORER LA QUALITE DES EXPLICATIONS [1]

Des S.E.S. tels que GLIMPSE ou REX, construits comme des interfaces intelligentes pour des systèmes de statistique n'ont qu'une connaissance limitée en statistique. Bien sûr, ils implémentent une stratégie, mais ils ne contiennent pas de représentation explicite de concepts statistiques.

Ainsi, dans REX, tout calcul numérique est encodé comme une chaîne de caractères respectant la syntaxe du langage S. Ces chaînes de caractères sont des "boîtes noires" pour le S.E.S. Il ne peut pas regarder à l'intérieur pour voir comment elles fonctionnent.

Un S.E.S. plus intelligent devrait être capable de raisonner à propos de ses propres procédures de calcul. Nous allons décrire une manière de représenter explicitement des calculs statistiques, et nous verrons comment utiliser cette représentation pour générer des explications.

1.1. REPRESENTATION DE CALCULS STATISTIQUES

La représentation proposée est capable d'encoder deux types d'information. L'une est purement algébrique, nécessaire pour exécuter une procédure, l'autre est documentaire, utile pour fournir des explications. Nous nous intéressons ici à l'information de type algébrique.

L'information algébrique est représentée selon un réseau de structures de type frames. Trois types de graphes permettent de représenter des expressions, des variables et des opérations.

Prenons un exemple tiré de REX. REX utilise une macro opération de S appelée ADDVAR. Cette macro teste si une régression peut être améliorée en ajoutant une variable indépendante, donnée en argument de la macro. Ainsi, pour savoir si le terme x.LOG(x) doit être ajouté à la regression, REX évalue l'expression ADDVAR(x.LOG(x)). Actuellement dans REX, cela est représenté comme une chaîne de caractères. Avec la nouvelle représentation, schématisée ci-dessous, on aurait un frame nommé ADDxLOGx, connecté à deux graphes, l'un représentant la macro ADDVAR et l'autre l' argument :

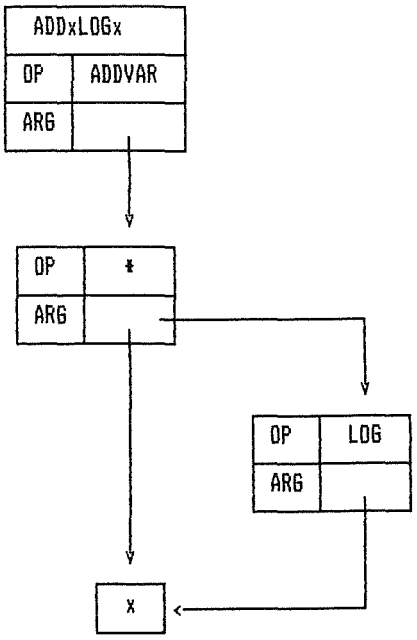


Figure III.1. : Expression ADDxLOGx

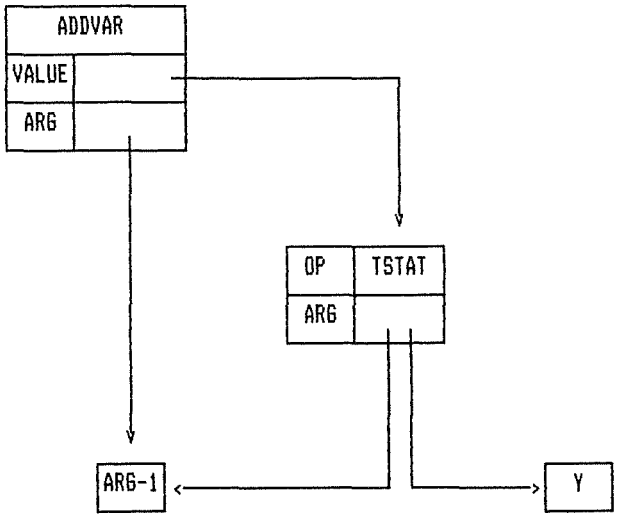


Figure III.1. : Macro opération ADDVAR

Pour simplifier, nous avons supposé que la macro ADDVAR était définie par un test nommé TSTAT qui s'applique à deux arguments.

Cette représentation est intéressante car :

- elle peut être facilement compilée en une expression d'un langage exécutable tel que S,
- on peut "attacher" aux noeuds de l'information de type documentaire, utile pour générer des explications.

1.2. GENERATION D'EXPLICATIONS

Un utilisateur pourrait souhaiter poser les questions suivantes à propos du test ADDxLOGx :

- Que calcule ce test ?
- Comment le calcule-t-il ?
- Pourquoi est-il nécessaire ?

Dans le but de répondre à ces questions, on pourrait rajouter à chaque noeud du graphe trois slots nommés WHAT, HOW, WHY et qui contiendrait des phrases en langue naturelle. Ainsi par exemple, le slot WHAT du noeud ADDxLODx pourrait contenir :

"Une mesure de l'amélioration qu'une transformation de la variable explicative courante apporterait à la régression."

Une méthode plus sophistiquée consisterait à générer, par défaut, des phrases en utilisant l'information algébrique contenue dans le graphe.
Supposons par exemple que le slot WHAT du frame ADDxLOGx soit laissé vide, et que les frames ADDVAR, * , LOG , x contiennent les slots WHAT suivants :

ADDVAR : Une mesure de la contribution de (ARG-1) à la régression,
*** : Le produit de (ARG-1) et (ARG-2),
LOG : Le logarithme naturel de (ARG-1),
x : La variable explicative courante.

En utilisant le graphe, on pourrait générer une phrase pour expliquer ce que calcule le test ADDxLOGx. Cette phrase serait :

"Une mesure de la contribution du produit de la variable explicative courante et du logarithme naturel de la variable explicative courante à la régression."

On remarquera que la première réponse dit bien ce que fait le test, tandis que la seconde dit plutôt comment il est calculé. C'est normal puisqu'on utilise l'information algébrique. Dès lors, si pour des raisons de performance, cette information algébrique était trop éloignée de la définition conceptuelle du terme à expliquer, on pourrait décider d'avoir deux graphes : l'un efficace utilisé pour effectuer les calculs, et l'autre moins efficace sur le plan algébrique, mais plus proche de la définition et utilisé pour générer des explications.

2. AUTOMATISER L'ACQUISITION ET L'IMPLEMENTATION DE LA STRATEGIE

Suite à l'expérience acquise en développant REX, les concepteurs ont étudié un autre projet, nommé STUDENT [2], qui peut être considéré comme une extension de REX. Nous nous contentons de donner les objectifs et grands principes de ce projet.

2.1. OBJECTIFS DE STUDENT

La démarche de développement d'un S.E.S. suppose une collaboration entre un statisticien, pour expliciter la stratégie, et un ingénieur de la connaissance pour la représenter et développer un moteur d'inférence.
L'objectif premier de STUDENT est de permettre à un statisticien de construire lui-même, un système de consultation à base de connaissances, pour une technique d'analyse de données, et sans se préoccuper des problèmes d'implémentation. Cet objectif revient à concevoir un système capable d'acquérir lui-même sa stratégie.

2.2. PRINCIPES DE STUDENT

D'après les auteurs, en se limitant à l'analyse de données, on peut fournir un modèle général de stratégie : on sait que l'on traite des ensembles de données, que l'analyse consiste à vérifier des hypothèses au moyen de tests, à effectuer des transformations, etc ...

Dans un premier temps, un expert en I.A. construit un système de base capable d'acquérir des stratégies calquées sur le modèle général. L'acquisition doit se faire à partir d'exemples d'une nouvelle méthode d'analyse de données. Le système de base doit être capable :

- d'acquérir le premier exemple,
- d'acquérir un exemple additionnel cohérent avec les précédents,
- d'acquérir un exemple incohérent avec un précédent.

Ensuite, deux statisticiens vont utiliser ce système de base sur des exemples pour acquérir une stratégie particulière. Le premier fixera les grandes lignes de la stratégie, le second la spécialisera à un domaine d'application (agriculture, ...)

Il en résultera un système de consultation que pourront utiliser des utilisateurs naïfs.

Le schéma suivant compare l'utilisation d'un système de statistique seul, avec une interface intelligente telle que REX et avec une interface telle que STUDENT. Il donne également les intervenants dans la réalisation d'une interface.

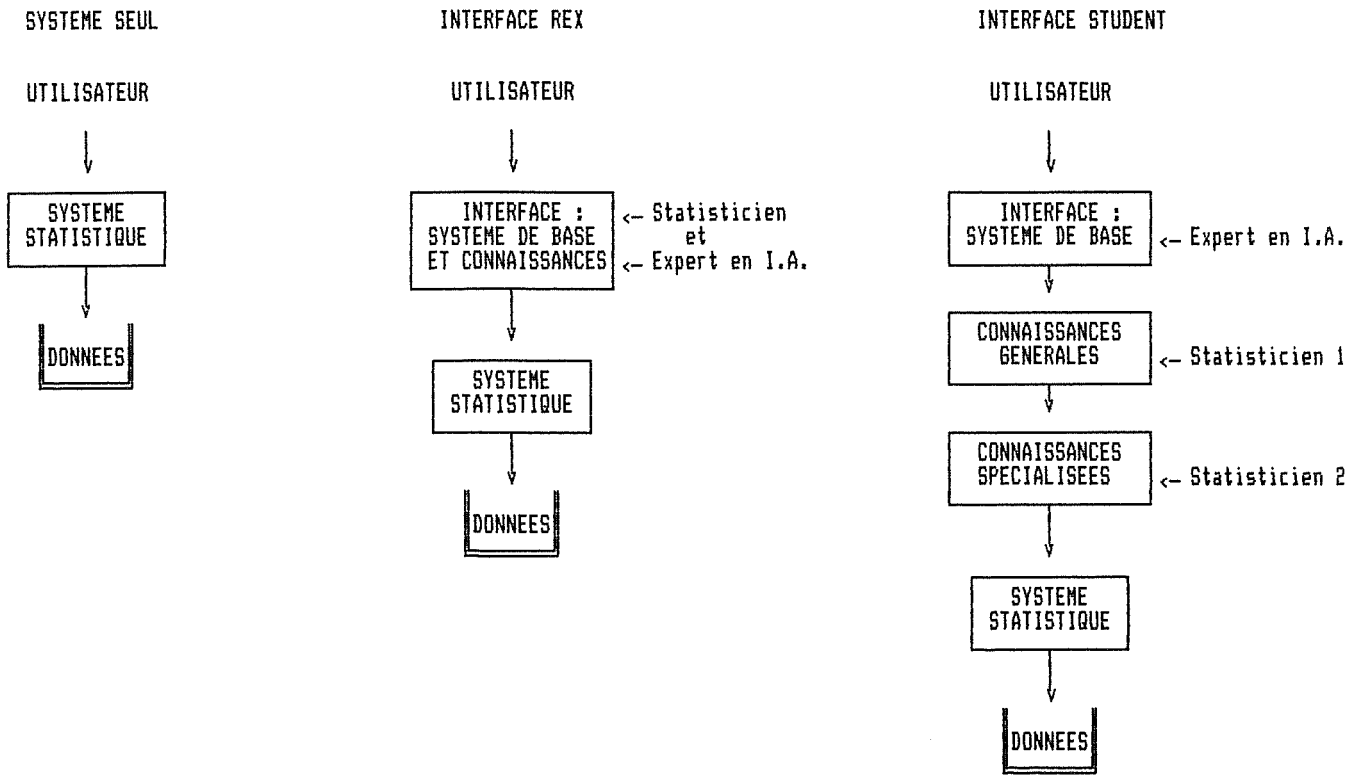


Figure III.2. : Utilisation d'un système de statistique avec interface

REFERENCES CHAPITRE III

[1] ELLMAN T., Representing Statistical Computations : Toward a Deeper Understanding
[2] GALE W.A., Student Phase 1 - A Report on Work in Progress

DEUXIEME PARTIE

DEVELOPPEMENT D'UN S.E.B.J.

INTRODUCTION

IV. LA METHODE BOX-JENKINS

V. CHOIX ET SPECIFICATIONS

VI. EXPLICITATION DE LA STRATEGIE

VII. IMPLEMENTATION DE LA STRATEGIE

VIII. VALIDATION DE LA STRATEGIE

IX. CRITIQUE DU SYSTEME

INTRODUCTION

Au cours d'un séminaire, G.LIBERT concluait son exposé sur les S.E. par la phrase :"Des systèmes experts, il faut cesser d'en parler, il faut en faire." Au terme de notre recherche documentaire, nous pensons que cette affirmation est particulièrement vraie pour les S.E.S. C'est pourquoi, nous avons décidé de poursuivre notre étude par la réalisation d'un S.E.S.

Cette réalisation devait nous fournir des arguments complémentaires concernant l'opportunité et la faisabilité des S.E.S. Elle devait surtout nous apporter une expérience dans le développement d'un logiciel d'un type particulier.

Nous avons choisi de réaliser un S.E.S. dont le but serait d'aider un utilisateur à analyser une série chronologique selon la méthode B&J. Ce domaine statistique a été retenu pour des raisons subjectives et objectives. Nous ne pouvons en effet nier notre intérêt pour la modélisation statistique. Nous pensons toutefois qu'un besoin existe dans ce domaine car les connaissances requises limitent la classe d'utilisateurs potentiels de cette méthode performante. De plus, nous pensons qu'un tel S.E.S est faisable car ces connaissances sont relativement bien formalisées et la démarche d'analyse ne varie guère d'un expert à l'autre.

Le lecteur trouvera dans cette partie six chapitres :

- Le premier (IV. La méthode BOX - JENKINS) expose très sommairement les fondements théoriques de la méthode B&J et ses principes d'utilisation.
- Les chapitres suivants sont consacrés au développement du S.E.S en respectant la démarche en quatre étapes proposée dans la première partie : Choix et spécifications, Explicitation, Implémentation et Validation de la stratégie statistique.
- Le dernier chapitre (IX. Critique du système) nous permettra de tirer les enseignements de cette expérience de réalisation.

IV. LA METHODE BOX - JENKINS

L'analyse d'une série chronologique Z_t (t=1..N) selon la méthode B&J [1] consiste à construire un modèle (processus stochastique) adapté à cette série puis, à utiliser ce modèle à des fins de prévision.

Ce chapitre décrit brièvement la classe de modèles considérés, la démarche à suivre pour construire un modèle et l'utilisation du modèle pour obtenir des prévisions. Pour plus de détails, le lecteur consultera l'annexe A.1.

1. CLASSE DE MODELES

La classe de modèles considérés permet de décrire des séries stationnaires ou non. Ils sont connus sous le nom de modèles "AutoRegressive Integrated Moving Average" (ARIMA) généralisés d'ordre (p,d,q). Ils s'écrivent :

$$\emptyset_p(B)W_t = \theta_0 + \theta_q(B)a_t$$

avec W_t : processus rendu gaussien et stationnaire par une double transformaton de Z_t:
- une transformation non linéaire de paramètre δ notée $Z^{(\delta)}$
- une transformation en différences de paramètre d notée $\nabla^d Z^{(\delta)}$

$\emptyset_p(B)$: opérateur autorégressif d'ordre p,

θ_0 : terme constant,

$\theta_q(B)$: opérateur moyenne mobile d'ordre q,

a_t : bruit blanc de distribution $N(0,\sigma_a)$.

Pour décrire une série saisonnière, de périodicité s, la méthode considère des modèles multiplicatifs ARIMA d'ordre $(p,d,q)x(P,D,Q)_s$ dans lesquels deux modèles interviennent :
- l'un pour décrire les variations de saison à saison, c'est-à-dire entre Z_t, Z_{t+s}, ...
- l'autre pour décrire les variations entre les valeurs successives Z_t, Z_{t+1}, ...

2. CONSTRUCTION D'UN MODELE

Pour construire un modèle, BOX et JENKINS recommandent une approche itérative comprenant trois étapes :

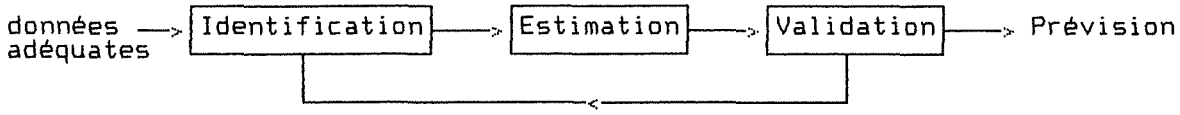


Figure IV.1. : Démarche pour la construction d'un modèle

2.1. IDENTIFICATION DES PARAMETRES

L'objectif de cette étape est de trouver une sous-classe des modèles ARIMA généralisés qui peut être utilisée pour représenter la série Z_t, tout en respectant un principe de parcimonie.

On commence par rechercher les paramètres δ, s, d, D permettant de rendre la série gaussienne et stationnaire.
On recherche ensuite l'ordre des modèles, c'est-à-dire les paramètres p,q et P,Q.

Cette étape demande une bonne part de jugement car le choix des paramètres se fait essentiellement en confrontant des représentations graphiques (Range-Mean Plot RMP, Fonction d'AutoCorrélation ACF, Fonction d'AutoCorrélation Partielle PACF, ...) avec des comportements théoriques.

2.2. ESTIMATION DES COEFFICIENTS

Une fois les paramètres du modèle identifiés, on calcule des estimations pour les p+q+P+Q+1 coefficients du modèles (les $\emptyset_i$, θ_j) et pour σ_a.
Cette étape, totalement automatisable, peut être ignorée par l'utilisateur.

2.3. VALIDATION DU MODELE

Un modèle sera déclaré valide s'il n'est pas surspécifié et si les résidus respectent les propriétés de bruit blanc.

Bien qu'il existe des tests pour décider de l'adéquation d'un modèle, cette étape nécessite du jugement car en cas de non-adéquation, les résidus sont utilisés pour identifier un nouveau modèle qui peut être à son tour estimé et validé.

3. GENERATION DE PREVISIONS

Lorsque l'on dispose d'un modèle ARIMA, dont on a testé l'adéquation, on peut l'utiliser pour prévoir des valeurs futures de la série Z_t. Des formules donnent :

- $\hat{Z}_t(h)$: une estimation de Z_{t+h} faite à l'instant t,
- un intervalle de confiance associé à chaque prévision.

REFERENCES CHAPITRE IV.

[1] BOX G.E.P., JENKINS G.M., Time Series Analysis, Forecasting and Conrol

V. CHOIX ET SPECIFICATIONS

La démarche de développemnt adoptée étant celle proposée en II.1. nous commençons par donner les caractéristiques de notre S.E. D'une manière très synthétique, l'objectif du système est d'assister un utilisateur naïf lors de l'application de la méthode B&J.

La description qui suit comprend un structure conceptuelle, les choix et hypothèses faits et les spécifications fonctionnelles.

1. STRUCTURE CONCEPTUELLE

Le schéma suivant donne une première idée de la structure conceptuelle de S.E. et de son environnement. Notons que cette structure est caractérisée par

- deux "composants" externes : un système statistique et un utilisateur,
- trois "composants" internes : un noyau et deux interfaces.

La justification de la structure et la description de chacun des composants seront données au cours des chapitres suivants.

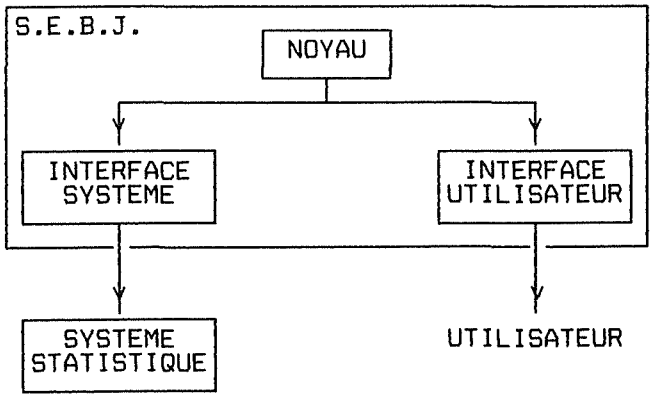


Figure V.1. : Structure conceptuelle du S.E.B.J.

2. CHOIX ET HYPOTHESES

Les choix et hypothèses faits concernant l'environnement du S.E. et le S.E. lui-même sont fortement guidés par les réalisations actuelles en matière de S.E.S.

2.1. CARACTERISTIQUES DU S.E.B.J.

L'objectif du système est d'assister un utilisateur lors de la phase "conduite de l'analyse". Ceci signifie que l'utilisateur a déjà :

- formulé le problème (modéliser une série chronologique et utiliser le modèle à des fins de prévision),
- sélectionné une méthode pour le résoudre (la méthode B&J).

Nous pensons que l'intégration de ces deux étapes préalables nécessiterait des connaissances et de l'expérience en matière de consultation statistique que nous n'avons pas.
De plus, nous supposons que les données sont adéquates.

2.2. CARACTERISTIQUES DE L'ENVIRONNEMENT DU S.E.B.J.

L'environnement du S.E. est constitué d'un utilisateur et d'un système statistique.

2.2.1. L'utilisateur

Comme recommandé (voir II.1.1.) nous décidons d'écarter les extrêmes, c'est-à-dire que notre système n'est pas destiné :

- aux utilisateurs ignorant tout de la statistique et de la méthode B&J car ceux-ci recherchent certainement, soit un système leur permettant d'apprendre la méthode, soit un système automatique,
- aux utilisateurs experts qui n'ont sans doute pas besoin d'être guidés pour appliquer une méthode qu'ils maîtrisent.

Notre système est plutôt destiné à des utilisateurs, certes débutants en ce qui concerne la méthode B&J, mais ayant des connaissances de base en statistique.

2.2.2. Le système statistique

Nous supposons également que le S.E. est construit sur un système statistique quelconque, doté d'un langage de commandes, de sorte que le S.E. puisse utiliser les services du sytème sous-jacent pour ses besoins en calcul numérique.

Cette hypothèse nous paraît légitime car :

- les enseignements des réalisations actuelles (GLIMPSE, REX) laissent penser que les capacités de calcul des systèmes statistiques sont satisfaisantes et réutilisables,
- notre objectif n'est pas de réaliser des routines de calcul statistique, mais d'étudier les spécificités des S.E.S.

3. SPECIFICATIONS FONCTIONNELLES

Nous allons à présent décrire les trois composants internes du S.E.B.J. en termes de fonctions à réaliser. Les spécifications du NOYAU décrivent les capacités du S.E., celles de l'INTERFACE UTILISATEUR des contraintes ergonomiques et celles de l'INTERFACE SYSTEME des contraintes d'ordre architecturale.

3.1. LE NOYAU

Le noyau du S.E. s'occupe de notre objectif principal : il incorpore des connaissances statistiques pour assister un utilisateur lors de l'application de la méthode B&J. Il doit être capable de :

- suggérer une séquence de tâches restant à accomplir pour analyser une série chronologique,
- suggérer des valeurs de paramètres lorsque le choix de celles-ci nécessite du jugement de la part de l'utilisateur,
- donner des explications sur le "pourquoi" et le "comment" d'une suggestion et sur des concepts statistiques propres à la méthode B&J.

De plus, l'assistance doit être non-autoritaire : l'utilisateur doit avoir la possibilité de ne pas suivre les suggestions.

3.2. L'INTERFACE UTILISATEUR

Dans la mesure où les systèmes statistiques actuels souffrent de défauts ergonomiques, il nous paraît important de soigner la présentation du S.E. Le rôle de cette interface est précisément de gérer le dialogue entre l'utilisateur et le noyau du S.E. Etant donné les caractéristiques de l'utilisateur et les recommandations ergonomiques [1], nous souhaitons un dialogue :

- contrôlé par l'utilisateur au moyen d'objets interactifs, notamment des "menus de commandes" et des "boîtes de dialogue" décrits dans l'annexe A.3.
- offrant un système d'assistance syntaxique on-line et une bonne gestion des erreurs.

De plus, nous imposons à cette interface une contrainte supplémentaire facilement compréhensible : elle doit être réutilisable dans un autre contexte.

3.3. L'INTERFACE SYSTEME

Cette interface gère les échanges d'informations entre le système statistique et le noyau du S.E. Elle doit être capable :

- d'envoyer une commande exécutable par le système sous-jacent,
- de fournir au noyau les résultats de l'exécution d'une commande.

Nous pouvons à présent justifier la séparation proposée dans la structure conceptuelle :

- si les exigences des utilisateurs changeaient en ce qui concerne la présentation du système ou,
- si nous souhaitions ajouter ou modifier des fonctions concernant l'application de la méthode B&J ou encore,
- si nous devions choisir un autre système statistique,

alors, grâce à cette découpe, les modifications n'affecteraient qu'un seul composant.

REFERENCES CHAPITRE V.

---

[1] SCHNEIDERMAN B., Designing the User Interface : Strategies for Effective Human Computer Interaction

VI. EXPLICITATION DE LA STRATEGIE

Compte tenu des spécifications du noyau du S.E., nous devons développer une stratégie pour construire un modèle, selon la méthode B&J et avec comme principal souci les phases d'identification et de validation.

La stratégie suivante est une première proposition. Il est clair qu'elle devra être modifiée et/ou affinée ultérieurement en fonction de son efficacité. Elle est basée sur la théorie [1] et sur la pratique des utilisateurs de la méthode B&J [2].

Le lecteur trouvera un schéma de synthèse de la stratégie sous forme de planification hiérarchique, puis les principes concernant les stratégies d'identification et de validation d'un modèle. Pour plus de détails (justifications et stratégies alternatives), il consultera l' annexe A.2.

1. SCHEMA DE LA STRATEGIE

Comme nous l'avons déjà signalé, la construction et l'utilisation d'un modèle comportent quatre étapes.

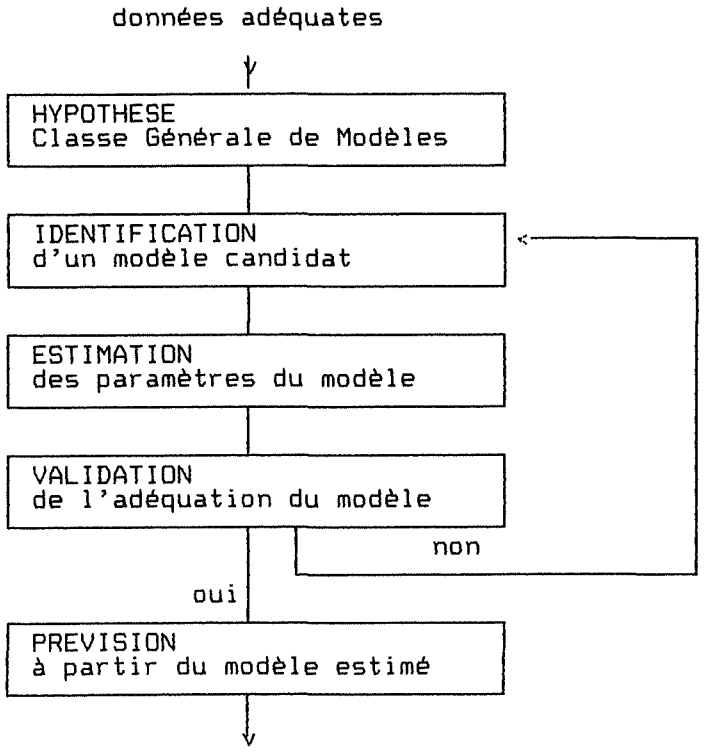


Figure VI.1. : Stratégie pour la construction d'un modèle

Nous allons à présent donner un schéma de la stratégie pour les étapes identification et validation.

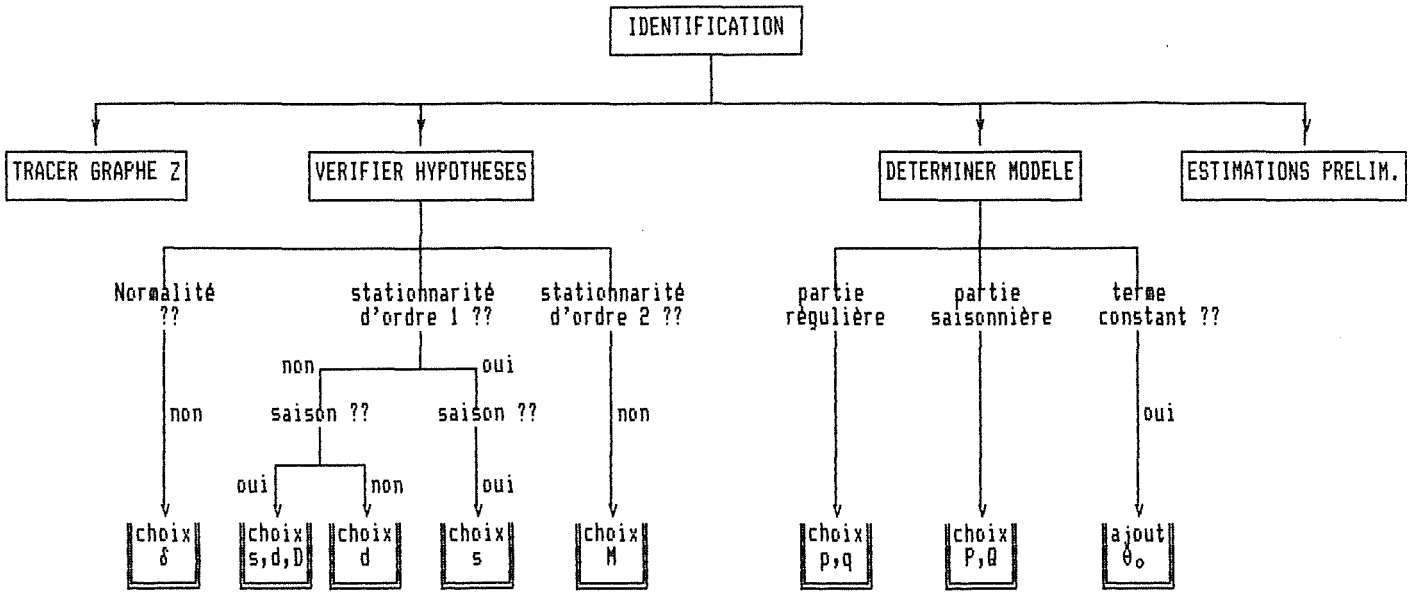


Figure VI.2. : Stratégie d'identification d'un modèle

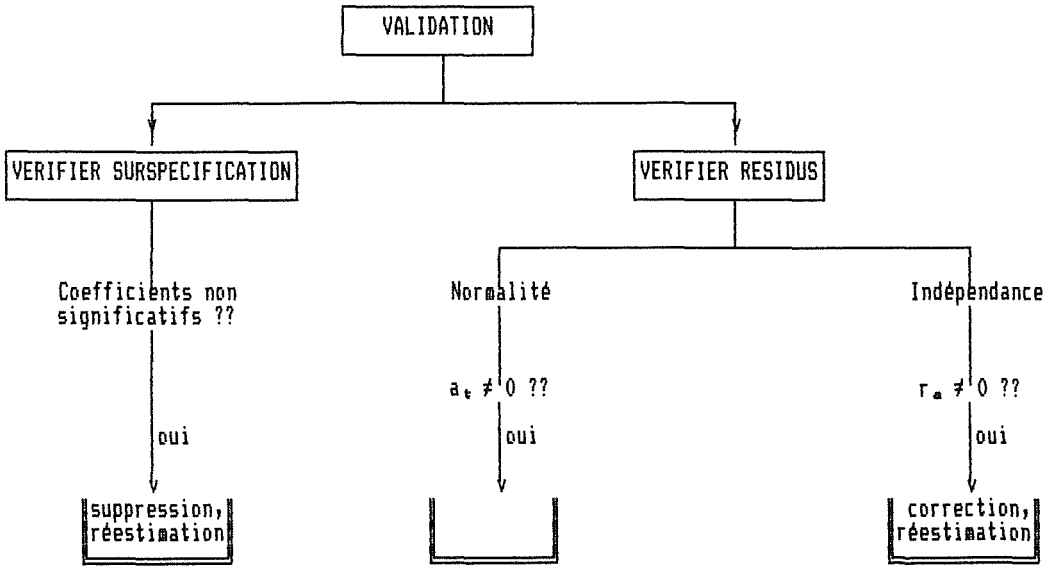


Figure VI.3. : Stratégie de validation d'un modèle

Nous convenons d'appeler une tâche, tout noeud de l'arbre de la stratégie. On distinguera trois types de tâches :

- les tâches décomposables : ayant un ou plusieurs noeuds fils,
- les tâches logiques : tâches décomposables mais dont la décomposition dépend du résultat d'un test. Elles sont marquées par ??
- les tâches terminales : tâches non décomposables et caractérisées par une procédure de traitement. Elles apparaissent au bas de la hiérarchie dans une petite boîte (⌊⌋).

2. STRATEGIE D'IDENTIFICATION

Pour identifier un modèle, nous commençons par vérifier les hypothèses de normalité et de stationnarité puis, nous déterminons l'ordre du modèle.

2.1. VERIFIER LES HYPOTHESES

2.1.1. Normalité : Transformation δ

D'après BARTLETT, pour obtenir la normalité, on regarde si la variance de la série dépend de la moyenne. Si ce n'est pas le cas, il n'y a pas besoin de transformer la série, sinon on recherche la transformation $Z^{(\delta)}$ qui stabilise la variance en fonction de la moyenne.

2.1.2. Stationnarité d'ordre 1

Ayant obtenu la normalité,nous recherchons les degrés de différenciation d et D tels que la moyenne de $\nabla^d\nabla^D Z^{(\delta)}$ soit stationnaire. La détermination de d et D est intimement liée à la détection d'une période s.
Si on ne connaît pas s, et si la série n'est pas stationnaire,on procède comme suit :

- nous déterminons un d tel que la série soit presque stationnaire,
- nous recherchons ensuite s,
- nous choisissons alors définitivement les valeurs de d et D.

Si on connaît s, on choisit directement les valeurs définitives de d et D.
Si la série est stationnaire, on recherche seulement s.

1) Différenciation provisoire : d

Pour obtenir une première valeur de d :

- on recherche la série $\nabla^d Z^{(\delta)}$ dont la variance est minimale (d'après ANDERSON),
- on vérifie, au moyen d'un test de tendance, que cette valeur de d n'est pas trop grande (d'après LIBERT).

2) Périodicité : s

Souvent la période est connue de l'utilisateur. Si ce n'est pas le cas, il procède souvent comme suit :

- il recherche les pics r_k de l'ACF les plus significatifs,

- il abandonne ceux qui sont expliqués par la partie non saisonnière du modèle,
- s'il reste des r_k, il regarde s'il existe un k tel que r_k et r_{2k} sont significatifs. Si c'est le cas, s=k sinon, éventuellement il choisit s tel que r_s soit le plus grand coefficient.

3) Différenciation définitive : d,D

D'une manière similaire à 2), si la série est périodique,
- on recherche la série $\nabla^d\nabla^D Z^{(s)}$ dont la variance est minimale,
- on vérifie que les valeurs d et D ne sont pas trop grandes.

En pratique d et D sont inférieurs ou égaux à 2

2.1.3. Stationnarité d'ordre 2

L'hypothèse de stationnarité d'ordre 2 (constance dans le temps des coefficients d'autocorrélation) est rarement vérifiée par les utilisateurs. Ce problème n'est d'ailleurs pas abordé explicitement dans la littérature.
Dans un but de prévision, LIBERT suggère de rechercher un sous-ensemble de la série transformée, comprenant les dernières valeurs, de longueur maximale M et vérifiant la stationnarité d'ordre 2. Pour cela, il utilise un test permettant de décider si deux ACF sont semblables ou non.

2.2. DETERMINER UN MODELE

La pratique montre que la plupart des séries peuvent être modélisées par des processus ARMA(p,q) avec $p+q \leq 2$. La démarche entreprise par un utilisateur, et décrite par LIBERT, est la suivante :
Il teste successivement différents modèles. Par souci de parcimonie, l'ordre des tests est bruit blanc, puis MA(1) et AR(1), puis MA(2) AR(2) et ARMA(1,1), ... Les tests se font en deux étapes :
- vérifier les conditions de stationnarité et d'inversibilité des modèles, exprimées en fonction des coefficients d'autocorrélation,
- comparer des coefficients des ACF ou PACF à des seuils de signification.

Pour déterminer l'ordre de la partie saisonnière, la méthode est identique mais appliquée aux coefficients saisonniers.

Enfin, un test de signification de la moyenne de la série transformée permet de décider s'il faut introduire un terme constant dans le modèle.

3. STRATEGIE DE VALIDATION

Pour vérifier qu'un modèle n'est pas surspécifié, on teste le caractère significatif des coefficients $\emptyset_i$, θ_j, en les comparant à deux fois leur erreur standard.
Si un coefficient n'est pas significativement différent de zéro, on l'abandonne et on réestime le nouveau modèle.

Pour vérifier l'indépendance des résidus, on teste le caractère significatif des coefficients d'autocorrélation des résidus. Si certains coefficients sont différents de zéro, on corrige le modèle et on le réestime.

REFERENCES CHAPITRE VI.

[1] BOX G.E.P., JENKINS G.M., Time Series Analysis Forecasting and Control
LIBERT G., Analyse de Séries Chronologiques : Elaboration Automatique et Continue de Prévisions
[2] Interview de BORSU A. et LIBERT G., notamment

VII. IMPLEMENTATION DE LA STRATEGIE

Dans ce chapitre, nous proposons une solution pour implémenter la stratégie décrite. Cette solution comprend une structuration du système en composants (architecture logicielle) et des explications complémentaires concernant les modules clés de l'architecture.

1. ARCHITECTURE LOGICIELLE

Rappelons tout d'abord le schéma de la structure conceptuelle que nous donnions en V.1.

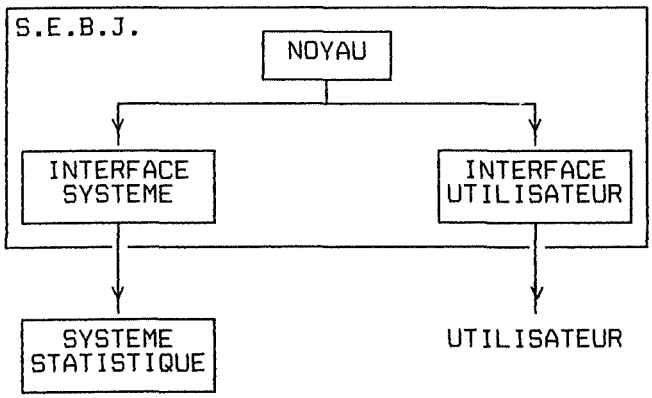


Figure VII.1. : Structure conceptuelle du S.E.B.J.

Nous allons à présent détailler la structure de chacun des composants internes. La démarche adoptée est bien connue en génie logicielle :

- Hiérarchisation : identification des niveaux d'une hiérarchie et d'une relation entre les niveaux.
- Modularisation : décomposition des niveaux en modules.

1.1. LE NOYAU

1.1.1. Hiérarchisation

Nous avons identifié trois niveaux liés par une relation "utilise". Nous les avons baptisés : Manager, Strategy et Memory respectivement dans l'ordre descendant de la hiérarchie.

Memory : Niveau comprenant des modules permettant de consulter et de sauvegarder des informations à propos du modèle en cours d'élaboration, telles que :

- valeurs de la série, paramètres et coefficients du modèle,
- tâches restant à accomplir,
- détails de l'exécution de la dernière tâche,
- explications susceptibles d'être données à l'utilisateur.

Le "secret" des modules de ce niveau est le mode de mémorisation des informations. Ils ont un caractère dynamique puisque les informations détenues sont différentes pour chaque analyse.

Strategy : Niveau comprenant des modules permettant d'exécuter des tâches décomposables, logiques ou terminales car ils connaissent la stratégie, c'est à dire :

- la hiérarchie permettant de décomposer une tâche,
- les tests à appliquer aux tâches logiques,
- les traitements correspondant aux tâches terminales.

Ce niveau utilise les services de Memory et de l'interface système. Il a un caractère statique puisque la stratégie est identique pour chaque analyse.

Manager : Niveau comprenant des gestionnaires de haut niveau qui réalisent les fonctions visibles par l'utilisateur. Ces gestionnaires utilisent les services de Strategy et Memory et ceux des interfaces utilisateur et système.

1.1.2. Modularisation

La découpe modulaire du noyau du système se présente de la façon suivante :

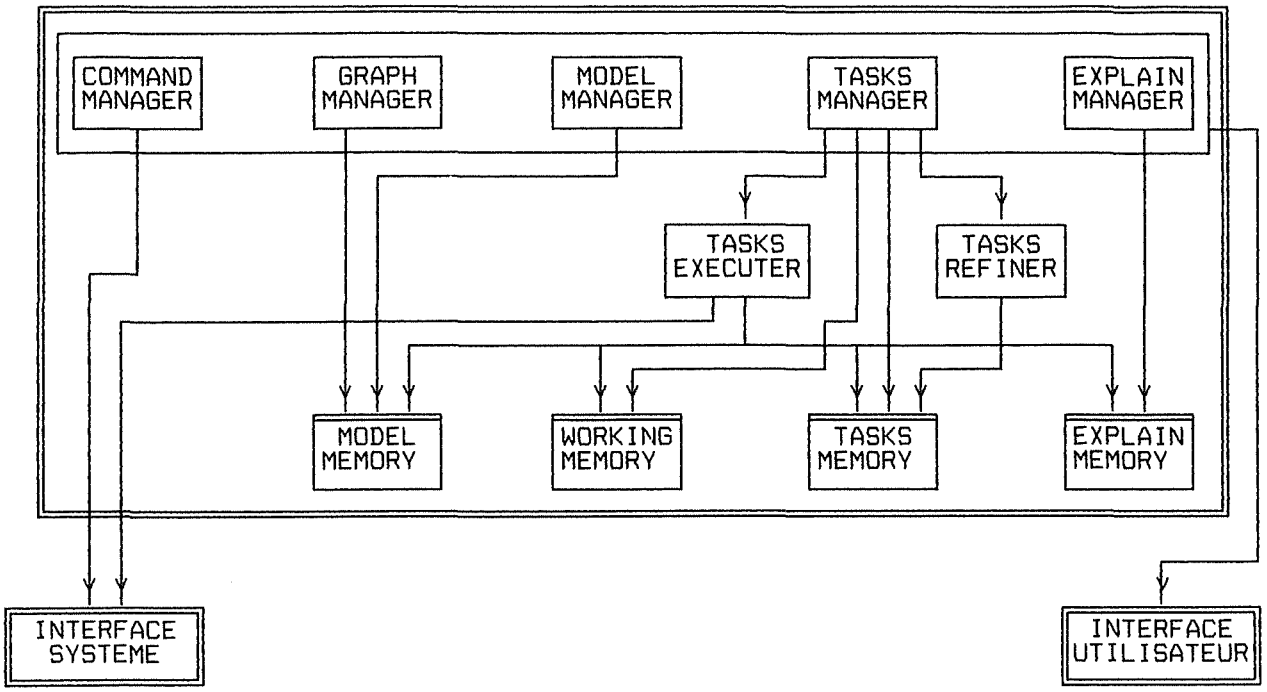


Figure VII.2. : Découpe modulaire du noyau du S.E.B.J.

1) Le niveau Memory

Les modules de ce niveau sont tous des structures de données. Les opérations disponibles sur les structures ont été classées en trois catégories :

- gestion d'une structure,
- accès aux informations d'une structure,
- sauvegarde et mise à jour d'informations dans une structure.

Nous donnons pour chaque structure un tableau de synthèse des opérations. Les paramètres soulignés sont des valeurs de retour. Les spécifications se trouvent en annexe A.3.

Model Memory

Structure de données permettant d'accéder et de sauvegarder des informations concernant un modèle en cours d'élaboration : les valeurs des différentes séries, les paramètres du modèle déjà identifiés, les coefficients déjà estimés. Les opérations disponibles sont :

Gestion	Accès	Sauvegarde
create_model(id_m) create_serie(id_m,id_s,lg)	get_serie(id_m,id_s,liste de valeurs) get_param(id_m,id_p,param) get_coeff(id_m,id_c,liste de valeurs)	put_serie(id_m,id_s,t,valeur) put_param(id_m,id_p,valeur) put_coeff(id_m,id_c,i,valeur)

avec id_m : Identifiant d'un modèle,
id_s : Identifiant d'une série : Z,Y,W,a,ACF,PACF,
id_p : Identifiant d'un paramètre : δ,s,p,d,q,P,D,Q,
id_c : Identifiant d'une série de coefficients : ar,ma,
t,i : Indices d'une série de valeurs ou de coefficients

Tableau VII.1. : Opérations du module Model Memory

Tasks Memory

Structure de données permettant d'accéder et de sauvegarder des informations concernant les tâches restant à exécuter pour élaborer un modèle. Ces tâches forment une pile et chacune est caractérisée par son nom (identification, estimation, choix_s, ...), son type (décomposable, logique, terminale), sa profondeur dans l'arbre de la stratégie (0,1,2,...). Les opérations disponibles sont :

Gestion	Accès	Sauvegarde
create_tstack(id_m,id_ts) add_task(id_ts,id_t) del_task(id_ts,id_t) get_toptask(id_ts,id_t)	get_tname(id_ts,id_t,name) get_ttype(id_ts,id_t,type) get_tlevel(id_ts,id_t,level)	add_task(id_ts,id_t)

avec id_ts : Identifiant d'une pile de tâches,
id_t : Identifiant d'une tâche.

Tableau VII.2. : Opérations du module Tasks Memory

Working Memory

Structure de données permettant d'accéder et de sauvegarder des informations concernant les détails de l'exécution de la dernière tâche de type terminal ou logique. Cette mémoire contient des informations telles que : meilleur choix (c'est-à-dire la suggestion faite), autres candidats, nom du critère de choix, évaluation du meilleur choix par rapport aux autres candidats. Les opérations disponibles sont :

Gestion	Accès	Sauvegarde
clean_WM() create_wmserie(lg,id_s) del_wmserie(id_s)	get_best(liste de valeurs) get_other(liste de valeurs) get_crit_name(name) get_eval_best(eval) get_wmserie(id_s,liste de val)	put_best(élém. de liste) put_other(élém. de liste) put_crit_name(name) put_eval_best(eval) put_wmserie(id_s,t,valeur)

Tableau VII.3. : Opérations du module Working Memory

Exlain Memory

Structure de données permettant d'accéder et de sauvegarder des informations de type explications à donner à l'utilisateur. Cette mémoire contient des textes (flux de caractères) précisant : pourquoi telle suggestion est faite, comment la dernière tâche est exécutée, ce que signifie un concept statistique. Elle permet également de mémoriser des événements, c'est-à-dire des couples (item, valeur d'item) pour accompagner un texte (voir 2.2.). Les opérations disponibles sont :

Gestion	Accès	Sauvegarde
clean_why() clean_how()	get_txtwhy(flux de car.) get_txthow(flux de car.) get_txtwhat(name,flux de car.) get_evtwhy(liste d'event) get_evthow(liste d'event)	put_evtwhy(flux de car.) put_evthow(flux de car.)

Tableau VII.4. : Opérations du module Explain Memory

2) Le niveau Strategy

On trouve à ce niveau des modules de traitement. Pour effectuer ce traitement les modules ont besoin, en plus de leurs connaissances stratégiques, d'informations préalablement mémorisées. Les spécifications se trouvent en annexe A.3.

Task Executer

Module de traitement permettant d'exécuter une tâche logique ou terminale. Exécuter signifie appliquer les tests et traitements de la stratégie. L'exécution d'une tâche a pour conséquence la mise à jour de toutes les mémoires.
L'opération disponible a la forme :

execute_task(id_m,id_ts)

Task Refiner

Module de traitement permettant d'exécuter une tâche décomposable. Exécuter signifie ici remplacer la tâche par d'autres plus simples (les noeuds fils de l'arbre). L'exécution a pour seule conséquence la mise à jour de la mémoire des tâches.
L'opération disponible a la forme :

refine_task(id_m,id_ts)

3) Le niveau Manager

Les modules de ce niveau implémentent les fonctions mises à la disposition de l'utilisateur. Ces fonctions sont :

FILE	MODEL	TASKS	EXPLAIN	GRAPH	COMMAND
Load Save :	Identif. Estimat. Validity Forecast	Show Execute Remove Details	Def (What) Why How	Series Resid Acf Pacf Options	Execute

Tableau VII.5. : Fonctions visibles par l'utilisateur

File Manager

Module permettant à l'utilisateur de manipuler des fichiers : ce sont les fonctions habituelles telles que charger, éditer, sauver des fichiers de données, ...

Model Manager

Module permettant à l'utilisateur d'obtenir ou de donner des informations concernant le modèle en cours d'élaboration.

Identif. : Affichage des paramètres du modèle (δ, $(p,d,q) \times (P,D,Q)_s$) avec possibilité de les modifier.
Estimat. : Affichage des estimations des coefficients du modèle identifié.
Validity : Affichage de valeurs caractérisant la validité du modèle estimé.
Forecast : Affichage de valeurs futures de la série.

Task Manager

Module permettant à l'utilisateur de gérer les tâches :

Show : Affichage des tâches restant à accomplir pour le modèle en cours d'élaboration.
Execute : Ordre d'exécuter (au sens vu plus haut) la tâche suivante.
Remove : Ordre de ne pas exécuter la tâche suivante et de la supprimer.
Details : Affichage des détails d'exécution de la dernière tâche.

Explain Manager

Module permettant à l'utilisateur de demander des explications.

Def. : Affichage d'un texte décrivant un mot-clé introduit ou sélectionné par l'utilisateur.
Why : Affichage d'un texte expliquant les raisons qui poussent le système à faire les dernières suggestions.
How : Affichage d'un texte expliquant la manière dont le système a procédé pour faire les dernières suggestions.

Graph Manager

Module permettant à l'utilisateur de demander l'affichage (sous forme graphique ou non) de données intéressantes : les séries Z,Y,W, les résidus, les fonctions d'autocorrélation, ...

Command Manager

Module permettant à l'utilisateur d'entrer directement une commande exécutable par le système statistique sous-jacent.

1.2. L'INTERFACE UTILISATEUR

Pour construire notre interface, nous nous sommes inspirés de quelques guides de construction d'interface homme-machine [1]. Nous avons notamment repris des idées concernant des niveaux d'abstraction pour établir une hiérarchie "utilise", concernant les applications extensibles pour développer une interface réutilisable et concernant la programmation par événement.

1.2.1. Hiérarchisation

Nous avons défini des niveaux d'abstraction, liés par une relation "utilise" et permettant de cacher la diversité de l'environnement matériel/système d'exploitation et la diversité des utilisateurs. Les niveaux retenus sont, dans l'ordre descendant de la hiérarchie et sous le niveau "application" (niveau Manager du noyau du S.E.) : "Dialog Control", "Structured I/O", "elementary I/O", "Devices Control".

Devices Control : Ce niveau assure l'indépendance matérielle. Il comprend des modules d'interface entre les niveaux supérieurs de la hiérarchie et les périphériques d'entrée-sortie.

Elementary I/O : Niveau comprenant des modules d'entrée-sortie de bas niveau offrant des services tels que édition et affichage d'une ligne de texte, ...

Structured I/O : Niveau comprenant des modules d'entrée-sortie de haut niveau. Les objets actuellement disponibles sont des menus déroulants et des boîtes de dialogue, dont la description se trouve en annexe A.3.

Dialog Control : Niveau assurant le lien entre les fonctions de l'application (le niveau Manager du noyau du S.E.) et leur présentation à l'utilisateur. Ce contrôle se fait au moyen d'objets structurés.

1.2.2. Modularisation

Nous donnons à présent la découpe modulaire de l'interface. Par souci de clarté, nous l'avons quelque peu simplifiée en omettant le niveau "Elementary I/O". Le lecteur intéressé par les spécifications externes de chaque module consultera l' annexe A.3.

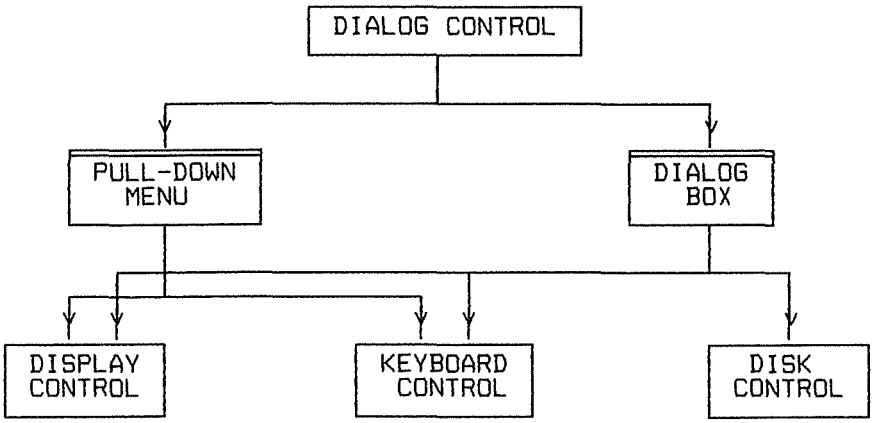


Figure VII.6. : Découpe modulaire de l'Interface Utilisateur

1.2.3. Principes de fonctionnement

Plutôt qu'une description des modules, nous préférons exposer les principes de fonctionnement d'un dialogue.

D'après les spécifications, nous souhaitions un dialogue contrôlé par l'utilisateur. Voici, dans les grandes lignes, comment il se déroule.
L'utilisateur manifeste ses intentions au moyen "d'événements de bas niveau" tels que touche au clavier, ... Lorsqu'ils interviennent, ces événements sont analysés par le "Dialog Control". Suite à l'analyse, le "Dialog Control" déclenche éventuellement un serveur spécialisé dans la manipulation d'ojets interactifs (menu, boîte, ...), lequel renvoie l'événement sous forme abstraite, c'est-à-dire enrichi d'un code. D'après ce code, le "Dialog Control" détermine le point d'entrée dans l'application à utiliser pour prendre en charge l'événement.

Lorsque l'application doit prendre l'initiative du dialogue (demander des informations à l'utilisateur), elle le fait également au moyen d'événements analysés par le "Dialog Control".

Ces interactions peuvent se schématiser de la manière suivante :

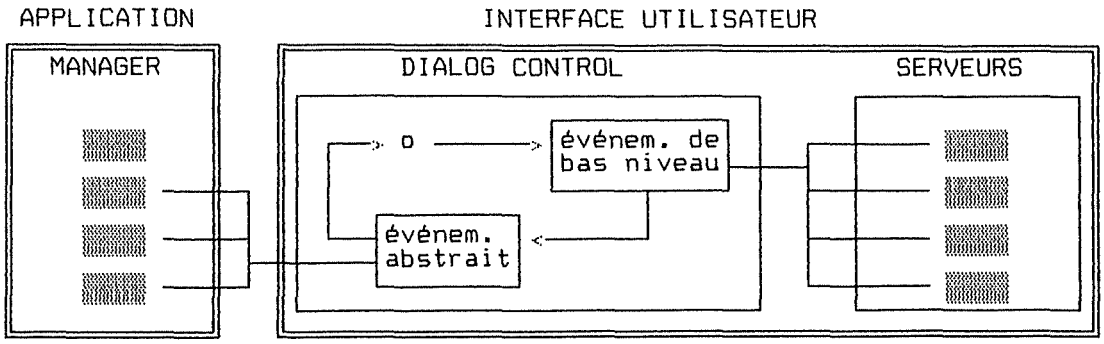


Figure VII.7. : Principe de fonctionnement de l'Interface Utilisateur

1.3. L'INTERFACE SYSTEME

L'interface système n'a pas été découpée en modules. En effet, nous avons simulé la présence d'un système statistique. Tous les besoins en calcul numérique (calcul de variance, tests statistiques,...) ont été implémentés par des procédures regroupées dans un module appelé SYSTEME.
Cette solution est évidemment provisoire.

2. REALISATION DES FONCTIONNALITES

La réalisation des deux principales fonctionnalités du système (voir V.3.1.), suggérer des valeurs de paramètres ou des tâches et donner des explications, mérite quelques commentaires. En terme de génie logiciel, nous nous intéressons ici à la conception détaillée de quelques modules clés.

2.1. SUGGERER DES VALEURS DE PARAMETRES OU DES TACHES

Pour les tâches décomposables, le problème est trivial. Par contre, pour chaque tâche logique et terminale, il nous faut des règles ou une procédure traduisant la stratégie (voir VI.). Ces procédures doivent fournir une "suggestion" (une tâche terminale ou des valeurs de paramètres) et une liste, éventuellement vide "d'autres possibilités". Ces procédures se trouvent regroupées dans le module "Tasks Executer" et sont connues par lui seul.

Nous allons donner un exemple d'une telle procédure pour une tâche logique (test d'une transformation δ) et pour une tâche terminale (choix du degré de différenciation). Pour les autres tâches, le lecteur consultera l' annexe A.2.

2.1.1. Test d'une transformation δ

Spécification :

PRE : Z[1..N] série observée.

POST : la procédure détermine si oui ou non la variance de la série est fonction de la moyenne.

Principe :

Initialisation : k := partie entière de $\sqrt{N}$

lg := partie entière de N/k

Traitement : - Découper le série Z en k sous-séries de longueur lg.
- Calculer moyenne et variance de chaque sous-série.
- Calculer le coefficient de corrélation r entre les moyennes et les variances.

Terminaison : Si $r \neq 0$ alors "suggestion" := effectuer tâche choix δ.
"autre possibilité" := ne pas l'effectuer.
Sinon "suggestion" et "autre possibilité" sont inversées.

Commentaires :

- Les valeurs de k et lg ont été choisies de manière à réaliser un compromis entre bonne estimation des moyennes et variances et bonne estimation de r.
- Ne connaissant pas la distribution de r, nous décidons arbitrairement de fixer à [-0.2 : 0.2] l'intervalle en dehors duquel r sera considéré différent de 0.
- Cette procédure a l'avantage d'être simple. Si elle devait se révéler inefficace, on la remplacerait par un test plus rigoureux (par exemple le test de BARTLETT décrit dans l'annexe A.2.).

2.1.2. Choix du degré de différenciation d

Spécification :

PRE : Y[1..N] série observée transformée ($Z^{(\sigma)}$).

POST : la procédure détermine le degré d pour que la série $\nabla^d Y$ ait une moyenne stationnaire.

Principe :

Initialisation : /

Traitement : Pour d := 0,1,2
- Calculer v(d) : variance de $\nabla^d Y$
- Retenir d tel que v(d) est minimum.

Tant que d>0
- Appliquer le test de tendance sur $\nabla^{d-1} Y$
- si tendance alors terminer sinon d := d-1

Terminaison : "suggestion" := la valeur de d retenue.
"autres possibilités" := les deux valeurs écartées.

Commentaires :

Le test de tendance retenu est celui de MANN [2] :

Pour toute valeur d'une série Y_t, soit n_t, le nombre de données Y_s qui le précède (s<t) telles que $y_s<Y_t$. La quantité $T = \Sigma n_t$ est, sous l'hypothèse (Ho) d'absence de tendance, distribuée asymptotiquement comme une loi normale N(m,σ) avec

$m = n(n-1)/4$

$\sigma^2 = n(n-1)(2n+5)/72$

Dès lors, soit $u(T) = (T-m)/\sigma$, si $u(T) > Q(N(0,1),\alpha)$ on peut rejeter Ho.

Nous choisirons $\alpha = 0.05$ et donc $Q(N(0,1),\alpha) = 1.96$

2.2. DONNER DES EXPLICATIONS

L'utilisateur peut demander trois types d'explications (voir V.3.1.) que nous avons appelés "What", "Why" et "How".

2.2.1. What : Définir des concepts

Puisque les utilisateurs ne sont pas des experts, ils souhaiteront certainement recevoir des explications sur des concepts statistiques utilisés par le système. La fonction "What" demande à l'utilisateur un mot, si ce mot fait partie d'un dictionnaire prédéfini, alors le système affiche un texte définissant le terme.

La réalisation d'une telle fonctionnalité n'est pas très compliquée. Nous pensons qu'un fichier séquentiel, contenant les définitions, indexé par un ensemble de mots clés pourrait convenir. Pour retrouver une définition, il suffirait de comparer le mot introduit par l'utilisateur à l'ensemble des mots clés.
Schématiquement, le principe est le suivant :

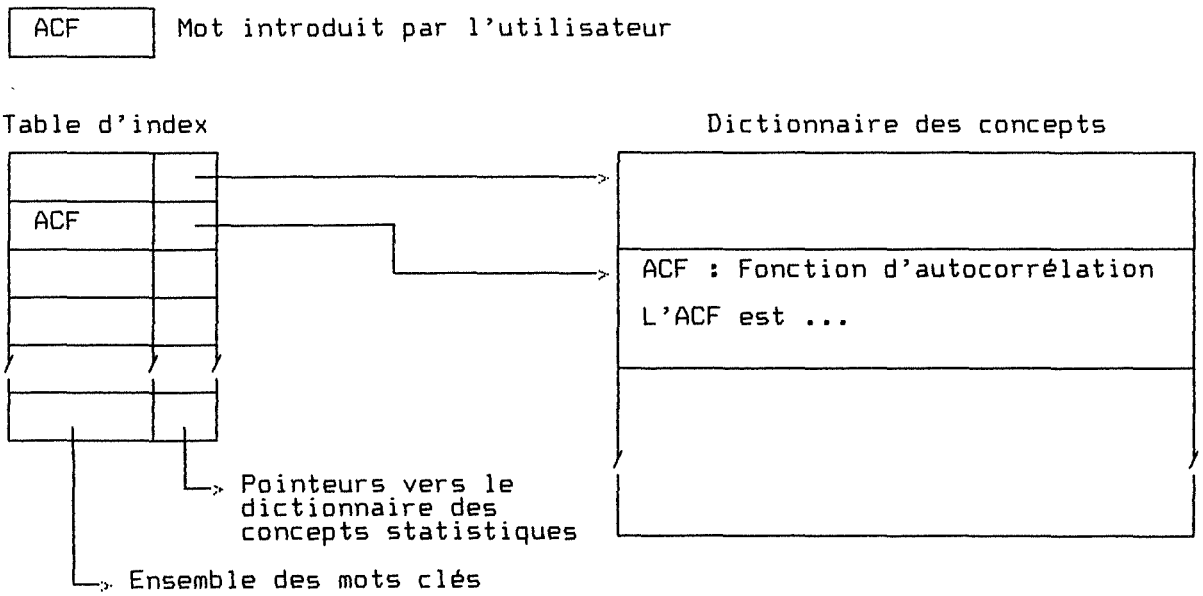


Figure VII.8. : Principe des explications de type What.

2.2.2. Why How : Expliquer une suggestion

Notre système est capable de faire des suggestions. Pour rendre celles-ci convaincantes, il nous paraît essentiel qu'il puisse les justifier. Une justification pourrait être composée d'un texte et d'une liste d'événements :

- Le texte expliquerait la manière de procéder (How), c'est-à-dire une traduction en langue naturelle des règles et procédures implémentant la stratégie. (voir VII.2.1. et A.2.).
- La liste serait un ensemble d'événements jugés intéressants (Why), intervenus lors de l'exécution des règles ou procédures.

La fonction How peut être implémentée d'une façon similaire à la fonction What. A chaque noeud logique et terminal de l'arbre de la stratégie correspondrait un texte. On y accéderait en recherchant dans un index des identifiants des noeuds, celui qui correspond au noeud courant.
Schématiquement cela donne :

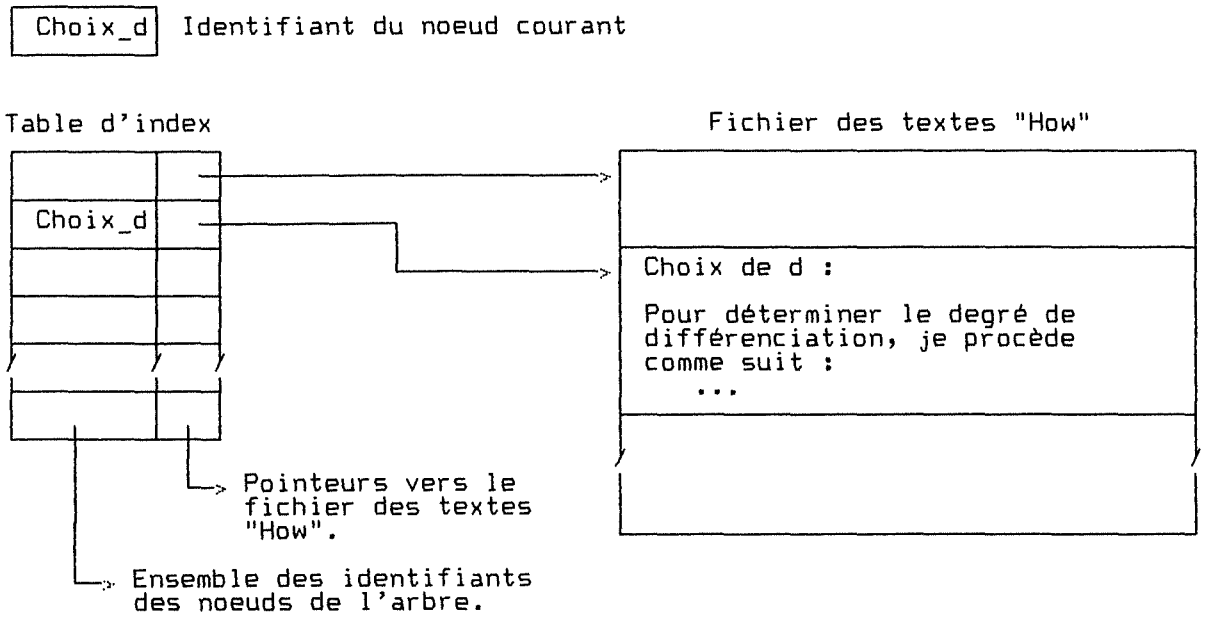


Figure VII.9. : Principe des explications de type How.

La fonction Why se différencie des deux précédentes car ce type d'explication ne peut être défini une fois pour toute, elle dépend de l'exécution de la procédure.
Si un événement est un couple (item, valeur d'item), la procédure d'exécution pourrait s'occuper de mettre à jour la liste d'événements et la fonction Why n'aurait plus qu'à affichar cette liste.

3. AUTRES CHOIX D'IMPLEMENTATION

Pour des raisons de disponibilité du matériel, nous avons développé le S.E.B.J. sur micro-ordinateur et sous MS-DOS.
Le langage de programmation utilisé est le C. Les raisons de ce choix sont données en IX.2.

REFERENCES CHAPITRE IX.

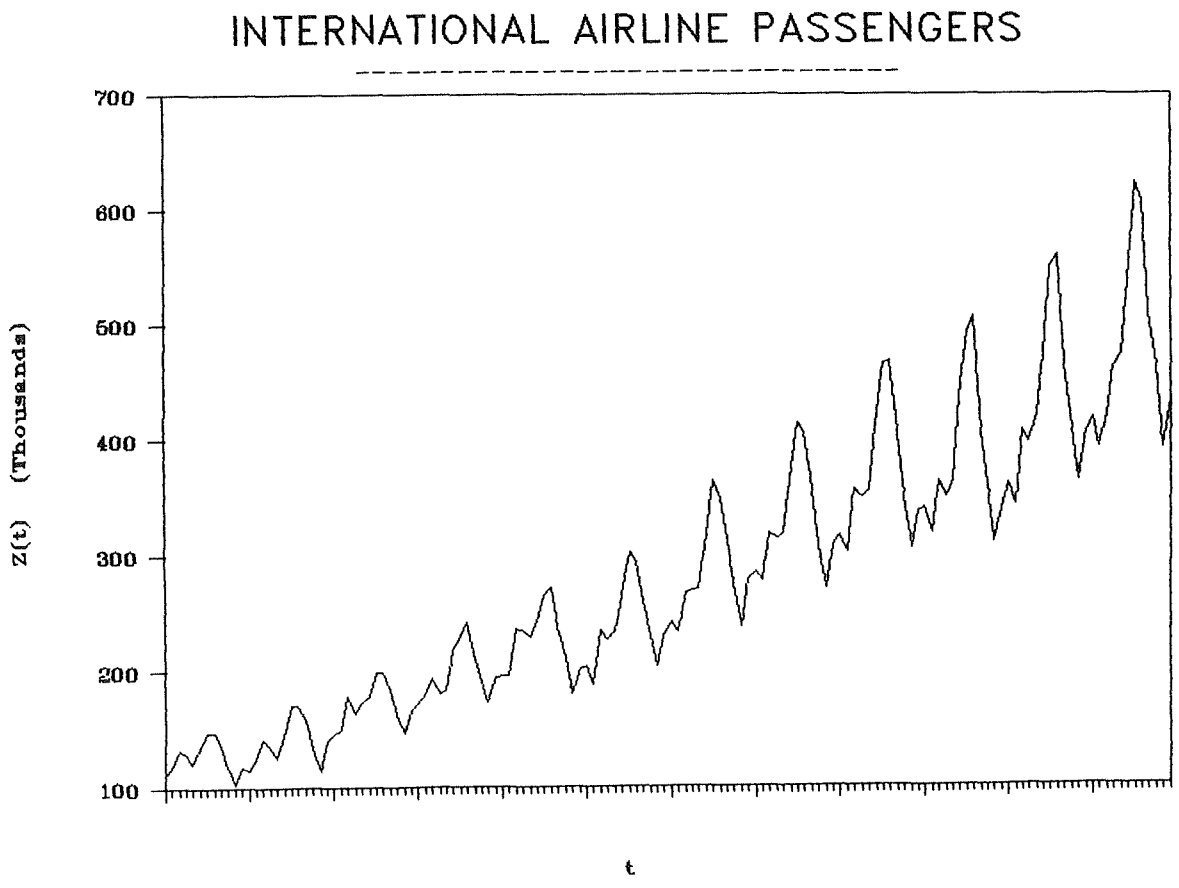
[1] SCHNEIDERMAN B., Designing the User Interface : Strategies for Effective Human Computer Interaction
[2] Test décrit dans LIBERT G., Analyse de Séries Chronologiques : Elaboration Automatique et Continue de Prévisions

VIII. VALIDATION DE LA STRATEGIE

Nous avons soumis au système plusieurs séries. Dans ce chapitre, nous présentons en détail les suggestions qu'il fait concernant l'identification d'un modèle pour l'une d'entre elles. Le modèle suggéré sera comparé avec le modèle proposé par un expert humain. Pour les autres séries testées, nous donnons un tableau de synthèse comparant les paramètres identifiés par les experts humains et par le S.E.B.J.

1. LA SERIE

Nous avons choisi d'analyser en détail la série "International Airline Passengers" tirée de BOX, JENKINS [1]. Il s'agit de 144 observations reprenant le nombre total mensuel de passengers pour les vols internationaux de janvier 1949 à décembre 1960. La présence évidente de plusieurs phénomènes (tendance, saisonnalité,...) laisse penser qu'il s'agit d'un bon premier test pour la stratégie. Graphiquement, la série se présente de la façon suivante.



2. IDENTIFICATION D'UN MODELE

2.1. LES SUGGESTIONS DU S.E.B.J.

Les suggestions faites par le S.E.B.J., pour l'identification d'un modèle, résultent de l'exécution des procédures décrites dans l'annexe 2.

2.2.1. TRANSFORMATION : δ

Test de normalité : la corrélation entre moyennes et variances de 12 sous-séries vaut 0.97. Le test échoue et le système suggère de transformer la série.

Choix de δ : la corrélation minimum entre moyennes et variances est atteinte pour δ=-0.25, valeur que le système suggère.

2.2.2. DIFFERENCIATIONS ET SAISONNALITE : s,d,D

Test de tendance : le test de MANN détecte une tendance et le système suggère de chercher la valeur des paramètres s,d et D.

Choix de d : le critère de variance minimum donne d=1.

Choix de s : la procédure heuristique appliquée à la série ∇Y_t donne les ensembles suivants :

A={4,8,12,16,20,24,32}

B={12}

le système suggère donc s=12.

Choix de d,D : le critère de variance minimum donne d=D=1. Les tests de tendance appliqués à $\nabla_{12}Y_t$ et ∇Y_t sont significatifs. Le sytème suggère donc d=D=1.

2.2.3. STATIONNARITE D'ORDRE 2 : M

Choix de M : par la procédure implémentée, le système propose de retenir les 93 dernières valeurs de la série. Au delà, une différence significative apparaît entre les ACF.

2.2.4. ORDRE DU MODELE : p,q,P,Q

Choix de p,q : le premier et unique modèle vérifiant les hypothèses de stationnarité et d'inversibilté et retenu comme CANDIDAT est un ARMA(1,1).

Choix de P,Q : deux modèles sont CANDIDATS (ARMA(0,1) et ARMA(1,0)). Le modèle ARMA(0,1) est CHOISI.

2.2. COMPARAISON AVEC L'IDENTIFICATION DES EXPERTS

Les experts humains, en l'occurrence BOX et JENKINS, proposent le modèle $(0,1,1)x(0,1,1)_{12}$ pour la série $(\ln Z_t)$ [1].
Il y a donc deux différences par rapport aux suggestions du S.E.B.J.: l'une concerne le paramètre δ et l'autre le paramètre p.

2.2.1. δ=-0.25 ou δ=0 ?

Il n'y a pas de justification économique pour choisir une valeur plutôt qu'une autre. Si l'on retient δ=0, il subsiste une dépendance entre moyennes et variances de $\ln Z_t$.
La différence provient de l'application de deux méthodes : le maximum de vraisemblance d'un côté et la stabilisation de la variance de l'autre. Nous ne sommes pas en mesure de décider laquelle est la plus efficace.

2.2.2. p=1 ou p=0 ?

La différence n'est pas définitive. On peut supposer que lors de la phase de validation le S.E.B.J. constaterait que son modèle est sur-spécifié ($\emptyset_1$ non significatif) et qu'il proposerait un nouveau modèle $(0,1,1)x(0,1,1)_{12}$ comparable à celui préconisé par les experts humains.

3. SYNTHESE DES RESULTATS

La stratégie du S.E.B.J. ayant passé son premier test avec succès, nous avons soumis au système d'autres séries, également tirées de BOX, JENKINS [1]. Le tableau suivant résume les résultats obtenus et permet une comparaison avec les modèles identifiés par les experts humains.

SERIE	IDENTIFICATION B&J			IDENTIFICATION S.E.B.J.		
	δ	pour Z_t $(p,d,q)x(P,D,Q)_s$	pour $Z_t^{(\delta)}$ $(p,d,q)x(P,D,Q)_s$	δ	pour Z_t $(p,d,q)x(P,D,Q)_s$	pour $Z_t^{(\delta)}$ $(p,d,q)x(P,D,Q)_s$
B	1	(0,1,1)		-1	(0,1,1)	(2,1,0)
E	1	(2,0,0) ou (3,0,0)			(2,0,0)	
F	1	(2,0,0)		1	(2,0,0)	
G	0		$(0,1,1)x(0,1,1)_{12}$	-0.25		$(1,1,1)x(0,1,1)_{12}$

Avec B : IBM Common Stock Closing Prices : Daily ; 255 observations
E : WOLFER Sunspot Numbers : Yearly ; 100 observations
F : Yields from Batch Chemical Process ; 70 observations
G : International Airline Passengers : Monthly ; 144 observations

REFERENCES CHAPITRE IX.

[1] BOX G.E.P., JENKINS G.M., Time Series Analysis, Forecasting and Control, p 531, p 305.
JENKINS G.M., Forecasting Seminar: Univariate Stochastic Models

IX. CRITIQUE DU SYSTEME

Cette critique du système est organisée comme un débat dans lequel une personne pose des questions et nous y répondons pour défendre et justifier des choix, ou pour reconnaître les faiblesses du système.

Nous aborderons successivement trois questions :"Le système est-il vraiment expert ?", "Le choix du langage C est-il judicieux ?", "Le système a-t-il des lacunes ?".

1. SYSTEME EXPERT OU NON ?

1.1. QUESTION

Le système présenté est-il vraiment un système expert ?
En effet, l'architecture rappelle plus un logiciel "classique", on n'y retrouve pas des modules tels que Base de Connaissances, Moteur d'Inférence.

1.2. REPONSE

Il nous semble que la définition d'un S.E. repose sur ses caractéristiques fonctionnelles, c'est-à-dire ses capacités d'imiter le comportement humain et d'expliquer son raisonnement. La manière de réaliser ces fonctions (base de connaissances ou non, représentation déclarative ou procédurale, ...) ne fait pas, à notre avis, partie de cette définition.

Il est vrai que la démarche de conception s'appuie sur des principes de génie logiciel qui conduisent à une architecture assez classique. Cependant, on peut retrouver dans notre système des similitudes avec des techniques d'I.A. Ces similitudes sont : une recherche dans un espace d'états, une approche réduction de problèmes et une hiérarchie de frames.

1) Une recherche dans un espace d'états

Rappelons que, selon cette approche, un problème à résoudre est un quadruplet <S,I,F,OP>

avec S : un ensemble d'états constituant un espace de recherche,

I : un ensemble d'états initiaux tels que I c S

F : un ensemble d'états finals, vérifiant une condition et tel que F c S

OP: un ensemble d'opérateurs permettant de passer d'un état de S à un autre.

Dans notre cas, on peut considérer qu'un état est un modèle $\emptyset_p(B)W_t = \theta_0 + \theta_q(B)a_t$ pour un série Z_t donnée et caractérisé par $<\delta,s,p,d,q,P,D,Q>$. Dès lors on aurait :

S : l'ensemble de tous les états otenus en combinant les différentes valeurs possibles des paramètres (l'espace est énorme).

I : la série Z_t c'est-à-dire l'état <1,0,0,0,0,0,0,0>.

F : un ensemble d'états tels que a_t soit un bruit blanc

OP: un ensemble d'opérateurs tels que "transformer(δ)", "différencier(d)", ... que l'on devrait spécifier plus précisément.

Dans notre système, il y a bien cette notion d'état présente dans le module Model Memory, et la stratégie peut être vue comme des heuristiques permettant d'orienter la recherche et d'éviter ainsi une recherche exhaustive.

2) Une approche réduction de problèmes

Selon cette approche, un problème à résoudre est un quadruplet <P,Pr,Po,OP>

avec P : un ensemble de problèmes.

Pr: un ensemble de problèmes primitifs tels que Pr c P

Po: le problème à résoudre tel que Po c P

OP: un ensemble d'opérateurs de réduction permettant de décomposer un problème en problèmes plus simples.

La stratégie développée n'est rien d'autre qu'un arbre de réduction dans lequel on a :

P : l'ensemble des tâches statistiques.

Pr: l'ensemble des tâches de type terminal.

Po: le problème "Analyser une série".

OP: opérateurs qui consistent simplement à remplacer une tâche par d'autres.

Remarquons toutefois que dans la plupart des systèmes de résolution de problèmes par réduction, l'arbre est potentiellement beaucoup plus grand et les problèmes primitifs beaucoup plus simples.

3) Une hiérarchie de frames

Les trois modules Tasks Memory, Tasks Executer et Tasks Refiner sont équivalents à un module qui contiendrait toute la connaissance sur les tâches, sous la forme d'une hiérarchie de frames. Dans un tel module, un frame serait une structure de données du type suivant :

```
task_frame :
        task_id   : un identifiant de la tâche.
        task_type : le type décomposable, logique ou terminal.
        task_sub  : la liste des tâches "filles" si la tâche
                    est décomposable ou logique.
        task_exe  : une procédure d'exécution si la tâche est
                    logique ou terminale.
```

Figure VIII.1. : Structure d'un frame pour représenter une tâche

On pourrait également y incorporer un quatrième module (Explain Memory) en ajoutant au frame des slots "task_how", "task_why". Ces idées rejoignent la technique d'implémentation de REX (voir II.3.2.) et les propositions de ELLMAN en matière d'explication (voir III.1.).

2. POURQUOI LE LANGAGE C ?

2.1. QUESTION

Le langage de programmation choisi (langage C) n'est pas vraiment orienté I.A., contrairement à LISP ou PROLOG. Alors pourquoi ce choix ?

2.2. REPONSE

Les caractéristiques des connaissances à représenter nous ont fait opter pour un langage riche en structures de contrôle. Parmi cette classe, le langage C possède certains attributs qui nous ont paru intéressants. Cela ne veut pas dire que le langage C soit, de manière générale, un "bon" langage pour développer un S.E.S., nous citerons quelques caractéristiques d'un tel langage.

1) Caractéristiques des connaissances à représenter

On distingue généralement deux types de connaissance : une connaissance dite déclarative, qui peut être caractérisée comme une connaissance sans information de contrôle, notamment sans information sur une séquence d'exécution [1], et une connaissance dite procédurale.
Une grande partie de la connaissance qui nous intéresse ici est de type procédural. En effet, la démarche d'analyse d'une série chronologique est bien connue, elle peut être représentée sous forme d'un algorithme. De plus, les tâches logiques et terminales contiennent aussi des informations de contrôle. C'est pourquoi, nous avons décidé de représenter la connaissance à l'aide d'un langage riche en structures de contrôle.

2) Caractéristiques intéressantes du langage C

Nous avons également choisi le langage C pour :

- ses qualités d'effficacité et de portabilité,
- ses possibilités de contrôler le matériel, ce qui est indispensable pour réaliser une interface utilisateur,
- le développement qu'il connaît actuellement.

3) Caractéristiques d'un langage pour développer un S.E.S.

Pour être un bon outil de développement d'un S.E., un langage devrait, à notre avis, avoir au moins les propriétés suivantes :

- il devrait supporter les structures de listes et d'arbres, car ce sont des objets commodes pour y mémoriser des informations (base de connaissances ou autres),
- il devrait supporter les fonctions récursives, car la récursion est un bon moyen pour construire des chaînes d'inférence.

Pour développer un S.E.S., on souhaitera également la caractéristique suivante :

- il devrait supporter des outils mathématiques de haut-niveau et efficaces.

Des langages tels que LISP ou PROLOG offrent certainement les deux premières caractéristiques, mais pas le troisième. Or celle-ci était indispensable pour développer notre S.E.B.J.

Le langage C ne supporte pas directement des structures de type liste, mais moyennant un effort de programmation, de tels objets peuvent être construits.

3. QUELLES SONT LES FAIBLESSES DU SYSTEME ?

3.1. QUESTION

Les auteurs de REX (voir II.3.2.) soulevaient deux difficultés rencontrées lors du développement de leur système :

- l'explicitation d'une stratégie robuste,
- la réalisation d'un module d'explication.

Ces deux points sont-ils satisfaisants pour le S.E.B.J. présenté ?

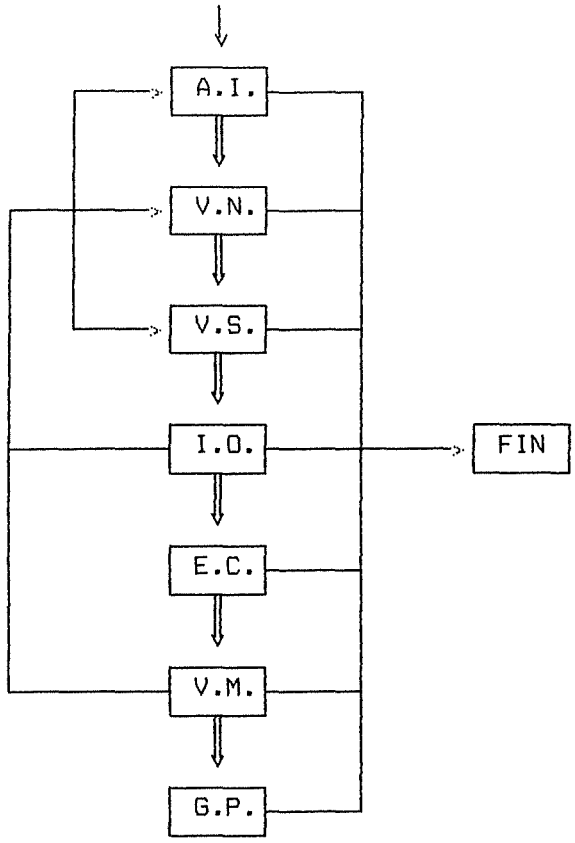
3.2. REPONSE

3.2.1. Amélioration de la stratégie

La stratégie ne pourra être considérée satisfaisante qu'après l'avoir confrontée à un grand nombre d'exemples. Toutefois, elle pourrait déjà être sensiblement améliorée.

Dans l'annexe A.2. nous avons suggéré des alternatives pour implémenter les tâches logiques et terminales. Notre système serait plus "intelligent" s'il connaissait toutes ces possibilités et s'il était capable d'en choisir une pour exécuter une tâche.

Dans l'état actuel, la stratégie est beaucoup trop rigide. La contrainte de flexibilité décrite en I.3.1.1. n'est pas respectée. En effet, le fait d'expliciter la stratégie sous la forme d'une planification hiérarchique détermine une séquence unique de tâches à accomplir. Le système serait plus souple s'il permettait à l'utilisateur de définir une autre séquence. Nous pensons dès lors qu'il serait préférable d'expliciter la stratégie sous la forme d'un graphe de transitions. Il définirait les transitions possibles entre les diverses tâches. Un tel graphe pourrait être :



Avec

A.I. : Analyse Initiale
Tendance, saisonnalité,...

V.N. : Vérification de normalité

V.S. : Vérification de stationnarité

I.O. : Identification de l'ordre

E.F. : Estimation des coefficients

V.M. : Validation du modèle

G.P. : Génération de prévisions

⇓ : Chemin suivi par défaut par le système

Cette façon de voir la stratégie est certainement plus réaliste. Un chemin dans le graphe définit une stratégie particulière. C'est d'ailleurs sous cette forme qu'ont été epxlicitées les stratégies de GLIMPSE [2] et du système A⁴ [3].
Elle implique toutefois une gestion beaucoup plus compliquée des tâches : il ne suffit plus de savoir "ce qu'il reste à faire", il faut aussi savoir "ce qui a déjà été fait" et "ce que l'on peut encore faire".

3.2.2. Amélioration des explications

Dans bien des cas, il serait souhaitable qu'une explication de type Why soit accompagnée d'un graphique. Ainsi par exemple, pour justifier une valeur du paramètre δ, le système pourrait afficher deux Range-Mean-Plot, l'un avant la transformation δ et l'autre après.

REFERENCES CHAPITRE IX.

---

[1] SCHOR M.I., Declarative Knowledge Programming : Declarative better than procedural, p 37
[2] WOLSTENHOLME D.E.,NELDER J.A., A front end for GLIM
[3] STREITBERG B.,NAEVE P., A modestly intelligent system for identification, estimation and forecasting of univariate time series : A4 ARIMA, artificial intelligence, and APL2

CONCLUSIONS

Pour conclure ce travail, nous voudrions donner une réponse personnelle à quelques questions fréquemment abordées dans la littérature.

1. UN S.E.S. : EST-CE UTILE ET EST-CE FAISABLE ?

"Dans l'absolu c'est certainement utile, mais, à cause de quelques obstacles techniques, ce qui est actuellement faisable n'est pas très utile."

Il est clair et indiscutable que les systèmes statistiques actuels sont conçus pour des statisticiens. Cela signifie que, s'il est vrai qu'à l'avenir, ils seront utilisés par d'autres classes d'utilisateurs, alors ils devront subir des améliorations. Ces améliorations concernent notamment l'ergonomie des systèmes et leur capacité de protéger l'utilisateur contre les erreurs sémantiques. Dans l'absolu, c'est-à-dire abstraction faite de toute contrainte, on peut imaginer des systèmes intelligents, incorporant des connaissances statistiques, des stratégies leur permettant d'assister efficacement un utilisateur. Dans ce contexte, un S.E.S. est certainement utile.

Mais dans la réalité, il faut tenir compte de certaines contraintes : la plus importante est, à notre avis, la difficulté d'expliciter une stratégie statistique. Ceci a pour conséquence que les S.E.S. actuellement réalisables concernent des domaines statistiques très spécialisés. Dans ces conditions, un S.E.S. perd une partie de son utilité.
En effet, un utilisateur soucieux d'acquérir un système pourrait baser sa décision sur le tableau simpliste suivant :

	SYSTEMES ACTUELS	SYSTEMES EXPERTS
AVANTAGES	Généralité	Sécurité d'utilisation
INCONVENIENTS	Difficulté d'utilisation	Spécificité

Nous pensons que dans bien des cas, l'attrait de la généralité l'emporterait.

Cela dit, ne soyons pas pessimstes, un S.E.S, même limité à un petit domaine est un premier pas dans la bonne direction. Regardons plutôt le chemin qui reste à parcourir.

2. QUE RESTE-T-IL A FAIRE ?

"Il faut étendre les domaines d'application des S.E.S. et les développer dans un but commercial."

Si l'on reprend les phases d'une analyse statistique : (1) Conception de l'analyse, (2) Conduite de l'analyse et (3) Interprétation des résultats, il apparaît que les phases (1) et (3) ont fait l'objet de très peu de recherches. La première fait souvent intervenir des connaissances extra statistique et l'on peut comprendre qu'elle soit inappropriée pour un S.E.S. Pour la dernière, par contre, il y a un besoin urgent d'idées et de travaux. Quant aux systèmes qui abordent la seconde phase, leur couverture est souvent minimale. Ainsi par exemple, si l'on considère les interfaces intelligentes (soit REX) pour des systèmes statistiques (soit S), elles ne couvrent qu'une infime partie des possibilités du système (l'analyse de régression).
La direction nous est donnée par TUKEY : "A separate system for each area of technique is all that will be feasible for a while ... When and if the time comes for systems to move closer together, this is likely to happen by bringing a variety of systems into a common environment rather than into a common system." [1]

La plupart des S.E.S. développés jusque maintenant l'ont été dans le but de démontrer leur faisabilité. Il serait bon de tester ces prototypes auprès de leurs utilisateurs potentiels afin de voir si effectivement ils sont bien acceptés et s'il y a lieu d'envisager une commercialisation. Aucun document consulté ne fait état d'un quelconque test d'acceptation.

Et si tout ceci était fait, que deviendraient les systèmes actuels ?

3. QUEL EST L'AVENIR DES SYSTEMES ACTUELS ?

"Idéalement, il serait préférable de concevoir des systèmes entièrement nouveaux, mais pour des raisons économiques, les systèmes actuels ont encore de longues années à vivre."

En ce qui concerne les choix de conception d'un S.E.S., deux grandes tendances s'opposent : soit on développe une interface intelligente pour un système existant, soit on développe un S.E.S. qui a ses propres routines de calcul. Idéalement, il semble préférable que le S.E.S puisse disposer de ses propres routines numériques. En effet, cela lui permet de raisonner et de fournir à l'utilisateur des explications plus détaillées. Un gain d'efficacité devrait également en résulter.

Mais dans la réalité, les sommes d'ores et déjà investies dans les systèmes actuels sont telles qu'il est vraisemblable qu'un choix économique favorisera les interfaces intelligentes.

4. QUELS SONT LES POINTS CLES DANS LE DEVELOPPEMENT D'UN S.E.S. ?

"Les conditions de succès d'un S.E.S. sont :
- une bonne définition des utilisateurs potentiels,
- une bonne stratégie statistique,
- une implémentation flexible."

Une "bonne" définition de l'utilisateur doit préciser son niveau de connaissances sémantiques. Tout ce qu'il peut ne pas connaître définit ce que le S.E.S doit être capable d'expliquer. Son niveau de connaissances définit également un style d'interaction. Pour le développement du S.E.B.J., la définition de l'utilisateur s'est révélée bien insuffisante.

Une "bonne" stratégie statistique, ce n'est pas forcément une méthode qui conduit à coup sûr à la meilleure solution. Une bonne stratégie serait plutôt une stratégie capable de traiter, c'est-à-dire donner une ou plusieurs solutions pour XX% des nouveaux problèmes qu'on lui soumettrait et qui connaîtrait ses limites en détectant les (100-XX)% qu'elle ne peut pas traiter. Une excellente valeur de XX serait par exemple 90. Il est clair que ce point clé nécessite la participation des statisticiens. Selon cette règle, la stratégie du S.E.B.J. n'est pas "bonne" : même si l'on n'a pas mesuré la valeur de XX, on peut déjà affirmer que le S.E.B.J. ne connaît pas ses limites.

Une implémentation flexible signifie notamment la possibilité de modifier la stratégie. Nous avons déjà critiqué le S.E.B.J. sur ce point (voir IX.3.).

Nous terminerons par cette phrase encourageante de PREGIBON : "You will be successful to the extent that you persevere. This may mean throwing away your first try and starting over again even after investing many man-hours. Do not give up. You will find the going easier the second time through. Easier yet the third. And easier yet the ..."[2]

REFERENCES CONCLUSION

---

[1] TUKEY J.W., Comments on More Intelligent Statistical Software and Statistical Expert Systems : Future Directions, p 12

[2] PREGIBON D., A DIY Guide to Statistical Strategy , p 398

ANNEXES

ANNEXE A.1.

LA METHODE BOX-JENKINS

TABLE DES MATIERES

INTRODUCTION

Cette annexe présente la méthode d'analyse de séries chronologiques de BOX - JENKINS. Il s'agit d'un résumé détaillé des fondements théoriques qui ont été exposés dans

BOX G.E.P., JENKINS G.M.
Time Series Analysis Forecasting and Control
Holden Day, 1976

Dans cette introduction, nous allons donner quelques principes généraux valables pour toute technique de prévision. Nous verrons comment caractériser la méthodes BOX - JENKINS par rapport à ces principes. Nous terminerons en schématisant la démarche préconisée par les auteurs, ce schéma nous servira de guide pour la suite de l'exposé.

1. LA PREVISION STATISTIQUE

1.1. DESCRIPTION DU PROBLEME

Données : Z_t : Une série chronologique (ou temporelle).
C'est-à-dire un ensemble d'observations générées séquentiellement dans le temps. La série peut en principe être continue ou discrète, mais en pratique, on dispose d'un ensemble discret de N observations. Nous noterons également Z_t l'observation au temps t.

Objectif : $Z'_t(h)$: Une fonction donnant les valeurs futures de la série.
C'est à dire prévoir, sur base des observations disponibles à l'instant t, la valeur de la série que l'on observerait au temps t+h : Z_{t+h} pour h=1,2,...

1.2. CARACTERISTIQUES D'UNE TECHNIQUE DE PREVISION

Afin d'atteindre son objectif, une technique de prévision construit un modèle pour expérimenter la réalité. Ce modèle est construit en faisant l'hypothèse qu'il existe une loi sous-jacente aux données, et selon un critère de précision. Les autres caractéristiques d'une technique sont son coût, son horizon temporel et son degré d'applicabilité.

1.2.1. Les types de modèles

D'une manière générale, on peut identifier trois classes non disjointes de modèles :

(1) Le modèle en série chronologique :
Il n'admet que deux variables, celle que l'on cherche à prévoir et la période de temps à laquelle on se réfère.

(2) Le modèle causal :
Il admet que la valeur d'une variable est fonction de plusieurs autres variables.

(3) Le modèle statistique :
Il utilise les procédés de l'analyse statistique pour identifier le modèle et pour déterminer le degré d'exactitude des prévisions.

1.2.2. La loi ou relation sous-jacente

Toutes les méthodes de prévision admettent l'existence d'une certaine loi ou relation qui peut être identifiée et utilisée comme base dans l'établissement de la prévision. Les quatre types de lois considérés habituellement et que l'on peut combiner sont :

(1) Une loi horizontale : la série est stationnaire
(2) Une loi saisonnière : la série fluctue de manière régulière
(3) Une loi cyclique : idem (2)
(4) Une loi de tendance : la série marque une croissance ou décroissance générale avec le temps.

1.2.3. La précision de la technique

L'hypothèse fondamentale impliquée par une technique de prévision est que la valeur réelle observée sera déterminée par une loi et par l'intervention du hasard :

$$\text{réel} = \text{loi} + \text{hasard}$$

La présence du terme aléatoire implique qu'il subsistera un écart entre valeurs prévues et valeurs observées, et ce ,même si l'on identifie exactement la loi sous-jacente:

$$e_{t+h} = Z'_t(h) - Z_{t+h}$$

Un but commun est de maximiser la précision, le plus souvent en rendant minimum la moyenne des carrés des écarts. Une technique précise est capable, non seulement de prédire la loi, mais aussi de s'adapter à une modification de la loi.

1.2.4. Le coût de la technique

Trois aspects de coût interviennent dans l'application d'une méthode de prévision :

(1) Les coûts de développement du modèle
(2) Le coût de stockage des données
(3) Les coûts d'exploitation.

1.2.5. L'horizon temporel et le degré d'applicabilté

Les deux derniers critères (que nous regroupons) permettant de différencier, voire d'évaluer, plusieurs techniques de prévision sont:

(1) L'horizon temporel pour lequel le modèle est valable : court, moyen, long terme.
(2) L'aptitude à l'application pratique de la méthode : temps de développement, interprétation des résultats, ...

1.3. PARTICULARITES DE LA METHODE B&J

1.3.1. Caractéristiques

En reprenant les critères énoncés ci-dessus, on peut caractériser la méthode B&J comme suit :

(1) Modèle : - statistique, en série chronologique,
- statistique, causal (nous n'en parlerons pas)
(2) Loi : les quatre types de loi sont combinés.
(3) Précision : elle est souvent reconnue comme étant la plus précise tant pour prédire la loi que les "tournants".
(4) Coût : le coût de la méthode est assez élevé et ce, dans chacun des trois aspects.
(5) Horizon : la méthode est plus adaptée pour les prévisions à court ou moyen terme.
(6) Applicabilité : les limites viennent du fait que c'est une méthode sophistiquée dont les principes sont difficiles à saisir.

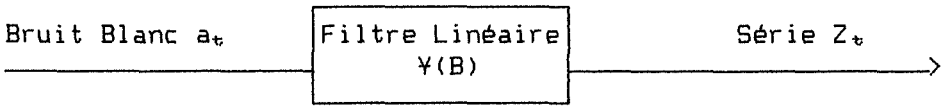
1.3.2. Idées de base de la méthode

Processus stochastique

Un phénomène statistique qui évolue dans le temps selon une loi de probabilité est appelé un processus stochastique.
Pour analyser une série temporelle, on la considère comme la réalisation d'un processus stochastique.

Filtre linéaire

Les modèles stochastiques employés sont basés sur l'idée que la série Z_t peut être générée à partir d'un processus stochastique particulier appelé bruit blanc et noté a_t.
Un bruit blanc est une séquence de variables aléatoires:
a_t, a_{t+1}, a_{t+2}, ... de distribution fixe et supposée $N(0,\sigma_a)$
Le bruit blanc a_t génère la série Z_t suite à l'application d'un filtre linéaire.



Eviter les décisions arbitraires

Contrairement à beaucoup d'autres approches, qui font des choix arbitraires, par exemple concernant :
- la nature de la fonction de prévision,
- une fonction de pondération,
- ...

la méthode B&J fournit des outils d'aide à la décision permettant d'orienter ces choix.
De plus, la technique met l'accent sur un principe de parcimonie qui consiste à n'inclure dans le modèle que le minimum de paramètres nécessaires.

2. CONSTRUCTION D'UN MODELE

2.1. PRINCIPES

L'approche itérative pour construire un modèle de prévision selon la méthode B&J comporte les étapes suivantes:

(1) Considérer une classe de modèles suffisamment générale pour s'adapter aux situations rencontrées dans la pratique.

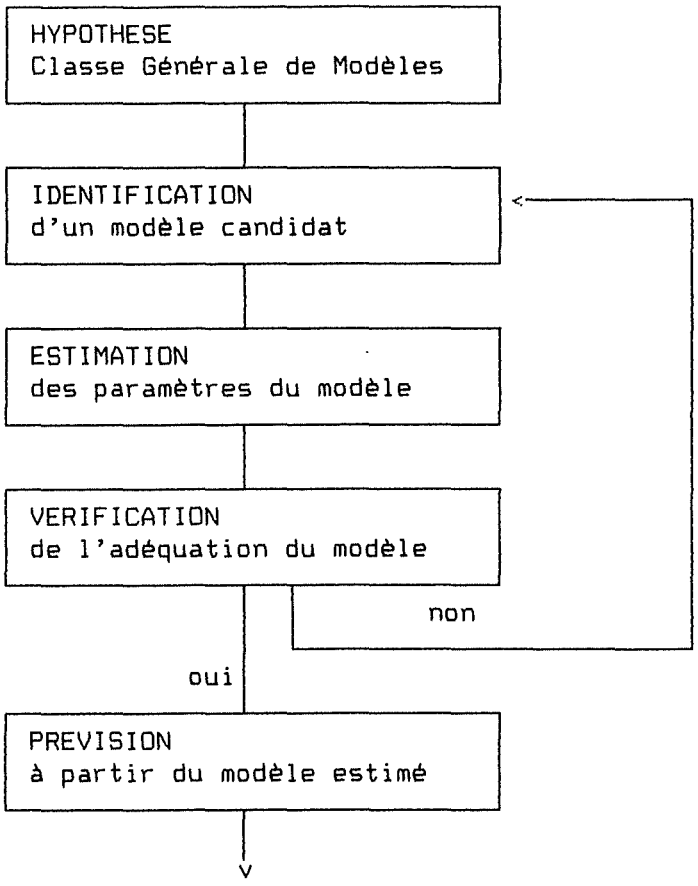
(2) Identifier des sous-classes de modèles
La série est analysée et un (ou plusieurs) modèle(s) suggéré(s).

(3) Estimer (ajuster) le modèle
Le modèle suggéré est ajusté aux données et ses paramètres estimés.

(4) Vérifier l'adéquation du modèle
Si le modèle est adéquat, on peut passer à l'étape suivante, sinon les causes d'inadéquation sont diagnostiquées et l'on retourne à l'étape (2)

(5) Utiliser le modèle pour de la prévision.

2.2. SCHEMA DE PRINCIPE



3. CONTENU

Nous allons suivre ce schéma en présentant d'abord la classe générale de modèles que nous considérons, ensuite nous traitons séparément chacune des étapes de la construction d'un modèle.

I. CLASSE GENERALE DE MODELES STOCHASTIQUES

1. CONCEPTS PREALABLES

1.1. PROCESSUS STATIONNAIRE

Une classe très particulière de processus stochastiques, appelés processus stationnaires, est basée sur l'hypothèse que le processus est dans un "état d'équilibre statistique".

De manière plus formelle : un processus est stationnaire si ses propriétés sont insensibles à un changement d'origine temporelle. Exprimé en terme de distribution de probabilité conjointe :

$$P(Z_{t1} , \ldots , Z_{tm}) = P(Z_{t1+k} , \ldots , Z_{tm+k})$$

En particulier (m=1), l'hypothèse de stationnarité implique que le processus a :
une moyenne constante : niveau autour duquel il fluctue,
une variance constante: mesure de la variation autour de la moyenne

1.2. FONCTION D'AUTOCORRELATION (A.C.F.)

On définit le coefficient d'autocorrélation au lag k : Ro_k en utilisant des paires de valeurs (Z_t , Z_{t+k}) de la série séparées par un intervalle (lag) constant k

$$Ro_k = (C_k)/(C_o) \qquad \text{avec } C_k = \text{cov}[Z_t, Z_{t+k}]$$

La fonction d'autocorrélation Ro(k) reprend le coefficient d'autocorrélation pour différentes valeurs du lag k.

Propriétés du coefficient Ro_k :

- $-1 < Ro_k < 1$
- si $Ro_k >> 0$ il y a une relation positive (variation dans le même sens) entre les paires de valeurs.

1.3. FONCTION D'AUTOCORRELATION PARTIELLE (P.A.C.F.)

Considérons les processus suivants (que nous appelerons plus tard autorégressifs) :

$$Z_t = \emptyset_{11} Z_{t-1} + a_t$$
$$Z_t = \emptyset_{21} Z_{t-1} + \emptyset_{22} Z_{t-2} + a_t$$
$$Z_t = \emptyset_{31} Z_{t-1} + \emptyset_{32} Z_{t-2} + \emptyset_{33} Z_{t-3} + a_t$$
$$\vdots$$
$$Z_t = \emptyset_{k1} Z_{t-1} + \ldots + \emptyset_{kk} Z_{t-k} + a_t$$

alors la séquence $\emptyset_{11}$, $\emptyset_{22}$, ... , $\emptyset_{kk}$, ... définit la fonction d'autocorrélation partielle pour différentes valeurs de k.

1.4. NOTATIONS

Les concepts que nous venons de présenter sont des caractéristiques théoriques d'un processus. En pratique, on ne peut qu'estimer les valeurs de ces caractéristiques à partir des valeurs disponibles de la série. Nous noterons :

	Valeur théorique	Estimation
Moyenne	μ_z	μ'_z
Ecart-type	σ_z	σ'_z
A.C.F.	Ro(k)	r(k)
P.A.C.F.	$\emptyset$(kk)	$\emptyset$'(kk)

Nous voudrions également introduire deux opérateurs qui seront fort utilisés par la suite :

B : Backward Shift Operator
$BZ_t = Z_{t-1}$
$B^n Z_t = Z_{t-n}$

∇ : Backward Difference Operator
$\nabla Z_t = Z_t - Z_{t-1} = (1-B)Z_t$
$\nabla^n Z_t = \quad\quad = (1-B)^n Z_t$
$\nabla_s Z_t = Z_t - Z_{t-s} = (1-B^s)Z_t$

Enfin nous convenons de noter
Z_t : la série qui correspond aux observations brutes (tout à fait quelconque)
W_t : une série stationnaire (réalisation d'un processus stationnaire)

2. MODELE LINEAIRE POUR SERIE STATIONNAIRE

Nous avons déjà mentionné que l'on considérait une série comme le résultat d'un filtre linéaire appliqué à un bruit blanc.

2.1. FILTRE LINEAIRE GENERAL

2.1.1. Deux formes équivalentes

(1) Filtre $\Psi(B)$

$$W_t' = a_t + \Psi_1 a_{t-1} + \Psi_2 a_{t-2} + \ldots$$

$$W_t' = \Psi(B) a_t$$

avec $W_t' = W_t - \mu_w$

$$\Psi(B) = 1 + \Psi_1 B + \Psi_2 B^2 + \ldots$$

(2) Filtre $\pi(B)$

$$W_t' = \pi_1 W_{t-1}' + \pi_2 W_{t-2}' + \ldots + a_t$$

$$\pi(B) W_t' = a_t$$

avec $\pi(B) = 1 - \pi_1 B - \pi_2 B^2 - \ldots$

2.1.2. Conditions de stationnarité et d'inversibilité

Admettons sans plus d'explications pour l'instant que

- le processus linéaire $W_t = \Psi(B) a_t$ est stationnaire si la série $\Psi(B)$ converge pour tout $|B| \leq 1$
- le processus linéaire $\pi(B) W_t = a_t$ est inversible si la série $\pi(B)$ converge pour tout $|B| \leq 1$

Nous illustrerons ces conditions dans le paragraphe suivant.

2.2. FILTRES LINEAIRES PARTICULIERS

2.2.1. Modèle auto-régressif d'ordre p : AR(p)

Le premier processus d'utilité pratique est obtenu en considérant le modèle en filtre $\pi(B)$ avec un nombre fini p de paramètres : c'est le modèle AR(p) qui s'écrit :

$$W_t = \phi_1 W_{t-1} + \phi_2 W_{t-2} + \ldots + \phi_p W_{t-p} + a_t$$

$$\phi_p(B) W_t = a_t$$

avec $\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p$

2.2.2. Modèle moving-average d'ordre q : MA(q)

Un second processus intéressant est obtenu en considérant le modèle en filtre $\Psi(B)$ avec un nombre fini q de paramètres : c'est le modèle MA(q) qui s'écrit :

$$W_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \ldots - \theta_q a_{t-q}$$

$$W_t = \theta_q(B) a_t$$

avec $\theta_q(B) = 1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q$

2.2.3. Modèle auto-régressif moving-average d'ordre p,q : ARMA(p,q)

Les deux modèles précédents peuvent se combiner pour donner le modèle ARMA(p,q) qui s'écrit :

$$W_t - \phi_1 W_{t-1} - \ldots - \phi_p W_{t-p} = a_t - \theta_1 a_{t-1} - \ldots - \theta_q a_{t-q}$$

$$\phi_p(B) W_t = \theta_q(B) a_t$$

2.2.4. Conditions de stationnarité et d'inversibilité

(1) Condition de stationnarité pour un processus AR

Soit le modèle AR(1) : $(1 - \phi_1 B)W_t = a_t$

$$
\begin{aligned}
W_1 &= \phi_1 W_0 + a_1 \\
W_2 &= \phi_1 W_1 + a_2 \\
&= \phi_1^2 W_0 + \phi_1 a_1 + a_2 \\
&\vdots \\
W_t &= \phi_1 W_{t-1} + a_t \\
&= \phi_1^t W_0 + \phi_1^{t-1} a_1 + \phi_1^{t-2} a_2 + \ldots + a_t
\end{aligned}
$$

On peut assurer la stationnarité de ce processus en imposant que ϕ_1 soit compris entre -1 et 1. Si ce n'était pas le cas, W_t serait principalement influencé par les événements les plus lointains W_0 et a_1. Par, contre imposer $-1<\phi_1<1$ revient à accorder plus d'importance aux événements récents a_t, ce qui paraît raisonnable.

Une autre façon d'exprimer cette condition est de dire que les racines de l'équation

$1 - \phi_1 B = 0$

avec B considéré comme variable, doivent être en dehors de l'intervalle [-1,1].

Si l'on prend la condition générale de stationnarité exprimée en 1.2., il faut que la série

$\Psi(B) = (1 - \phi_1 B)^{-1} = \Sigma \phi_j B^j \quad (j=0..\infty)$

converge pour $|B| \leq 1$.
La condition sera satisfaite pour $|\phi_1|<1$. C'est bien le résultat intuitif que nous avions trouvé.

(2) Condition d'inversibilité pour un processus MA

Un raisonnement semblable appliqué au modèle MA(1) conduit à la conclusion que θ_1 doit satisfaire $-1<\theta_1<1$

(3) Généralisation

Les conditions de stationnarité et d'inversibilité d'un processus ARMA(p,q) seront satisfaites pour autant que les racines (réelles ou complexes) de

$\phi_p(B) = 0$
$\theta_q(B) = 0$

soient situées en dehors du cercle unitaire.

2.3. RESULTATS THEORIQUES : SYNTHESE

	Processus AR(p)	Processus MA(q)	Processus ARMA(p,q)
Modèle	$\phi_p(B)W_t = a_t$ $W_t = \phi_p^{-1}(B)a_t$	$\theta_q^{-1}(B)W_t = a_t$ $W_t = \theta_q(B)a_t$	$\theta_q^{-1}(B)\phi_p(B)W_t = a_t$ $W_t = \phi_p^{-1}(B)\theta_q(B)a_t$
Série : en π	finie	infinie	infinie
en Ψ	infinie	finie	infinie
Stationnarité	racines de $\phi_p(B) = 0$ en dehors du cercle unit.	toujours stationnaire	racines de $\phi_p(B) = 0$ en dehors du cercle unit.
Invertibilité	toujours inversible	racines de $\theta_q(B) = 0$ en dehors du cercle unit.	racines de $\theta_q(B) = 0$ en dehors du cercle unit.
A.C.F.	infinie : décroissance - exponentielle - exp. avec altern. signe - en forme vague sinus	finie : q valeurs différentes de 0	infinie : décroissance - exponentielle - exp. avec altern. signe - en forme vague sinus
P.A.C.F.	finie : p valeurs différentes de 0	infinie : décroissance - exponentielle - exp. avec altern. signe - en forme vague sinus	infinie : décroissance - exponentielle - exp. avec altern. signe - en forme vague sinus

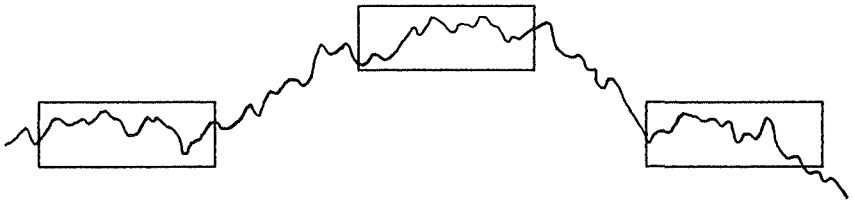
3. MODELE LINEAIRE POUR SERIE NON STATIONNAIRE HOMOGENE

En pratique, les séries sont rarement stationnaires. Toutefois, elles présentent souvent une homogénéité en ce sens que l'on peut se ramener à une série stationnaire en analysant la série "différenciée". Un modèle peut alors être construit.

3.1. MODELE AUTO-REGRESSIF INTEGRATED MOVING-AVERAGE : ARIMA(p,d,q)

3.1.1. Cas simple : d=1

Soit la série Z_t



Elle n'est pas stationnaire, mais mis à part une translation verticale, une partie de la série ressemble fort à une autre. On dit que Z_t est stationnaire dans ses différences premières. C'est à dire que la série

$$W_t = \nabla Z_t = (1 - B)Z_t$$

est stationnaire.
On pourra utiliser un modèle ARMA(p,q) pour représenter W_t.

Z_t sera représentée par un modèle ARIMA(p,d,q) où d=1 et signifie que l'on considère non pas les valeurs brutes mais les différences premières.

3.1.2. Généralisation

Si Z_t est la série dont on souhaite représenter le comportement, alors on suppose que la série en d^{me} différence $\nabla^d Z_t = W_t$ est stationnaire et peut être représentée par un modèle du paragraphe II.1.2.2.

Le modèle pour Z_t est un ARIMA(p,d,q) et s'écrit :

$$\emptyset_p(B)\nabla^d Z_t = \theta_q(B)a_t$$

3.2. EXTENSION DU MODELE

3.2.1. Terme constant

Il est parfois nécessaire d'introduire un terme constant dans le modèle. La forme la plus générale du ARIMA(p,d,q) est donc

$$\emptyset_p(B)\nabla^d Z_t = \theta_0 + \theta_q(B)a_t$$

3.2.2. Transformation préalable

L'applicabilité du modèle peut être considérablement étendue si l'on admet la possibilité de transformation non linéaire de la série Z_t.

Le but d'une transformation est d'éviter que l'amplitude de la variation des observations autour de la moyenne dépende de cette valeur moyenne.

Les transformations proposées dépendent d'un paramètre δ, à déterminer :

$Y_t = Z_t^{\delta}$ pour $\delta \neq 0$
$Y_t = \ln(Z_t)$ pour $\delta = 0$

L'analyse porte ensuite sur la série transformée Y_t

II. CONSTRUCTION D'UN MODELE STOCHASTIQUE

1. IDENTIFICATION D'UN MODELE

Les méthodes d'identification sont des procédures grossières appliquées à un ensemble de données pour indiquer le type de modèle qui semble adéquat pour représenter les données.

Dans notre cas, l'objectif est d'obtenir une idée de la valeur des paramètres
- δ pour identifier une transformation préalable,
- p,d,q pour identifier une sous-classe de modèles hors de la famille générale ARIMA(p,d,q)

De plus, il est intéressant d'obtenir des estimations initiales des valeurs des paramètres $\emptyset_i$, θ_j avant la phase d'estimation proprement dite.

C'est une étape qui demande une bonne part de jugement : les choix se font essentiellement en confrontant des représentations graphiques tirées des observations (A.C.F., P.A.C.F., ...) avec des résultats théoriques.

1.1. CHOIX D'UNE TRANSFORMATION PREALABLE : δ

1.1.1. Méthode rigoureuse

C'est une méthode assez lourde qui consiste à :
Pour différentes valeurs de δ
- ajuster un modèle à $Z^{(\delta)} = (Z^{\delta} - 1)/(\delta Z_g^{\delta-1})$
  avec Z = données brutes
  Z_g = moyenne géométrique de la série
  $Z^{(0)} = Z \ln(Z)$

- calculer la somme des carrés des résidus S_δ

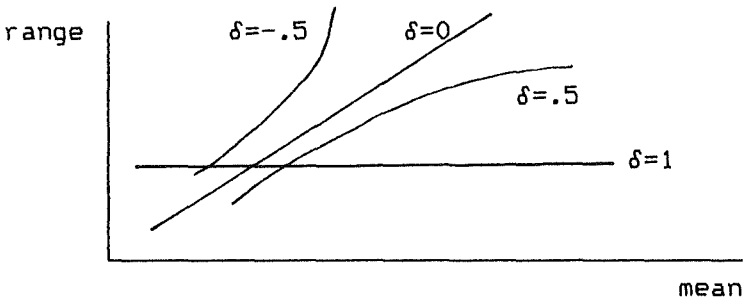
Choisir δ pour lequel S_δ est minimum.

1.1.2. Méthode graphique : Range-Mean Plot

Heureusement, une simple analyse graphique peut souvent suggérer une transformation.

La méthode consiste à "saucissonner" la série en plusieurs portions, pour chacune des portions on calcule la moyenne et l'écart entre la valeur minimale et maximale. On obtient donc des paires (mean , range) que l'on peut représenter sur un graphique.

La forme du graphe suggère δ :



1.2. IDENTIFICATION DU DEGRE DE DIFFERENCE

On utilise la propriété suivante des modèles stationnaires ARMA(p,q) : leur A.C.F. diminuera rapidement à partir d'un lag k "raisonnablement grand".

La méthode consiste donc à examiner l'A.C.F. des séries Z_t, ∇Z_t, $\nabla^2 Z_t$, ... On suppose que le degré de différence d, nécessaire pour obtenir une série stationnaire, a été atteint lorsque l'A.C.F. de $W_t = \nabla^d Z_t$ diminue assez rapidement.

En pratique d vaut 0,1,ou 2 et il est suffisant d'inspecter r(k) pour $k \leq 20$.

1.3. IDENTIFICATION DU PROCESSUS STATIONNAIRE ARMA RESULTANT : p,q

Après avoir décidé quel d utiliser, on identifie p et q en examinant simultanément les A.C.F. et P.A.C.F. et on les compare au comportement théorique des A.C.F. et P.A.C.F. des processus les plus courants en pratique.

Le tableau suivant est particulièrement utile : il donne le comportement des A.C.F. et P.A.C.F. d'un processus ARIMA(p,d,q) pour des valeurs courantes de p et q :

Ordre	(1,d,0)	(0,d,1)
Ro(k) Ø(kk)	décrois. exponentielle seul $Ø_{11} \neq 0$	seul $Ro_1 \neq 0$ décrois. à domin. expon.
Ordre	(2,d,0)	(0,d,2)
Ro(k) Ø(kk)	décrois. exponentielle, décrois en vague sinus seuls $Ø_{11}$, $Ø_{22} \neq 0$	seuls Ro_1, $Ro_2 \neq 0$ décrois. à domin. expon. décrois en vague sinus
Ordre	(1,d,1)	
Ro(k) Ø(kk)	décrois. exponentielle après le premier lag décrois. à domin. expon. après le premier lag	

Encore plus utile serait la traduction de ce tableau en graphes typiques pour les processus courants.

Pour pouvoir juger si les coefficients d'autocorrélation et d'autocorrélation partielle s'annulent effectivement après un certain lag, on calcule les erreurs standards de la manière suivante :

$$\sigma'[r_k] = 1/\sqrt{n} \cdot \{1 + 2.(r_1^2 + r_2^2 + \ldots + r_q^2)\} \quad \text{pour } k>q$$

$$\sigma'[Ø'_{kk}] = 1/\sqrt{n} \quad \text{pour } k>p$$

(on fait l'hypothèse que le processus est autorégressif d'ordre p)

En pratique, on considérera comme significativement $\neq 0$ les valeurs qui sortent de l'intervalle $[-2\sigma', 2\sigma']$.

1.4. CHOIX D'INCORPORER UN TERME CONSTANT

Pour décider de l'incorporation d'une constante θ_0 dans le modèle, on applique la règle suivante :

Si d=0 alors on inclut d'office une constante

Si d≥1 alors
- calculer μ'_w : moyenne de $W_t = \nabla^d Z_t$
- calculer $\sigma'(\mu'_w) = 1/\sqrt{n} \cdot \sigma'_w \cdot \sqrt{\{1 + 2.(r_1 + r_2 + \ldots)\}}$
- si $0 \notin [\mu'_w \pm 2\sigma'(\mu'_w)]$
  alors introduire une constante

1.5. ESTIMATION INITIALE DES PARAMETRES $\emptyset$, θ

A titre indicatif, nous donnons dans le tableau suivant les équations et conditions qui permettent de calculer des estimations préliminaires pour les paramètres du modèle identifié. Nous avons noté R_k les coefficients théoriques de l'A.C.F. (Ro(k)).

Ordre	(1,d,0)	(0,d,1)
Equations	$\emptyset_1 = R_1$	$R_1 = \dfrac{-\theta_1}{1 + \theta_1^2}$
Conditions	$-1 < \emptyset_1 < 1$	$-1 < \theta_1 < 1$
Ordre	**(2,d,0)**	**(0,d,2)**
Equations	$\emptyset_1 = \dfrac{R_1(1-R_2)}{1 - R_1^2}$ $\emptyset_2 = \dfrac{R_2 - R_1^2}{1 - R_1^2}$	$R_1 = \dfrac{-\theta_1(1 - \theta_2)}{1 + \theta_1^2 + \theta_2^2}$ $R_2 = \dfrac{-\theta_2}{1 + \theta_1^2 + \theta_2^2}$
Conditions	$-1 < \emptyset_2 < 1$ $\emptyset_2 + \emptyset_1 < 1$ $\emptyset_2 - \emptyset_1 < 1$	$-1 < \theta_2 < 1$ $\theta_1 + \theta_2 < 1$ $\theta_2 - \theta_1 < 1$
Ordre	**(1,d,1)**	
Equations	$R_1 = \dfrac{(1 - \theta_1.\emptyset_1)(\emptyset_1 - \theta_1)}{1 + \theta_1^2 - 2.\emptyset_1.\theta_1}$ $R_2 = R_1.\emptyset_1$	
Conditions	$-1 < \emptyset_1 < 1$ $-1 < \theta_1 < 1$	

2. ESTIMATION DU MODELE

Cette étape a pour but d'obtenir des estimations efficaces des paramètres du modèle identifié.

Les méthodes utilisées consistent à déterminer p+q+1 paramètres ($\emptyset_p^?$, $\theta_q^?$, $\sigma_a^?$) de manière à :

- soit maximiser une fonction de vraisemblance,
- soit minimiser la somme des carrés des écarts entre valeurs observées et valeurs estimées.

Nous ne détaillerons pas ces méthodes, qui sont tout à fait automatisables. Nous supposons qu'un programme informatique nous fournit, pour le modèle identifié, les résultats suivants :

- les estimations des paramètres : $\emptyset_p^?$, $\theta_q^?$,
- les écarts-types de ces paramètres,
- les corrélations entre ces paramètres,
- l'estimation de la variance résiduelle : $(\sigma_a^?)^2$.

3. VERIFIATION DU MODELE

L'objectif de cette étape est de décider si le modèle ajusté est adéquat ou non. S'il ne l'est pas, un but complémentaire est de savoir comment il faut le modifier pour une nouvelle itération. C'est une étape qui demande à nouveau du jugement.

Deux techniques sont utilisables :

- la sur-spécification d'un modèle,
- la vérification appliquée aux résidus du modèle ajusté.

3.1. SUR-SPECIFICATION

Le principe de cette technique consiste à ajuster un modèle plus élaboré, contenant des paramètres supplémentaires.

Le modèle initial sera adéquat si le modèle plus élaboré n'apporte pas d'amélioration sensible.

On remarquera que cette méthode n'est pas très rigoureuse : comment savoir "dans quelle direction" il faut augmenter le modèle ? C'est pourquoi, il est nécessaire de recourir à une méthode plus générale.

3.2. VERIFICATION APPLIQUEE AUX RESIDUS

Le principe est le suivant : si le modèle ajusté est adéquat, alors les résidus a_t doivent avoir les propriétés d'un bruit blanc
- ils suivent une distribution $N(0,\sigma_a^2)$
- ils sont non corrélés.

Dans cette technique, les résidus sont analysés pour vérifier cette hypothèse, mais ils sont également utilisés pour corriger le modèle si nécessaire.

3.2.1. Vérification de l'adéquation

(1) Graphe des résidus avec des limites de contrôle à $\pm 2\sigma_a$

C'est une première étape indispensable qui permet de juger si les a_t sont normalement distribués: il ne faut pas (ou très peu) de pics significatifs.

(2) Calculer l'A.C.F. des résidus : $r_k(a')$

Si le modèle ajusté est correct, des résultats théoriques montrent que les $r_k(a')$ doivent être non corrélés et distribués selon $N(0,1/\sqrt{n})$. Dès lors, on peut utiliser les $r_k(a')$ de deux manières : soit globalement , soit individuellement.

(a) Globalement : test du X^2

Supposons que l'on ait calculé les K premiers coefficients d'autocorrélation $r_k(a')$. Il est possible de montrer que, si le modèle est adéquat :

$$Q = n \cdot (r_1^2(a') + \ldots + r_K^2(a'))$$

est approximativement distribué selon un chi carré

$$X^2(K-p-q)$$

On peut donc adopter la règle suivante (test d'hypothèse)
Soit . Ho : les résidus sont non corrélés
. X^2_ν : la valeur critique théorique du X^2 à
v degrés de liberté et au seuil α
. Q : la valeur calculée par la formule ci-dessus.
Alors : . Si $Q<X^2_\nu$: on accepte Ho
. Sinon : on rejette Ho

(b) Individuellement

Un graphe des $r_k(a')$ avec des limites de contrôle à $\pm 2/\sqrt{n}$ permet de juger si les résidus sont distribués selon $N(0,1/\sqrt{n})$: il ne faut pas (ou très peu) de pics significatifs.

3.2.2. Utilisation des résidus pour modifier le modèle

Soit le modèle initial

$$\emptyset(B)\nabla^d Z_t = \theta(B)b_t$$

Supposons encore que les résidus b_t ne satisfont pas l'hypothèse de bruit blanc. Dès lors, on identifie un second modèle pour la série b_t :

$$\emptyset'(B)\nabla^{d'} Z_t = \theta'(B)a_t$$

En éliminant b_t entre les deux équations, on obtient un nouveau modèle :

$$\emptyset(B)\emptyset'(B)\nabla^d\nabla^{d'} Z_t = \theta(B)\theta'(B)a_t$$

qui peut être à nouveau estimé puis vérifié.

3.3. TESTS COMPLEMENTAIRES

Dans certains cas, le seul fait d'écrire le modèle suggère des simplifications.

3.3.1. En omettant des paramètres non significatifs

Soit le modèle ARIMA(0,1,2)

$$\nabla Z_t = (1 - .86B + .02B^2)a_t$$
$$\qquad\quad (\pm.10)\ (\pm.06)$$

L'écart-type de θ_2 suggère que l'on essaie un modèle plus simple ARIMA(0,1,1).

3.3.2. En éliminant des opérateurs redondants

Soit le modèle ARIMA(2,1,1)

$$(1 - 1.10B + .28B^2)\nabla Z_t = (1 - .65B)a_t$$
$$(1 - .70B)(1 - .40B)\nabla Z_t = (1 - .65B)a_t$$

Dans un souci de parcimonie, on devrait essayer un modèle plus simple ARIMA(1,1,0).

3.3.3. Choix entre modèles concurrents

Lorsque plusieurs modèles semblent adéquats, on les compare sur les critères suivants :

- Variance résiduelle : la plus petite possible,
- Nombre de paramètres : le plus petit possible,
- Valeur calculée du X^2 : le plus petit possible.

4. PREVISION

Le but de cette étape est d'utiliser un modèle ARIMA dont on a testé l'adéquation pour prévoir des valeurs futures de la série temporelle. En d'autres termes nous cherchons :

- $Z_t^?(h)$: une prévision de Z_{t+h} faite à l'instant t
- un intervalle de confiance associé à chaque prévision.

4.1. LA FONCTION DE PREVISION

4.1.1. Principe

Soit le modèle ajusté

$$\Omega(B)Z_t = \theta(B)a_t \qquad \text{avec } \Omega(B) = \phi(B)\nabla^d$$

Ce modèle peut s'écrire, en remplaçant t par t+h

$$Z_{t+h} = \Omega_1 Z_{t+h-1} + \ldots + \Omega_{p+d} Z_{t+h-p-d} - \theta_1 a_{t+h-1} - \ldots - \theta_q a_{t+h-q} + a_{t+h}$$

Les deux résultats théoriques suivants permettent de calculer les prévisions

(1) La meilleure prévision, faite en t, pour l'époque t+h, est l'espérance mathématique de Z_{t+h} conditionnée par la connaissance de tous les Z jusqu'au temps t. Nous noterons cela

$$Z_t^?(h) = [Z_{t+h}]$$

Ce qui permet d'écrire

$$[Z_{t+h}] = Z_t^?(h) = \Omega_1[Z_{t+h-1}] + \ldots + \Omega_{p+d}[Z_{t+h-p-d}] - \theta_1[a_{t+h-1}] - \ldots - \theta_q[a_{t+h-q}] + [a_{t+h}]$$

(2) On utilisera les règles de calcul suivantes :

$$\begin{aligned} [Z_{t-j}] &= Z_{t-j} && j=0,1,2,.. \\ [Z_{t+j}] &= Z_t^?(j) && j=1,2,.. \\ [a_{t-j}] &= a_{t-j} = Z_{t-j} - Z_{t-j-1}^?(1) && j=0,1,2,.. \\ [a_{t+j}] &= 0 && j=1,2,.. \end{aligned}$$

Ce qui permet de calculer les $Z_t^?(h)$ de manière récursive dans l'ordre $Z_t^?(1)$, $Z_t^?(2)$, ...

4.1.2 Exemple

Soit un modèle ARIMA(2,0,0) estimé comme suit :

$$(1 - 1.8B + .8B^2)Z_{t+h} = a_{t+h}$$
$$Z_{t+h} = 1.8Z_{t+h-1} - .8Z_{t+h-2} + a_{t+h}$$

Les prévisions que l'on peut générer sont :

$$Z_t'(1) = 1.8Z_t - .8Z_{t-1}$$
$$Z_t'(2) = 1.8Z_t'(1) - .8Z_t$$
$$:$$

4.2. LES INTERVALLES DE CONFIANCE

4.2.1. Principe

Réécrivons le modèle sous la forme d'une somme pondérée infinie de a_j. C'est à dire en utilisant le filtre linéaire $\Psi(B)$:

$$Z_{t+h} = a_{t+h} + \Psi_1 a_{t+h-1} + \ldots + \Psi_{h-1} a_{t+1} + \Psi_h a_t + \ldots$$
$$Z_t'(h) = \Psi_h a_t + \ldots$$
$$e_t(h) = Z_{t+h} - Z_t'(h) = a_{t+h} + \Psi_1 a_{t+h-1} + \ldots + \Psi_{h-1} a_{t+1}$$
$$\sigma'(h) = \sqrt{(1 + \Psi_1^2 + \ldots + \Psi_{h-1}^2)} \cdot \sigma_a'$$

On prendra comme intervalle de confiance (I.C.) au seuil α

$$Z_{t+h}(\pm) = Z_t'(h) \pm \mu_{\epsilon/2} \cdot \sigma'(h)$$

avec $\mu_{\epsilon/2}$ donné par la distribution normale réduite, et valant 1.96 au seuil α=5%

4.2.2. Exemple

Reprenons le modèle

$$(1 - 1.8B + .8B^2)Z_{t+1} = a_{t+1}$$

(1) On commence par calculer les Ψ_j en faisant correspondre les coefficients dans

$$\Omega(B)(1 + \Psi_1 B + \Psi_2 B^2 + \ldots) = \theta(B)$$

On obtient ainsi

$$\Psi_0 = 1$$
$$\Psi_1 = 1.8$$
$$\Psi_j = 1.8\Psi_{j-1} - .8\Psi_{j-2}$$

(2) On peut alors calculer un I.C., ainsi au seuil de 5%

$$Z_{t+2}(\pm) = Z_t'(2) \pm 1.96 \sqrt{(1 + 1.8^2)} \cdot \sigma_a'$$

III. MODELES SAISONNIERS

En pratique, les séries que l'on doit analyser présentent souvent un phénomène récurrent qui apparaît pour une certaine périodicité s. Par exemple, un phénomène annuel pour des données de ventes mensuelles se traduira par une période s=12.

On peut encore utiliser les méthodes décrites jusqu'ici pour analyser de telles séries, dites saisonnières.

1. LE MODELE MULTIPLICATIF

Supposons que l'on analyse une série dont les données sont des ventes mensuelles, de telle sorte que s=12

1.1. MODELE POUR LES VARIATIONS D'ANNEE A ANNEE

Une observation pour un mois particulier, soit avril, peut être liée avec les observations des mois d'avril précédents par un modèle de la forme :

$$\Phi_P(B^s)\nabla_s^D Z_t = \theta_Q(B^s)e_t$$

avec $\nabla_s = 1 - B^s$

Un modèle similaire peut relier les observations faites pour les mois de mars, ... Il est habituellement raisonnable de faire l'hypothèse que les P+Q coefficients Φ_i , θ_j sont identiques pour ces s modèles.

1.2. MODELE POUR LES VARIATIONS DE MOIS A MOIS

Les résidus (e_t pour avril, e_{t-1} pour mars, ...) sont généralement corrélés. Pour permettre cette dépendance, un second modèle est introduit, il a la forme

$$\phi_p(B)\nabla^d e_t = \theta_q(B)a_t$$

1.3. MODELE GENERAL

En éliminant le terme e_t entre ces deux modèles, on obtient le modèle multiplicatif général, qui s'écrit

$$\emptyset_p(B)\Phi_P(B^s)\nabla^d\nabla_s^D Z_t = \theta_q(B)\Theta_Q(B^s)a_t$$

On parlera d'un modèle ARIMA d'ordre $(p,d,q)x(P,D,Q)_s$

2. CONSTRUCTION D'UN MODELE SAISONNIER

Les méthodes décrites en II. CONSTRUCTION D'UN MODELE STOCHASTIQUE peuvent s'appliquer pour analyser une série saisonnière réaliste.

IV. UTILISATION DE LA METHODE

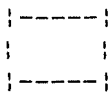
Dans ce chapitre, nous proposons de synthétiser les fondements théoriques,pour les rendre utilisables en pratique. Pour cela, nous reprenons les différentes étapes de la construction d'un modèle, nous les décomposons en sous-étapes, et nous assemblons ces sous-étapes sous la forme d'un organigramme.

En d'autres termes, le but est de représenter sous une forme simple, le raisonnement qu'un utilisateur de la méthode doit suivre pour construire un modèle.

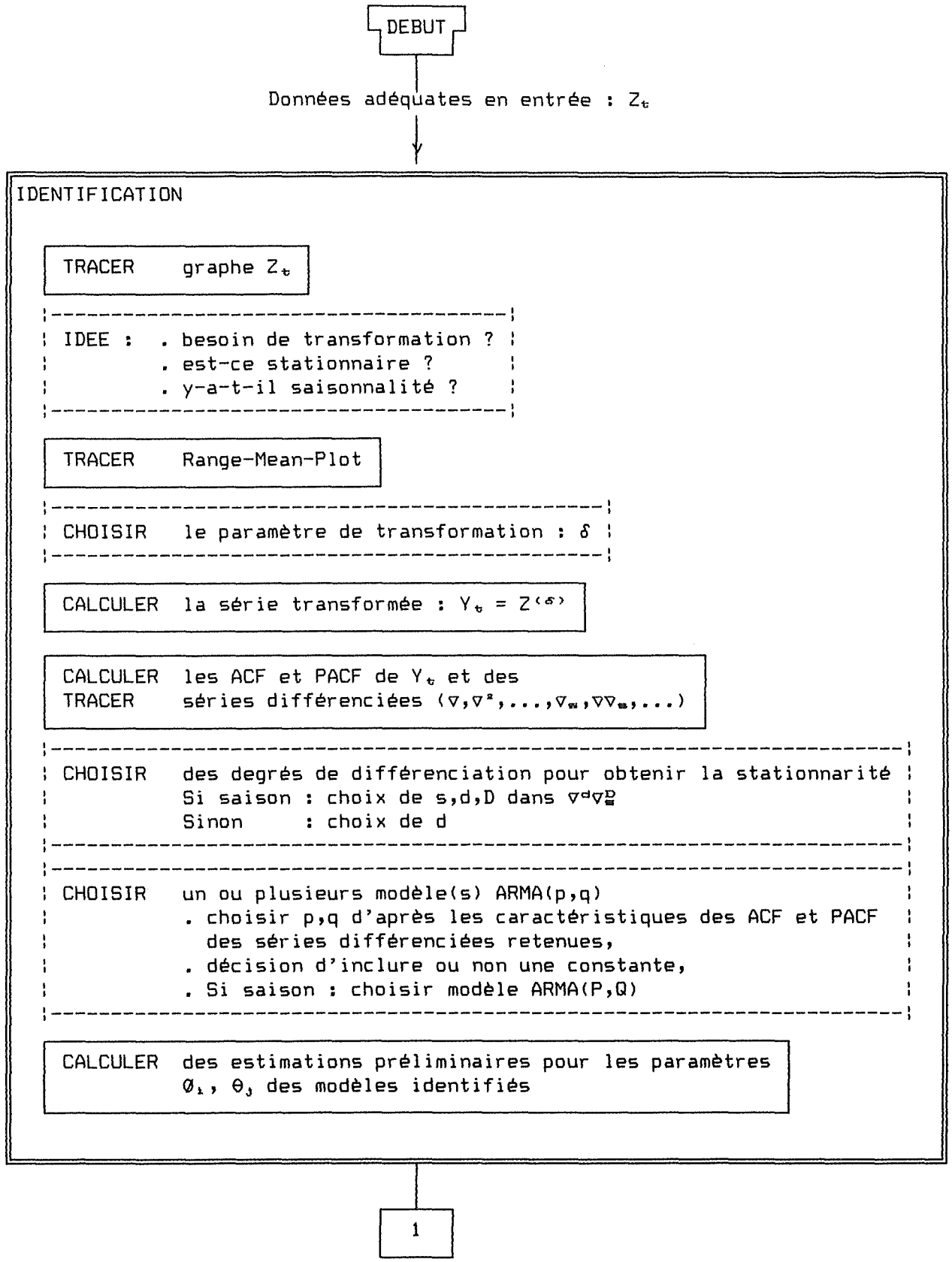
Dans ce schéma :

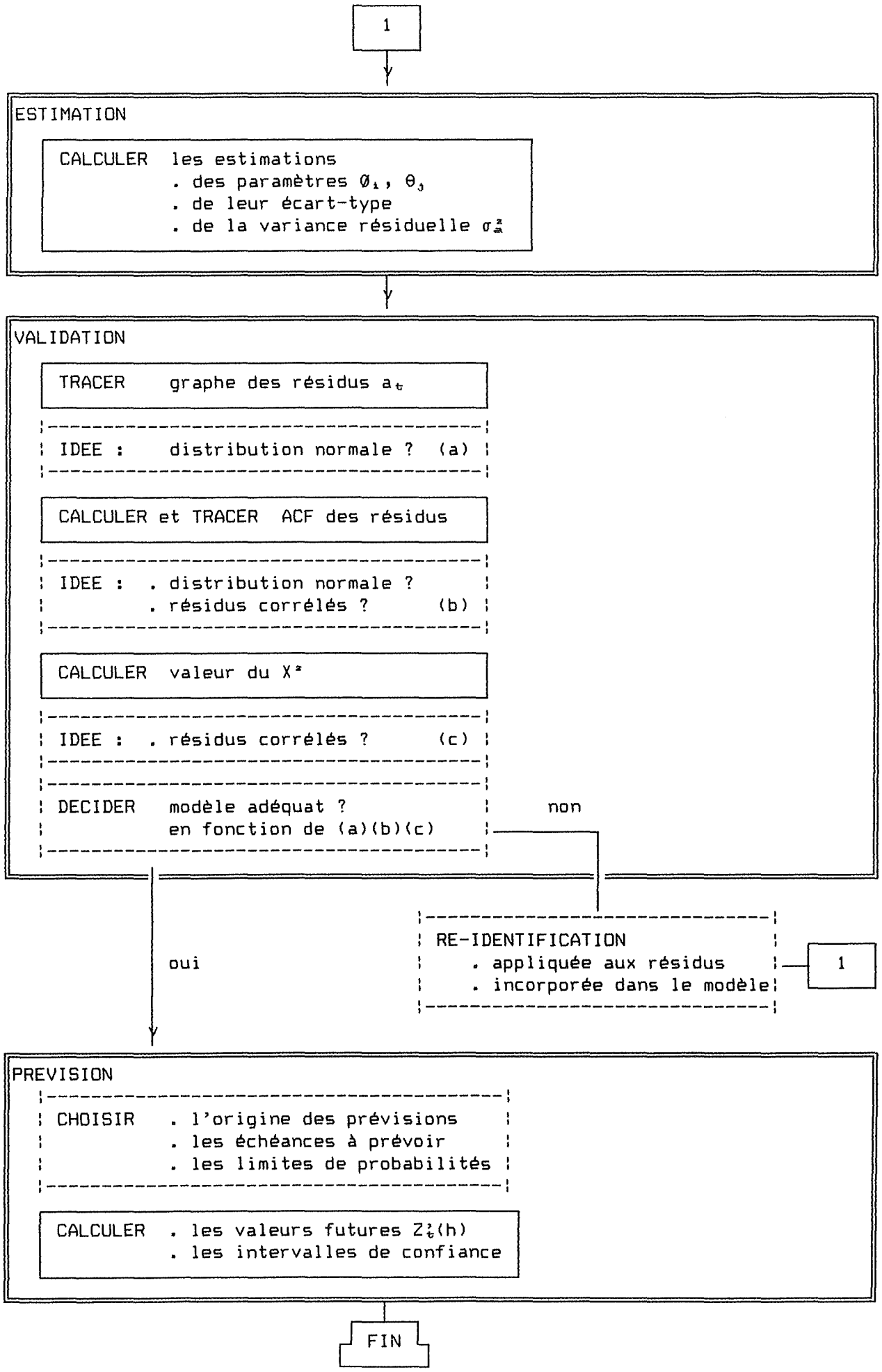
représente une étape principale dans la construction d'un modèle,

représente une sous-étape totalement automatisable,



représente une sous-étape nécessitant du jugement de la part de l'utilisateur.


DEBUT
Données adéquates en entrée : Z_t
IDENTIFICATION
TRACER graphe Z_t
IDEE : . besoin de transformation ?
. est-ce stationnaire ?
. y-a-t-il saisonnalité ?
TRACER Range-Mean-Plot
CHOISIR le paramètre de transformation : δ
CALCULER la série transformée : $Y_t = Z^{(\delta)}$
CALCULER les ACF et PACF de Y_t et des
TRACER séries différenciées ($\nabla, \nabla^2, \ldots, \nabla_m, \nabla\nabla_m, \ldots$)
CHOISIR des degrés de différenciation pour obtenir la stationnarité
Si saison : choix de s,d,D dans $\nabla^d \nabla_m^D$
Sinon : choix de d
CHOISIR un ou plusieurs modèle(s) ARMA(p,q)
. choisir p,q d'après les caractéristiques des ACF et PACF des séries différenciées retenues,
. décision d'inclure ou non une constante,
. Si saison : choisir modèle ARMA(P,Q)
CALCULER des estimations préliminaires pour les paramètres $\emptyset_i$, θ_j des modèles identifiés
1


1
ESTIMATION
CALCULER les estimations
. des paramètres ϕ_i, θ_j
. de leur écart-type
. de la variance résiduelle σ_a^2
VALIDATION
TRACER graphe des résidus a_t
IDEE : distribution normale ? (a)
CALCULER et TRACER ACF des résidus
IDEE : . distribution normale ?
. résidus corrélés ? (b)
CALCULER valeur du X^2
IDEE : . résidus corrélés ? (c)
DECIDER modèle adéquat ?
en fonction de (a)(b)(c)
non
oui
RE-IDENTIFICATION
. appliquée aux résidus
. incorporée dans le modèle
1
PREVISION
CHOISIR . l'origine des prévisions
. les échéances à prévoir
. les limites de probabilités
CALCULER . les valeurs futures $Z_t(h)$
. les intervalles de confiance
FIN

ANNEXE A.2.

IMPLEMENTATION DE LA STRATEGIE

INTRODUCTION

I. LA STRATEGIE

II. TRANSFORMATION

III. DIFFERENCIATION PROVISOIRE

IV. PERIODICITE

V. DIFFERENCIATION DEFINITIVE

VI. LONGUEUR DE LA SERIE

VII. ORDRE DU MODELE

TABLE DES MATIERES

INTRODUCTION

Dans cette annexe, nous reprenons la stratégie statistique (proposée dans le rapport en VI.) pour l'analyse des séries chronologiques selon la méthode B&J.

Rappelons que cette stratégie comprend des tâches décomposables, logiques et terminales. L'exécution d'une tâche logique doit permettre au système de suggérer l'exécution ou non d'une tâche terminale tandis que l'exécution d'une tâche terminale débouche sur la suggestion de valeurs de paramètres.

Pour chaque tâche logique et terminale de la stratégie d'identification, nous donnons une procédure permettant d'exécuter la tâche. Le lecteur trouvera donc pour chaque étape de la stratégie :

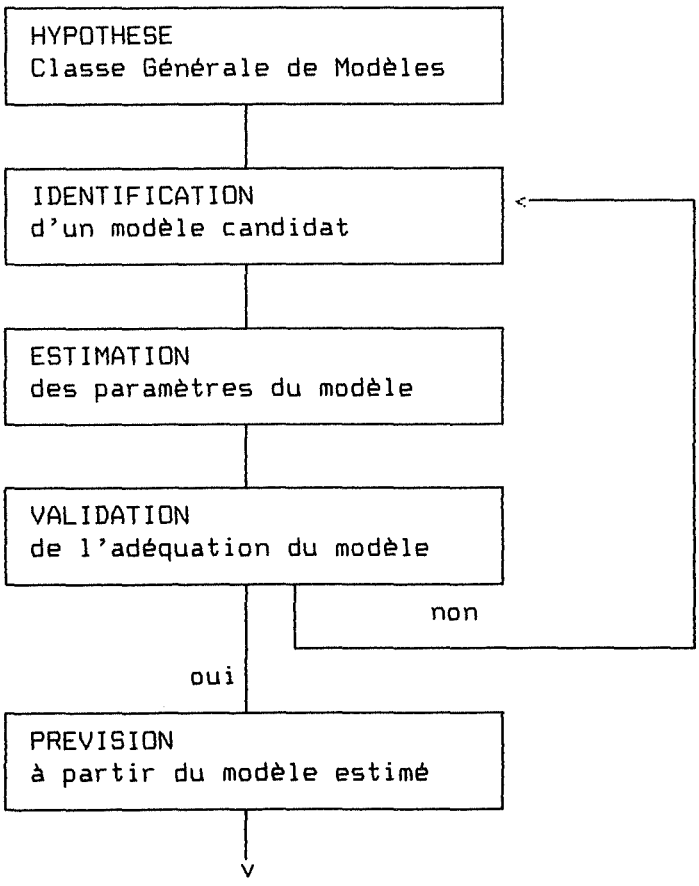
- L'objectif de l'étape,
- Le test, implémentant la tâche logique,
- Le choix, implémentant la tâche terminale.

Nous citerons plusieurs procédures possibles et nous spécifierons plus précisément celle que nous avons retenue.

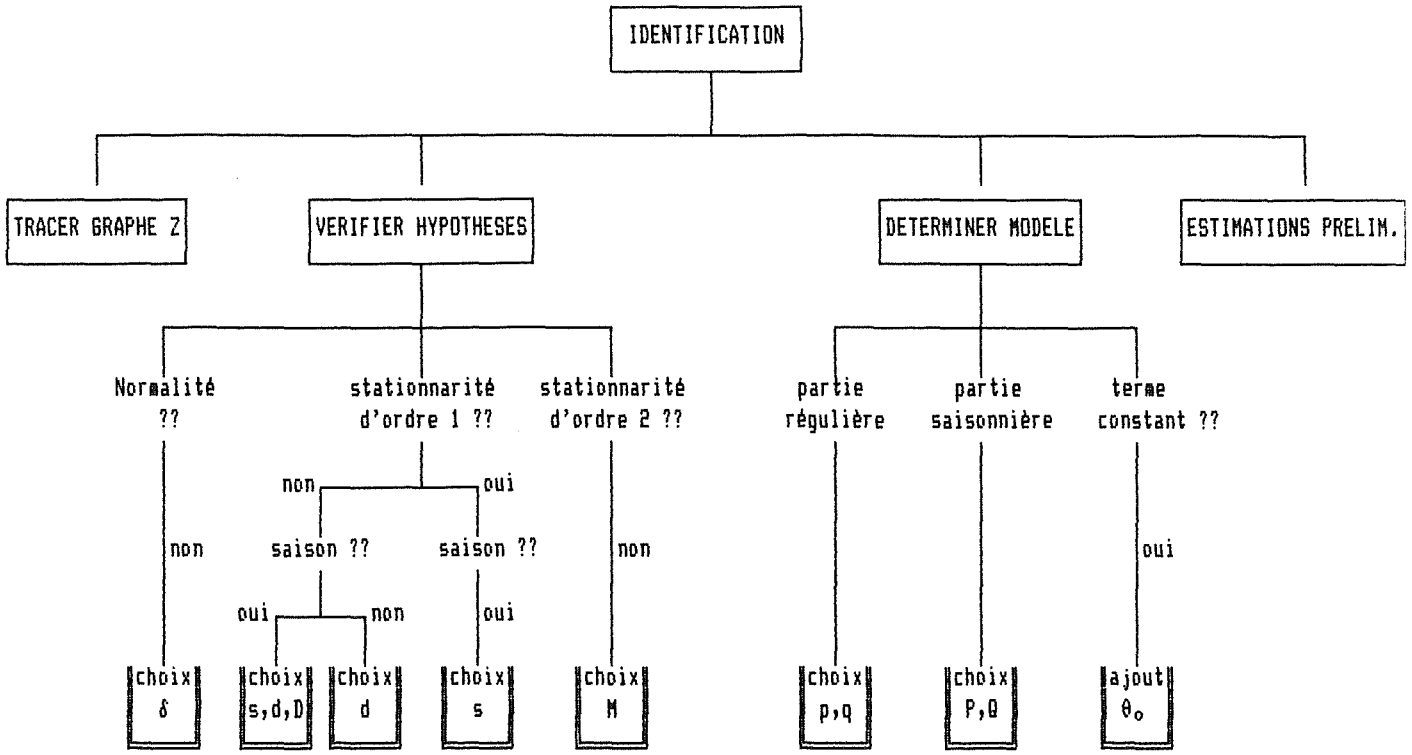
I. LA STRATEGIE

Nous rappelons, sous sa forme graphique, la stratégie proposée dans le rapport en VI.

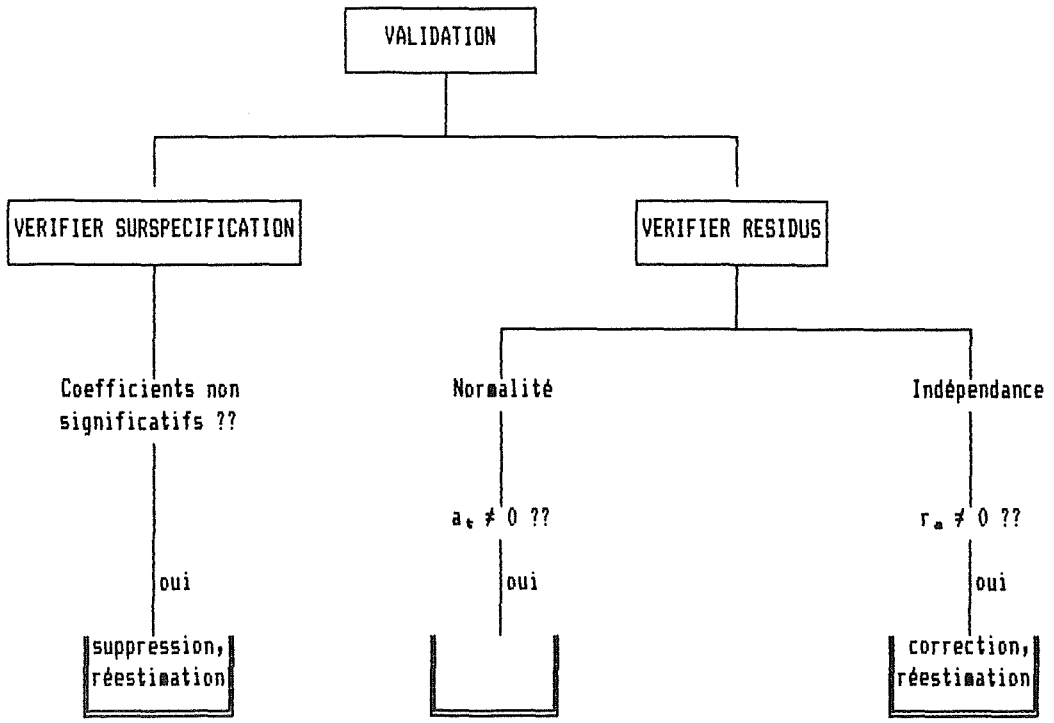
La démarche de construction d'un modèle est la suivante :



La stratégie d'identification propsée est (Figure A2.1) :



La stratégie de validation proposée est (Figure A2.2) :



Nous allons à présent détailler les tâches logiques (noeuds ??) et terminales (noeuds ⊔) de la stratégie d'identification.

II. PARAMETRE DE TRANSFORMATION : δ

1. OBJECTIF

La méthode B&J fait l'hypothèse que la série Z_t est la réalisation d'un processus stochastique de loi Normale. Si ce n'est pas le cas, une transformation non linéaire de Z_t de paramètre δ doit être appliquée. (voir A1.I.3.2.)

2. TEST : normalité ??

2.1. PROCEDURES POSSIBLES

La décision d'effectuer une transformation est basée sur un résultat de BARTLETT : une variance stable en fonction de la moyenne entraîne souvent la normalité.

2.1.1. Méthode graphique

On peut appliquer la méthode du Range-Mean-Plot (voir A1.II.1.1.) pour voir si la variance de k sous-séries dépend de la moyenne.

Inconvénient : Une méthode graphique ne permettrait pas au système de faire une suggestion, elle nécessiterait l'intervention de l'utilisateur. De plus, la méthode ne précise pas la valeur de k.

2.1.2. Test de BARTLETT

BARTLETT a introduit une statistique permettant de tester l'égalité des variances de k sous-séries. Ce test est fréquemment utilisé en économétrie pour détecter l'hétéroscédasticité. En associant à la série i, n_i observations de variance égale à S_i^2, il considère

$$L = \{N.\ln(S^2) - \Sigma\, n_i.\ln(S_i^2)\} / C$$

avec $N = \Sigma\, n_i$, S^2 = variance de la série complète

$$C = 1 + \{1/3.(k-1)\}.\{\Sigma(1/n_i) - 1/N\}$$

Si les observations suivent une loi Normale, et sous l'hypothèse d'égalité des variances, L a approximativement une distribution $X^2_{(k-1)}$.

Inconvénient : Implémenter un test du X^2 n'est pas trivial et, comme en 2.1.1., la méthode ne précise pas les valeurs de k et n_i.

2.2. PROCEDURE RETENUE

La procédure retenue est proche de la méthode graphique, mais elle introduit des heuristiques pour remédier aux inconvénients.

2.2.1. Spécification

PRE : Z[1..N] série observée.
POST : la procédure détermine si, oui ou non, la variance de la série est fonction de la moyenne.

2.2.2. Principe

Initialisation : k := partie entière de √N
lg := partie entière de N/k

Traitement : - Découper le série Z en k sous-séries de longueur lg.
- Calculer moyenne et variance de chaque sous-série.
- Calculer le coefficient de corrélation r entre les moyennes et les variances.

Terminaison : Si $r \neq 0$ alors "suggestion" := effectuer tâche choix δ.
"autre possibilité" := ne pas l'effectuer.
Sinon "suggestion" et "autre possibilité" sont inversées.

2.2.3. Commentaires

- Les valeurs de k et lg ont été choisies de manière à réaliser un compromis entre bonne estimation des moyennes et variances et bonne estimation de r.
- Ne connaissant pas la distribution de r, nous décidons arbitrairement de fixer à [-0.2 : 0.2] l'intervalle en dehors duquel r sera considéré $\neq 0$.
- Cette procédure a l'avantage d'être plus simple que le test de BARTLETT et plus rigoureuse que la méthode graphique.

3. CHOIX DU PARAMETRE δ

3.1. PROCEDURES POSSIBLES

3.1.1. Méthodes rigoureuses

BOX et JENKINS ont proposé une méthode (voir A1.II.1.1.) qui choisit un δ minimisant la somme des carrés des résidus d'un modèle. BOX et COX [1] ont proposé de choisir δ par la méthode du maximum de vraisemblance.

Inconvénient : Ces méthodes consomment beaucoup de temps calcul.

3.1.2. Méthode graphique

La méthode du Range-Mean-Plot peut indiquer un paramètre de transformation (voir A1.II.1.1.).

Inconvénient : Cette méthode manque de précision.

3.2. PROCEDURE RETENUE

La procédure retenue est fort semblable à la procédure de test, mais elle s'applique pour différentes valeurs de δ.

3.2.1. Spécification

PRE : Z[1..N] série observée.
POST : δ : valeur du paramètre de transformation des Z en $Z^{(\delta)}$ qui stabilise la variance en fonction de la moyenne.

3.2.2. Principe

Initialisation : k := partie entière de $\sqrt{N}$
lg := partie entière de N/k

Traitement : Pour différentes valeurs de δ
- Découper le série $Z^{(\delta)}$ en k sous-séries de longueur lg.
- Calculer moyenne et variance de chaque sous-série.
- Calculer le coefficient de corrélation $r(\delta)$ entre les moyennes et les variances.
- Retenir δ qui fournit la valeur minimale de $|r(\delta)|$

Terminaison : "suggestion" := δ.
"autres possibilités" := tous les δ tels que $|r(\delta)| < 0.2$.

3.2.3. Commentaires

- Nous décidons de faire varier δ de -1 à +1 par pas de 0.25. Si la valeur de δ retenue donne un $|r(\delta)|$ grand (>0.2) alors la procédure est réexécutée avec un pas de 0.1. Nous réalisons ainsi un compromis entre précision et efficacité.

REFERENCES CHAPITRE II.

---

[1] BOX G.E.P., COX D.R. : An Analysis of Transformations

III. DEGRE DE DIFFERENCIATION : d

1. OBJECTIF

Une seconde hypothèse doit être vérifiée pour appliquer la méthode B&J : la série Z_t est la réalisation d'un processus stochastique stationnaire. Si ce n'est pas le cas, une transformation en différences de Z_t de paramètre d doit être appliquée. (voir A1. I.3.1.).
Dans un premier temps, on s'intéresse à la stationnarité de la moyenne, et on détermine un d provisoire (ne tenant pas compte d'une éventuelle périodicité).

2. TEST : stationnarité 1 ??

2.1. PROCEDURES POSSIBLES

La décision de différencier la série est basée sur la détection d'une tendance.

2.1.1. Tests de tendance

Plusieurs tests permettent de détecter la présence d'une tendance. SPEARMAN propose de calculer un coefficient de corrélation entre la série t et une série dérivée de Z_t. KENDALL suggère un test similaire, il est décrit ci-dessous. Dans le même ordre d'idée, il est possible de découper la série en k sous-séries et de tester l'égalité des moyennes.

2.1.2. Décroissance de l'ACF

L'ACF d'un processus non stationnaire décroît lentement vers 0. Il est possible de tester cette décroissance soit visuellement, soit en comparant les premiers r_k à une limite.

Inconvénient : - Une méthode graphique nécessiterait l'intervention de l'utilisateur et/ou
- comme le montre LIBERT, la décroissance des premiers coefficients de l'ACF peut révéler aussi bien des processus non stationnaires que stationnaires.

2.2. PROCEDURE RETENUE

La procédure retenue est un test de tendance. Nous avons choisi celui de KENDALL (test de MANN) car il est simple et efficace. Le test est le suivant : Pour toute valeur d'une série Y_t, soit n_t, le nombre de données Y_s qui la précède ($s<t$) telles que $y_s<Y_t$. La quantité $T = \Sigma n_t$ est, sous l'hypothèse (Ho) d'absence de tendance, distribuée asymptotiquement comme une loi normale $N(m,\sigma)$ avec

$$m = n(n-1)/4$$
$$\sigma^2 = n(n-1)(2n+5)/72$$

2.2.1. Spécification

PRE : Y[1..N] série observée normalisée ($Z^{(\delta)}$)
POST : la procédure détecte la présence ou non d'une tendance dans la série Y.

2.2.2. Principe

Initialisation : /

Traitement : - Calculer n_t, T
- Calculer m, σ
- Calculer u(T) = (T-m)/σ

Terminaison : Si $|u(T)| > Q(N(0,1),\alpha)$
alors "suggestion" := effectuer tâche choix δ.
"autre possibilité" := ne pas l'effectuer.
Sinon "suggestion" et "autre possibilité" sont inversées.

2.2.3. Commentaires

- Nous fixons par défaut α à 0.05 et donc $Q(N(0,1),\alpha)$ à 1.96.

3. CHOIX DU PARAMETRE d

3.1. PROCEDURES POSSIBLES

3.1.1. Décroissance de l'ACF

BOX et JENKINS recommandent de choisir le plus petit d tel que l'ACF de $\nabla^d Z^{(\delta)}$ décroisse rapidement vers 0.

Inconvénient : Outre ceux cités en 2.1.2., cette méthode n'est pas très précise.

3.1.2. Variance minimale

ANDERSON préconise de choisir comme valeurs de d et D celles qui fournissent la variance de $\nabla^d\nabla^D Z^{(\delta)}$ minimale. On pourrait se contenter pour l'instant de minimiser la variance de $\nabla^d Z^{(\delta)}$.

Inconvénient : LIBERT montre que l'application de cette méthode peut conduire à retenir un d trop grand.

3.2. PROCEDURE RETENUE

La procédure retenue utilise le critère d'ANDERSON mais vérifie, par un test de tendance, que le d retenu n'est pas trop grand.

3.2.1. Spécification

PRE : Y[1..N] série observée transformée ($Z^{(\delta)}$)
POST : d : valeur du paramètre de transformation des Y en $\nabla^d Y$ qui assure la stationnarité de la moyenne.

3.2.2. Principe

Initialisation : /

Traitement : Pour d := 0,1,2
- Calculer v(d) : variance de $\nabla^d Y$
- Retenir d tel que v(d) est minimum.

Tant que d>0
- Appliquer le test de tendance de MANN sur $\nabla^{d-1} Y$
- si tendance alors terminer sinon d := d-1

Terminaison : "suggestion" := la valeur de d retenue.
"autres possibilités" := les deux valeurs écartées.

IV. PERIODICITE : s

1. OBJECTIF

Si une série est périodique, la méthode B&J peut encore s'appliquer en utilisant un modèle multiplicatif $(p,d,q)x(P,D,Q)_s$ (voir A1.III.1). Cela suppose de pouvoir déterminer le paramètre saisonnier s.

2. TEST : saisonnalité ??

Nous ne connaissons pas de moyen de répondre à la question "Telle série est-elle oui ou non périodique ?" sans répondre également à "et quelle est la période ?". Dès lors, pour cette étape, nous ne séparerons pas la tâche logique et terminale : nous chercherons d'office le paramètre s et si $s \neq 0$, nous décidons que le phénomène est périodique.

3. CHOIX DU PARAMETRE s

3.1. PROCEDURES POSSIBLES

3.1.1. Connaissance à priori

BOX et JENKINS considèrent que la période est connue de l'utilisateur. Eventuellement, cette connaissance peut être confirmée par les coefficients de l'ACF s'ils sont significatifs aux lags ks (k=1,2,...).

Inconvénient : Les rôles sont inversés par rapport à la démarche souhaitée : ici, c'est l'utilisateur qui suggère et le système qui confirme.

3.1.2. Tests de détection

LIBERT présente deux tests pour détecter la présence d'une période. Le premier est une analyse de la variance appliquée, pour diverses valeurs de s, au tableau $Z(i,j) = Z_{i+\langle j-1\rangle_s}$ ($i \leq s$; $j \leq N/s$) et teste un "effet colonne". On retient comme valeur de s, si elle existe, celle qui conduit à une influence significative de l'effet colonne.

Le second ajuste à la série, pour diverses valeurs de s, un modèle purement périodique et teste la valeur du modèle. Parmi les modèles retenus, on choisit s qui conduit à la plus faible variance des résidus.

Inconvénient : Le premier test ne semble pas très efficace, le second s'applique aux séries purement périodiques.

3.2. PROCEDURE RETENUE

La procédure retenue est une heuristique proche de la démarche suivie par un utilisateur qui interprète une ACF. Elle est justifiée par LIBERT.

3.2.1. Spécification

PRE : W[1..N] série observée transformée ($\nabla^d Z^{(s)}$)
r[1..K] ACF de W
POST : s : valeur de la période d'un phénomène saisonnier présent dans la série W. s=0 si un tel phénomène n'est pas présent.

3.2.2. Principe

Initialisation : /

Traitement : - Chercher les r_k significatifs et
Abandonner ceux expliqués par la partie régulière.
$A = \{k: 2<k<N/4 ; |r_k| > 3/\sqrt{N}\}$
$A = A \setminus \{k: |r_k| < \max(|r_{k-1}|, |r_{k+1}|) + 0.5/\sqrt{N}\}$

Si A=∅ s=0 Terminer
Si A={k} s=k Terminer

- Chercher les r_k les plus significatifs
$r_M = \max(|r_k|)$ pour k∈A
$B = A \setminus \{k: |r_k| < |r_M| - 0.5/\sqrt{N}\}$

Si B={M} s=M Terminer

- Chercher les r_{ek} significatifs
$C = B \setminus \{k: |r_{ek}| < 3/\sqrt{N}\}$

Si C={k} s=k Terminer

- Chercher les $\emptyset'_{kk}$ significatifs et
Retenir celui qui fournit le $|r_k|$ maximum

$D = C \setminus \{k: |\emptyset'_{kk}| < 2/\sqrt{N}\}$

Si D=∅ s=0 Terminer
Sinon s=M tel que $r_M = \max(|r_k|)$ pour k∈A

Terminaison : Si s=0 "suggestion" := pas de phénomène saisonnier.
"autres possibilités" := A U B U C
Sinon "suggestion" := phénomène saisonnier de période s.
"autres possibilités" := A U B U C U D

V. DEGRES DE DIFFERENCIATION : d, D

1. OBJECTIF

A ce stade, on sait que la série $Z^{(s)}$ n'est pas stationnaire et qu'elle est périodique de période s. On recherche un d et D définitifs assurant la stationnarité de la moyenne de la série $\nabla^d\nabla^D Z^{(s)}$. Les procédures possibles ont été exposées au chapitre III., nous décrivons uniquement les procédures retenues.

2. TEST : tendance saisonnière ??

Avant tout, on regarde s'il est nécessaire de rechercher D. Cela revient à détecter une tendance saisonnière en appliquant le test de MANN sur les données distantes de s intervalles de temps.

2.1. Spécification

PRE : Y[1..N] série observée normalisée ($Z^{(s)}$)
s : valeur de la période
POST : la procédure détecte la présence ou non d'une tendance saisonnière dans la série Y.

2.2. Principe

Initialisation : /

Traitement : Pour i:=1..s
- Appliquer MANN à la série Y_i, Y_{i+s}, ...
- Calculer n = nombre de rejets de Ho ("Absence de tendance")

Terminaison : Si n < s/2
alors "suggestion" := effectuer choix d,D.
"autre possibilité" := ne pas l'effectuer.
Sinon "suggestion" := ne pas l'effectuer
(d inchangé et D:=0)
"autre possibilité" := effectuer choix d,D.

3. CHOIX DES PARAMETRES d,D

La procédure est comparable à celle destinée à choisir un d provisoire :

3.1. Spécification

PRE : Y[1..N] série observée transformée ($Z^{(\delta)}$)
s : valeur de la période
POST : d,D : valeurs des paramètres de transformation des Y en $\nabla^d\nabla^D Y$ qui assurent la stationnarité de la moyenne.

3.2. Principe

Initialisation : /

Traitement : Pour D := 0,1,2
Pour d := 0,1,2
- Calculer v(d,D) : variance de $\nabla^d\nabla^D Y$
- Retenir d,D tel que v(d,D) est minimum.

Tant que d>0 et pas fin
- Appliquer le test de tendance de MANN sur $\nabla^{d-1}\nabla^D Y$
- si tendance alors fin sinon d := d-1

Tant que D>0 et pas fin
- Appliquer le test de tendance de MANN sur $\nabla^d\nabla^{D-1} Y$
- si tendance alors fin sinon D := D-1

Terminaison : "suggestion" := les valeurs de d,D retenues.
"autres possibilités" := les quatre valeurs écartées.

VI. LONGUEUR DE LA SERIE : M

1. OBJECTIF

Après avoir obtenu la stationnarité de la moyenne, il faut encore obtenir la stationnarité de la variance. Pour cela, on analyse la constance dans le temps de l'ACF. Si cette stationnarité n'est pas vérifiée, nous cherchons la plus grande sous-série, de longueur M en partant de la fin, qui assure la constance dans le temps des coefficients de l'ACF.

2. TEST : stationnarité 2 ??

Nous ne connaissons pas de test. Nous supposerons donc que la série n'est pas stationnaire d'ordre 2 et appliquerons d'office la procédure de choix du paramètre M. Si la série est stationnaire d'ordre 2, cette procédure devrait trouver M=N.

3. CHOIX DU PARAMETRE M

3.1. PROCEDURES POSSIBLES

Bien que le problème ne soit pas abordé explicitement, BOX et JENKINS proposent pour une série particulière de déterminer M arbitrairement. Ce n'est évidemment pas satisfaisant.

LIBERT propose une solution par l'analyse en composantes principales dans laquelle les observations sont des sous-séries de longueurs différentes et les variables des coefficients d'ACF. Les distances entre observations sont interprétées comme des différences entre les ACF. Bien qu'intéressante sur le plan théorique, nous ne retenons pas cette méthode car elle entraîne un important surcroît de calculs.

3.2. PROCEDURE RETENUE

La procédure retenue est basée sur un test de comparaison de deux ACF proposé par LIBERT.

3.2.1. Spécification

PRE : W[1..N] série observée transformée (normale et stationnaire)
POST : M : longueur maximale de la sous-série W[N-M+1..N] respectant l'hypothèse de stationnarité d'ordre 2.

3.2.2. Principe

Initialisation : M := 60
f := 3

Traitement : Calculer R_i : ACF de W[N-M+1..N]
Tant que pas fin
- M := M+f
- Calculer r_i : ACF de W[N-M+1..N]
- Comparer R_i et r_i
  $\alpha := \Sigma(R_i - r_i)^2 / \Sigma R_i^2$
  si $\alpha > 0.2$ M := M-f; fin

Terminaison : "suggestion" := conserver les M dernières observations
"autres possibilités" := $\emptyset$

3.2.3. Commentaires

- Nous initialisons arbitrairement M à 60 et f à 3 car 60 valeurs nous semblent un minimum pour appliquer la méthode B&J et il nous faut réaliser un compromis entre précision et rapidité.
- Nous fixons également arbitrairement à 16 le nombre de coefficients intervenant dans le calcul de α et à 0.2 le seuil de signification de α.

VII. ORDRE DU MODELE : p,q,P,Q

1. OBJECTIF

Les hypothèses de normalité et de stationnarité étant vérifiées, nous pouvons chercher l'ordre d'un modèle ARMA (p,q)x(P,Q)$_{\blacksquare}$ approprié.

2. PROCEDURES POSSIBLES

STREITBERG a introduit pour le système A^{4} [1] une procédure pour déterminer l'ordre d'un modèle. En voici les grandes lignes. Pour différentes valeurs de p et q, on construit une matrice dont les éléments sont des coefficients de corrélation entre deux vecteurs extraits de la série transformée

$(w_1, \ldots, w_{p+1})$ et $(w_{q+1}, \ldots, w_{q+p+1})$

La position d'un bloc de 0 dans la matrice permet de déterminer l'ordre du modèle.

Bien qu'intéressante, cette méthode n'est pas intuitive et nous n'avons pas de renseignement sur son efficacité. C'est pourquoi, nous lui préférerons une démarche plus proche de celle de l'utilisateur.

3. PROCEDURE RETENUE

La procédure retenue est celle proposée par LIBERT. Le principe est de tester successivement les modèles bruit blanc, puis MA(1) et AR(1), puis MA(2), AR(2) et ARMA(1,1) et de retenir le premier modèle qui convient. Avant de spécifier la procédure principale, nous devons spécifier deux sous-programmes : l'un pour déterminer si un modèle est CANDIDAT et l'autre pour CHOISIR un modèle parmi plusieurs modèles candidats.

3.1. PROCEDURE CANDIDAT

3.1.1. Spécification

PRE : W[1..N] série observée transformée (normale et stationnaire)
p,q : entier tels que $0 \leq p+q \leq 2$

POST : La procédure détermine si oui ou non un modèle ARMA(p,q) est adapté à la série W.

3.1.2. Principe

Initialisation : /

Traitement : - Vérifier les conditions de stationnarité et d'inversibilité. Cela se fait en comparant certains r_{kk} à une valeur.
- Tester le caractère significatif de certains coefficients d'autocorrélation.

Terminaison : Répondre "oui" si les deux tests précédents sont acceptés et "non" dans le cas contraire.

3.1.3. Commentaires

- Le détail des coefficients à consulter et leur seuil de signification se trouve dans LIBERT [2].

3.2. PROCEDURE CHOIX

3.2.1. Spécification

PRE : W[1..N] série observée transformée (normale et stationnaire)
{(p,q)} un ensemble de modèles testés et retenus comme candidats.
POST : (p,q) : un modèle parmi l'ensemble des modèles candidats, le plus approprié à la série W.

3.2.2. Principe

Nous choisissons le modèle pour lequel le coefficient d'ACF ou de PACF d'ordre immédiatement supérieur à p+q est le plus proche de la valeur théorique.

3.3. PROCEDURE PRINCIPALE

3.3.1. Spécification

PRE : W[1..N] série observée transformée (normale et stationnaire)

POST : p,q : entier tels que $0 \leq p+q \leq 2$. La procédure détermine plusieurs modèles appropriés à la série W et suggère l'un d'entre eux.

3.3.2. Principe

Initialisation : /

Traitement : Pour i := 0,1,2
- Appliquer CANDIDATS à tous les ARMA(p,q) tels que p+q≤i
- Si plusieurs candidats
  Appliquer CHOIX et retenir (p,q)
  terminer
  Si 1 candidat
  retenir (p,q)
  terminer

Terminaison : "suggestion" := (p,q) retenu s'il existe
"Autres possibilités" := la liste des candidats.

3.1.3. Commentaires

- Pour déterminer l'ordre de la partie saisonnière, nous utilisons la même méthode mais appliquée aux coefficients saisonniers des ACF et PACF.

REFERENCES CHAPITRE VII.

---

[1] STREITBERG B. : Vector Correlations of Time Series

[2] LIBERT B. : Analyse de Séries Chronologiques : Elaboration Automatique et Continue de Prévisions, p 168, 169, 171

ANNEXE A.3.

DOSSIER CONCEPTUEL DU S.E.B.J.

TABLE DES MATIERES

I N T R O D U C T I O N

Cette annexe est un dossier conceptuel détaillé du S.E.B.J. Après avoir rappelé l'architecture logicielle du système, nous détaillerons chaque module de la hiérarchie.

Pour chaque module nous donnons lorsque cela se justifie :
- une brève description du module,
- les constantes symboliques utilisées dans le programme,
- les structures de données,
- la liste des primitives externes et internes,
- les spécifications des primitives externes.

Le niveau de détail de ce dossier se situe donc entre des spécifications externes et des spécifications internes.

ARCHITECTURE LOGICIELLE

La structure conceptuelle du S.E.B.J. est faite de trois composants :
- un noyau organisé en trois niveaux :
  - . Niveau Manager
  - . Niveau Strategy
  - . Niveau Memory
- une interface utilisateur organisée en cinq niveaux :
  - . Niveau Dialog Control
  - . Niveau Structured I/O
  - . Niveau Elementary I/O
  - . Niveau Devices Control
  - . Niveau Utility
- une interface système

STRUCTURE CONCEPTUELLE

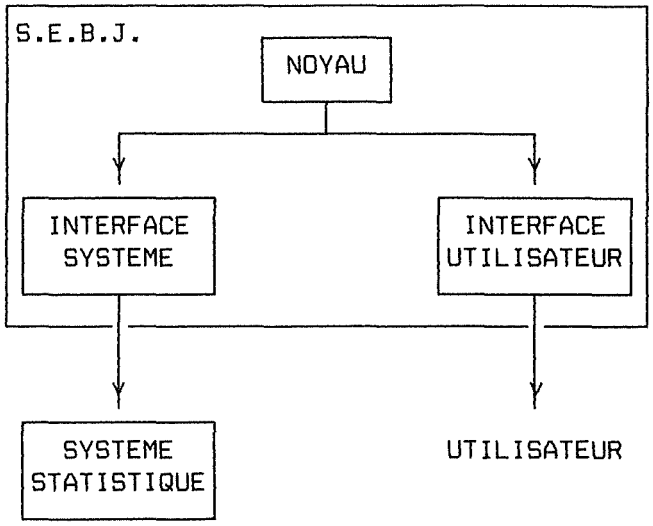


Figure A3.1. : Structure conceptuelle du S.E.B.J.

ARCHITECTURE DU NOYAU

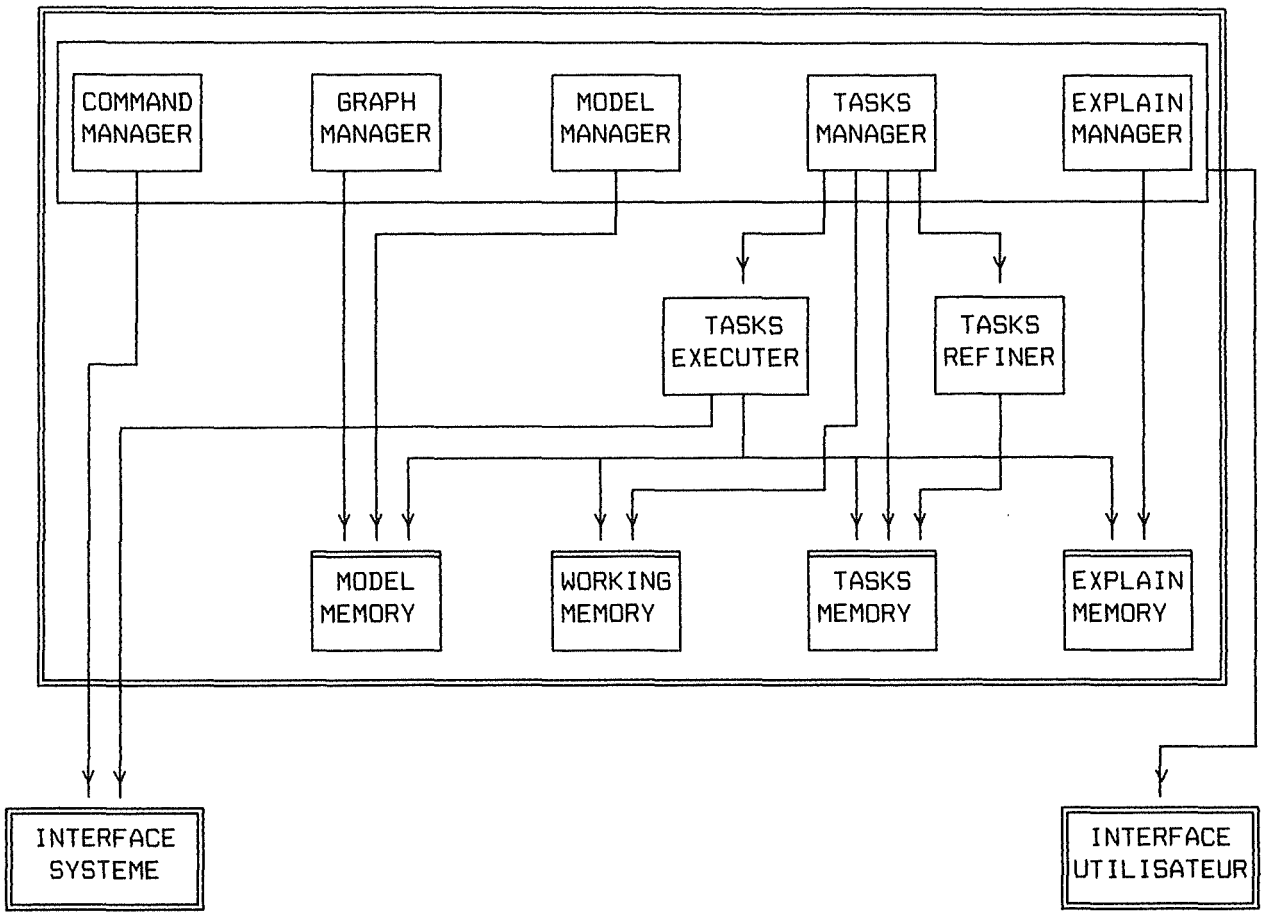


Figure A3.2. : Découpe modulaire du noyau du S.E.B.J.

ARCHITECTURE DE L'INTERFACE UTILISATEUR

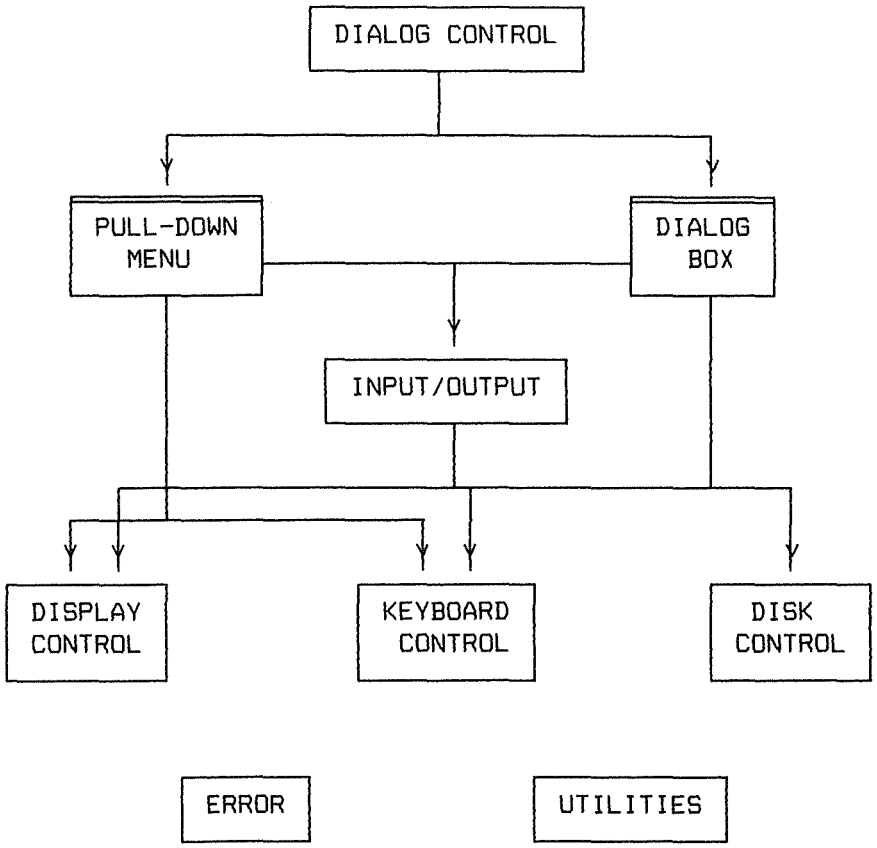


Figure A3.3. : Découpe modulaire de l'Interface Utilisateur

L'INTERFACE SYSTEME

L'interface n'a pas été découpée en modules. En effet, nous avons simulé la présence d'un système statistique. Tous les besoins en calcul numérique (calcul de variance, tests statistiques,...) ont été implémentés par des procédures regroupées dans un module appelé SYSTEME.

I. NIVEAU MANAGER

Ce niveau est composé de gestionnaires de haut niveau qui réalisent les fonctions visibles par l'utilisateur par l'intermédiaire des menus de commandes.

1. FILE MANAGER

1.1. DESCRIPTION

Gérant des fichiers de données.
Réalisation des sous titres du titre FILE du menu principal, propres au noyau.
C'est à dire : - Load
- Save

1.2. STRUCTURES DE DONNEES

Un fichier de données est un flux de caractères, il est composé de 2 parties :
- les caractères suivant @ID permettent d'identifier une série à modéliser,
- les caractères suivant @DATA représentent les valeurs de la série

```
@ID
name            /* Nom de la série,  de longueur < NAMELEN                    */
type            /* Type des données (INT ou FLOAT)                            */
length          /* Nombre de valeurs dans la partie @DATA (long de la série) */
@DATA
:
value           /* Valeurs de la série dans l'ordre chronologique             */
:
```

1.3. LISTE DES PRIMITIVES

1.3.1. EXTERNES

```
void  file_load(void);
```

1.3.2. INTERNES

```
char  *get_filename(void);
int   fget_ID(FILE *f);
void  fget_DATA(FILE *f,int id_m);
int   fget_lg(FILE *f);
int   fget_type(FILE *f);
float fget_value(FILE *f);
char  *fget_string(FILE *f);
```

1.4. SPECIFICATION DES PRIMITIVES

```
void file_load(void);
```

PRE : . /

POST: . Demande à l'utilisateur un nom de fichier de données,
. Utilise Model Memory pour créer un modèle et une série Z pour stocker les données du fichier,
. Utilise Tasks Memory pour créer une pile de tâches associée au modèle créé, et place la tâche "ANALYSIS".

2. MODEL MANAGER

2.1. DESCRIPTION

Gérant des modèles.
Réalisation des sous titres du titre MODEL du menu principal.
C'est à dire : - Identif.
- Estimat.
- Validity
- Forecast

2.2. LISTE DES PRIMITIVES

EXTERNES

```
void model_identif(void)
```

2.3. SPECIFICATION DES PRIMITIVES

```
void model_identif(void);
```

PRE : .

POST: . Si ! id_m : exist_model(id_m) et current_model(id_m)
Alors affichage de l'identité du modèle (Nom et longueur de la série) et des paramètres déjà identifiés, avec possibilité de les changer,
Sinon message d'erreur.

3. TASKS MANAGER

3.1. DESCRIPTION

Gérant des tâches statistiques à accomplir.
Réalisation des sous titres du titre TASKS du menu principal.
C'est à dire : - Show
- Execute
- Remove
- Details

3.2. LISTE DES PRIMITIVES

EXTERNES

```
void tasks_show(void);
void tasks_execute(void);
void tasks_remove(void);
```

3.3. SPECIFICATION DES PRIMITIVES

```
void tasks_show(void);
```

PRE : . /

POST: . Si ! id_ts : exist_tstack(id_ts) et current_tstack(id_ts)
Alors affichage de toutes les tâches restant à exécuter dans la pile courante,
Sinon message d'erreur.

```
void tasks_execute(void);
```

PRE : . /

POST: . Si ! id_ts : exist_tstack(id_ts) et current_tstack(id_ts)
et not_empty(id_ts)
Alors la tâche en haut de la pile id_ts est exécutée en utilisant
. task_refine si la toptask est décomposable,
. task_exe sinon
Sinon message d'erreur.

```
void tasks_remove(void);
```

PRE : . /

POST: . Si ! id_ts : exist_tstack(id_ts) et current_tstack(id_ts)
et not_empty(id_ts)
Alors la tâche en haut de la pile id_ts est enlevée
Sinon message d'erreur

4. EXPLAIN MANAGER

4.1. DESCRIPTION

Gérant des explications à donner à l'utilisateur.
Réalisation des sous titres du titre EXPLAIN du menu principal.
C'est à dire : - Def (What)
- How
- Why

4.2. CONSTANTES SYMBOLIQUES

4.3. STRUCTURES DE DONNEES

4.4. LISTE DES PRIMITIVES

4.4.1. EXTERNES

4.4.2. INTERNES

4.5. SPECIFICATION DES PRIMITIVES

5. GRAPH MANAGER

DESCRIPTION

Gérant de l'affichage des données.
Réalisation des sous titres du titre GRAPH du menu principal.
C'est à dire : - Series
- Resid
- Acf
- Pacf
- Options

Ce module n'a pas encore été implémenté.

6. COMMAND MANAGER

DESCRIPTION

Gérant des commandes à envoyer au système statistique.
Réalisation des sous_titres du titre COMMAND menu principal
C'est à dire : - Execute

Ce module n'a pas encore été implémenté.

7. HELP MANAGER

7.1. DESCRIPTION

Module de gestion d'un help online.

7.2. STRUCTURES DE DONNEES

7.2.1. PRINCIPE

Un fichier d'aide est un fichier texte structuré en "thèmes", chaque "thème" contient des "pages". Concrètement les thèmes correspondent aux choix proposés à l'utilisateur dans le titre HELP du menu; les pages sont les blocs d'information présentés à l'utilisateur en une fois. Pour indiquer cette structure, des commandes de contrôle apparaissent dans le fichier texte. Les commandes doivent apparaître seules sur une ligne :

@THEME : Indique le début d'un nouveau thème.
@PAGE : Indique le début d'une nouvelle page.

Un programme (helpndx.exe) lit ce fichier texte et produit un fichier d'index contenant un identifiant de page et un pointeur vers cette page.

Une procédure lit ce fichier d'index et initialise une table d'index permettant d'accéder directement à chaque page du fichier d'aide, à la suivante et à la précédente.

7.2.2. TRADUCTION C

```
struct PO1{                              /* STRUCTURE DE LA TABLE D'INDEX   */
        int  no_page;                    /* Numéro d'une page du fichier    */
       long  offset;                     /* Distance (en bytes) % au début  */
 struct PO1  *prec;                      /* Pointeur vers la page précédante*/
 struct PO1  *suiv;};                    /* Pointeur vers la page suivante  */

typedef struct PO1 PO;

static PO *indextable[NB_THEME_HELP];

struct POf{                              /* STRUCTURE DU FICHIER D'INDEX      */
        int no_page;                     /* Numéro d'1 page du fichier d'aide */
       long offset;};                    /* Distance (en bytes) % au début    */
```

7.3. LISTE DES PRIMITIVES

7.3.1 EXTERNES

```
void init_help(void);
void get_help(int choix);
void helpdirect(void);
```

7.3.2 INTERNES

```
void init_helpmenu(void);
void init_helpindex(void);
void init_helpbuffer(char *scrbuf);
char *openhelpwindow(void);
char *openhelpbuffer(void);
PO   *chercherpo(int choix, int page);
void fillhelpwindow(PO *curpo, char *scrbuf, FILE *f);
void movefileptr(PO *curpo, FILE *f);
```

7.4. SPECIFICATION DES PRIMITIVES

```
void init_help(void);

   PRE : . Table d'index
             - HELP.NDX :  vérifie les POST de l'utilitaire HELPNDX.EXE
             - En particulier : NB_THEME_HELP records de HELP.NDX vérifient
                                no_page == 0.

   POST: . indextable[NB_THEME_HELP] initialisé
             Tout i est relié à 1 élément d'une chaîne AV/AR de PO

void get_help(int choix);

   PRE : . indextable[] initialisé : <- HELP.NDX <- HELP.TXT
         . 0 <= choix < NB_THEME_HELP

   POST: . Pour le thème choisi :
             - Affichage de l'écran d'aide correspondant à (thème,page 0)
             - Attend instruction de l'utilisateur
               PGDN,PGUP pour "voyager" dans HELP.TXT
               ESC pour sortir

void helpdirect(void);

   PRE : .

   POST: .
```

II. NIVEAU STRATEGY

On trouve à ce niveau des modules de traitement ayant des connaissances sur la stratégie statistique.

1. TASKS EXECUTER

1.1. DESCRIPTION

Module de traitement capable d'exécuter une tâche de type logique ou terminale. L'exécution d'une tâche logique ou terminale donne lieu à une suggestion. Pour une tâche logique la suggestion prend la forme :

Suggestion : Exécuter ou non une tâche terminale.

Pour une tâche terminale, la suggestion prend la forme :

Best : La "meilleure" valeur pour un paramètre,
Others: Une liste de "bonnes" valeurs pour ce paramètre.

1.2. LISTE DES PRIMITIVES

EXTERNES

```
void execute_task(int id_ts);
```

1.3. SPECIFICATION DES PRIMITIVES

```
void execute_task(int id_ts);
```

PRE : . id_ts : exist_tstack(id_ts) et not_empty(id_ts) et
get_tasktype(get_toptask(id_ts)) = LOGIC ou TERMIN

POST: . Exécute la tâche au sommet de la pile id_ts.

2. TASKS REFINER

2.1. DESCRIPTION

Module de traitement capable d'exécuter une tâche de type décomposable. L'exécution d'une tâche décomposable a pour effet de remplacer cette tâche par d'autres plus simples.

2.2. LISTE DES PRIMITIVES

EXTERNES

```
void refine_task(int id_ts);
```

2.3. SPECIFICATION DES PRIMITIVES

```
void refine_task(int id_ts);
```

PRE : . id_ts : exist_tstack(id_ts) et not_empty(id_ts) et
get_tasktype(get_toptask(id_ts)) = DECOMP

POST: . Remplace la tâche au sommet de la pile id_ts par l'ensemble de ses fils.

III. N I V E A U M E M O R Y

Les modules de ce niveau sont des structures de données permettant d'accéder et de sauvegarder des informations.

1. MODEL MEMORY

1.1. DESCRIPTION

Module structure de données permettant d'accéder et de sauvegarder des informations concernant les modèles en cours d'élaboration.

1.2. CONSTANTES SYMBOLIQUES

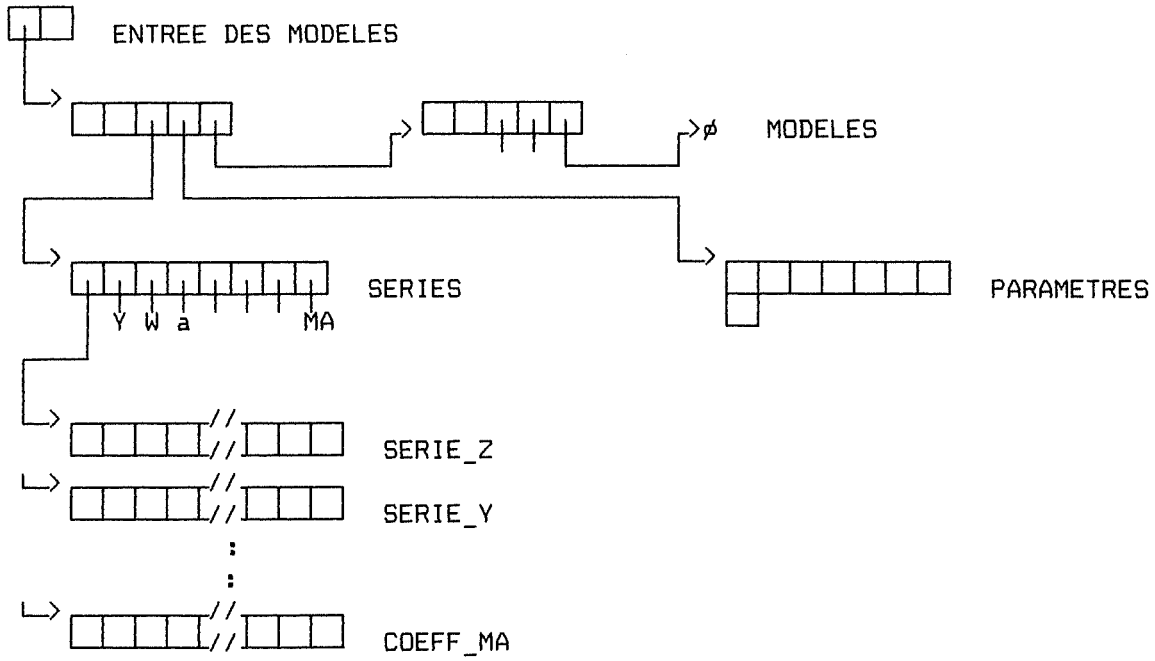
Identifiants des séries	Identifiants des coefficients	Identifiants des paramètres δ,s (p,d,q)x(P,D,Q)
SERIE_Z	COEFF_AR	PARAM_s
SERIE_Y	COEFF_MA	PARAM_l
SERIE_W		PARAM_p
SERIE_A		PARAM_d
SERIE_ACF		PARAM_q
SERIE_PACF		PARAM_P
		PARAM_D
		PARAM_Q

1.3. STRUCTURE DE DONNEES

1.3.1. PRINCIPE

La mémoire peut contenir plusieurs modèles formant une chaîne et identifiés par un numéro. Chaque modèle est caractérisé par 8 séries de valeurs et 8 paramètres.

1.3.2. SCHEMA DE LA STRUCTURE



1.3.3. TRADUCTION C

```
typedef struct{                          /* STRUCTURE SERIE                    */
            int type;                    /* Type de la série: INT ou FLOAT     */
            int lg;                      /* Longueur N de la série             */
          float *val;} serie;            /* Pointeur vers la première valeur   */

typedef struct{                          /* STRUCTURE PARAMETRE                */
          float l;                       /* Valeur du paramètre δ              */
            int p[NBPARAM-1];} param;    /* Valeur des autres paramètres       */

struct ID{                               /* STRUCTURE MODELE                   */
             int id_m;                   /* Identifiant du modèle              */
            char name[NAMELEN];          /* Nom associé au modèle              */
           serie *serie;                 /* Pointeur vers la série observée    */
           param *param;                 /* Pointeur vers les param identif    */
        struct ID *m_s;};                /* Pointeur vers le modèle suivant    */
typedef struct ID model;

typedef struct{                          /* STRUCTURE D'ENTREE DES MODELES     */
             int nb_m;                   /* Nombre de modèles en cours         */
           model *m1;}model1;            /* Pointeur vers le premier modèle    */
```

1.4. LISTE DES PRIMITIVES

1.4.1. EXTERNES

```
void  init_modmem(void);
int   create_model(char *name);
void  set_default_m(int id_m);
int   get_default_m(void);
void  create_serie(int id_m, int id_s, int type, int lg);
float *get_serie(int id_m, int id_s);
int   get_serielg(int id_m, int id_s);
int   get_param(int id_m, int id_p);
float get_param_l(int id_m);
void  put_serie(int id_m, int id_s, int t, float val);
void  put_param(int id_m, int id_p, int val);
void  put_param_l(int id_m, float val);
```

1.4.2. INTERNES

```
model *search_model(int id_m);
```

1.5. SPECIFICATION DES PRIMITIVES

```
void init_modmem(void);
```

PRE : . /

POST: . Initialise le module

```
int create_model(char *name);
```

PRE : . name : Chaîne de char telle que len(name)≤NAMELEN

POST: . id_m : Identifiant d'un nouveau modèle créé et initialisé. Si l'opération a réussi id_m>0 sinon id_m=0

```
char *get_modelname(int id_m)
```

PRE : . id_m : exist_model(id_m)

POST: . name : Nom du modèle identifié par id_m

```
void set_default_m(int id_m);
```

PRE : . id_m : exist_model(id_m)

POST: . Le modèle identifié par id_m est sélectionné par défaut

```
int get_default_m(void);
```

PRE : . /

POST: . id_m : Identifiant du modèle sélectionné par défaut

```
void create_serie(int id_m, int id_s, int type, int lg);

   PRE : . id_m : exist_model(id_m)
         . id_s : exist_serie(id_m,id_s) et get_serielg(id_s)>0

   POST: . Crée une structure capable de recevoir une série de type type, de
              longueur lg, l'attache au modèle id_m en tant que série id_s


float *get_serie(int id_m, int id_s);

   PRE : . id_m : exist_model(id_m)
         . id_s : exist_serie(id_m,id_s)

   POST: . renvoie un tableau contenant la série id_s du modèle id_m


int get_serielg(int id_m, int id_s);

   PRE : . id_m : exist_model(id_m)
         . id_s : exist_serie(id_m,id_s)

   POST: . renvoie la longueur de la série id_s du modèle id_m


int get_param(int id_m, int id_p);

   PRE : . id_m : exist_model(id_m)
         . id_p : exist_param(id_m,id_p)

   POST: . renvoie la valeur du paramètre id_p du modèle id_m


float get_param_1(int id_m);

   PRE : . id_m : exist_model(id_m)

   POST: . renvoie la valeur du paramètre PARAM_1 du modèle id_m


void put_serie(int id_m, int id_s, int t, float val);

   PRE : . id_m : exist_model(id_m)
         . id_s : exist_serie(id_m,id_s)
         . t    : 0 ≤ t ≤ get_serielg(id_m,id_s)
         . val  : Réel qcq

   POST: . get_serie(id_m, id_s, t) = val


void put_param(int id_m, int id_p, int val);

   PRE : . id_m : exist_model(id_m)
         . id_p : exist_param(id_m,id_p) et (id_p != PARAM_1)
         . val  : Entier qcq

   POST: . get_param(id_m, id_p) = val
```

```
void put_param_l(int id_m, float val);
```

PRE : . id_m : exist_model(id_m)
. val : réel quelconque

POST: . get_param_l(id_m, id_p) = val

2. WORKING MEMORY

2.1. DESCRIPTION

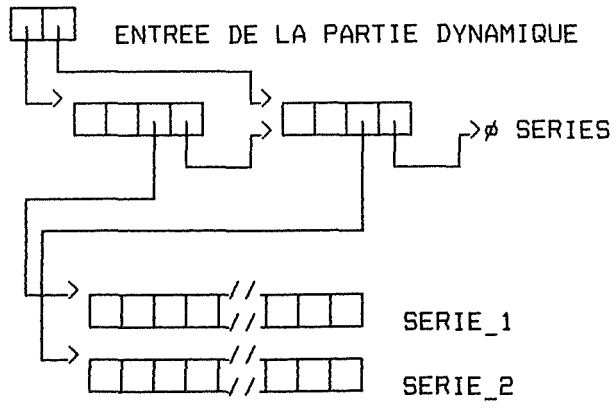
Module structure de données permettant d'accéder et de sauvegarder des informations concernant les détails de l'exécution de la dernière tâche (de type logique ou terminale).

2.2. STRUCTURE DE DONNEES

2.2.1. PRINCIPE

La mémoire est constituée de 2 parties, l'une est dynamique et est composée de séries de valeurs qu'il est possible de créer pour y stocker des résultats intermédiaires. Les séries sont organisées selon une chaîne. L'autre est statique et reçoit les résultats significatifs de l'exécution d'une tâche.

2.2.2. SCHEMA DE LA STRUCTURE



2.2.3. TRADUCTION C

```
/* DYNAMIC PART */

struct S{                                   /* STRUCTURE SERIE  (DYNAMIQUE)        */
            int id_s;                       /* Identifiant de la série             */
            int lg;                         /* Longueur de la série                */
          float *val;                       /* Pointeur vers la première valeur    */
          struct S *s_s;};                  /* Série suivante dans la chaîne       */
typedef struct S wm_serie;

typedef struct{                             /* ENTREE DE LA PARTIE DYNAMIQUE       */
           wm_serie *s1;                    /* Pointeur vers la première série     */
           wm_serie *sn;}wm_serin;          /* Pointeur vers la dernière série     */
static wm_serin w;
```

2.3. LISTE DES PRIMITIVES

2.3.1. EXTERNES

```
void  init_workmem(void);
int   create_wmserie(int lg);
float *get_wmserie(int id_s);
int   get_wmserielg(int id_s);
void  put_wmserie(int id_s, int t, float val);
void  del_wmserie(int id_s);
```

2.3.2. INTERNES

```
wm_serie *search_wmserie(int id_s);
```

2.4. SPECIFICATION DES PRIMITIVES

```
void  init_workmem(void);
```

PRE : . /

POST: . Initialise le module

```
int   create_wmserie(int lg);
```

PRE : . lg : lg>0

POST: . id_s : Identifiant d'une structure créée capable de recevoir une série de valeurs de longueur lg

```
float *get_wmserie(int id_s);
```

PRE : . id_s : existe_wmserie(id_s)

POST: . val : renvoie le premier élément du tableau contenant la série id_s

```
int   get_wmserielg(int id_s);
```

PRE : . id_s : existe_wmserie(id_s)

POST: . lg : renvoie la longueur de la série id_s

```
void  put_wmserie(int id_s, int t, float val);
```

PRE : . id_s : existe_wmserie(id_s)
t : 0<= t <lg(id_m,id_s)
val : valeur quelconque (réelle)

POST: . get_wmserie(id_s, t) = val

```
void  del_wmserie(int id_s);
```

PRE : . id_s : exist_wmserie(id_s)

POST: . Libère la zone réservée pour contenir la série de valeurs id_s

3. TASKS MEMORY

3.1. DESCRIPTION

Structure de données permettant d'accéder et de sauvegarder des informations concernant les tâches : celles restant à accomplir pour élaborer les modèles en cours et celles faisant partie de la stratégie.

3.2. CONSTANTES SYMBOLIQUES

Identifiants des tâches

```
ANALYSIS     A_IDENT     A_ESTIM     A_VALID     A_FORECAST     AI_SHOW_Z
AI_CHECK     AI_ORDER    AI_ESTIM    AV_OCHECK   AV_RCHECK      AIC_NORM
AIC_STAT1    AIC_STAT2   AIO_pq      AIO_PQ      AIO_CONST      AVO_SIGN
AVR_NORM     AVR_CORR    AICS_SEAS   AICN_FIX1   AICS_FIXd      AICS_FIXs
AICS_FIXdD   AICS_FIXM
```

Types de tâche

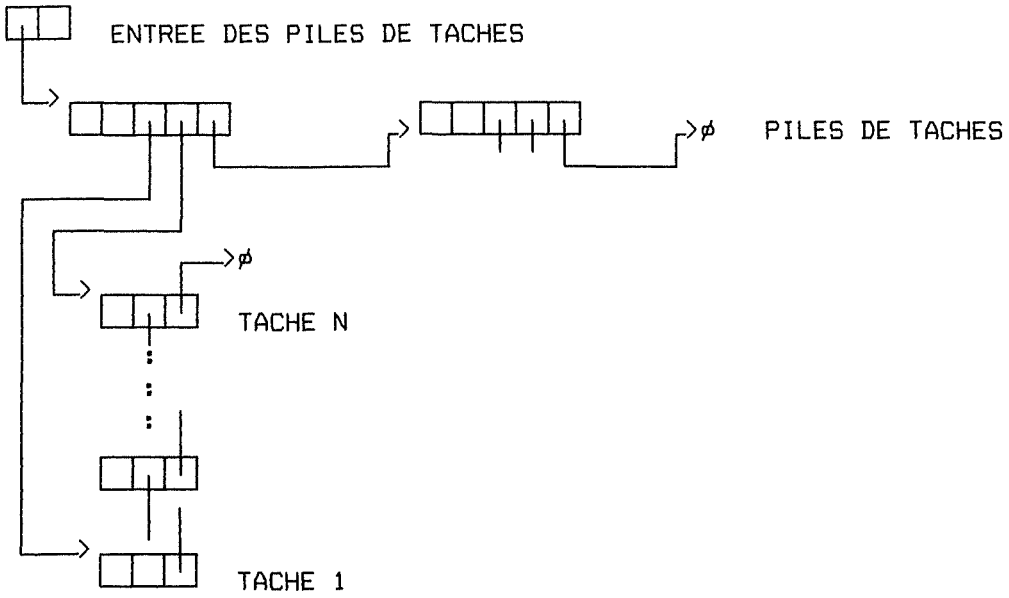
```
DECOMP              /* Tâche de type décomposable      */
LOGIC               /* Tâche de type logique           */
TERMIN              /* Tâche de type terminal          */
```

3.3. STRUCTURE DE DONNEES

3.3.1. PRINCIPE

Cette mémoire contient des piles de tâches. Les piles sont enchaînées les unes aux autres. Les tâches d'une pile sont reliées par un chaînage avant arrière.

3.3.2. SCHEMA DE LA STRUCTURE



3.3.3. TRADUCTION C

```
struct T{                       /* STRUCTURE TACHE                              */
      int id_t;                 /* Identifiant de la tâche dans une pile        */
 struct T *t_p;};               /* Tâche juste en-dessous sur la pile           */
 struct T *t_s;};               /* Tâche juste au-dessus sur la pile            */
typedef struct T task;

struct TS{                      /* STRUCTURE PILE DE TACHES                     */
      int id_ts;                /* Identifiant de la pile de tâches             */
      int nb_t;                 /* Nombre de tâches dans la pile                */
     task *t1;                  /* Pointeur vers la première tâche de la pile   */
     task *tn;                  /* Pointeur vers la dernière tâche de la pile   */
struct TS *ts_s;};              /* Pointeur vers la pile suivante               */
typedef struct TS tstack;

typedef struct{                 /* STRUCTURE D'ENTREE DANS LES PILES            */
      tstack *ts1;              /* Pointeur vers la première pile de tâches     */
        int nb_ts;}tstack1;     /* Nombre de piles de tâches                    */
```

3.4. LISTE DES PRIMITIVES

3.4.1. EXTERNES

```
void init_taskmem(void);
int  create_tstack(int id_m);
int  get_tstackmodel(int id_ts);
void set_default_ts(int id_ts);
int  get_default_ts(void);
int  add_task(int id_ts, int id_t);
void remove_task(int id_ts);
int  get_tstacklevel(int id_ts);
int  get_toptask(int id_ts);
int  get_firsttask(int id_ts);
int  get_nexttask(int id_ts, int id_t);
char *get_taskname(int id_t);
int  get_tasktype(int id_t);
int  get_tasklevel(int id_t);
```

3.4.2. INTERNES

```
tstack *search_tstack(int id_ts);
task   *search_task(int id_ts, int id_t);
```

3.5. SPECIFICATION DES PRIMITIVES

```
void init_taskmem(void);
```

PRE : . /

POST: . Initialise le module

```
int create_tstack(int id_m);
```

PRE : . id_m : exist_model(id_m) et ¬exist_tstack(id_m)

POST: . id_ts : identifiant d'une nouvelle pile de tâches associée au modèle id_m. exist_tstack(id_ts)

```
int get_tstackmodel(int id_ts);
```

PRE : . id_ts : exist_tstack(id_ts)

POST: . id_m : Identifiant du modèle associé à la pile id_ts

```
void set_default_ts(int id_ts);
```

PRE : . id_ts : exist_tstack(id_ts)

POST: . La pile id_ts est sélectionnée par défaut */

```
int get_default_ts(void);

   PRE : . /

   POST: . id_ts : Identifiant de la pile sélectionnée par défaut


int add_task(int id_ts, int id_t);

   PRE : . id_ts : exist_tstack(id_ts)
         . id_t  : Identifiant d'une tâche connue par le système

   POST: . id_t : Identifiant de la nouvelle tâche ajoutée au sommet de la pile
                  id_ts. exist_task(id_ts,id_t)


void remove_task(int id_ts);

   PRE : . id_ts : exist_tstack(id_ts) et not_empty(id_ts)

   POST: . Enlève la tâche au sommet de la pile id_ts.
                 ┐exist_task(id_ts, top_task(id_ts))


int get_tstacklevel(int id_ts);

   PRE : . id_ts : exist_tstack(id_ts)

   POST: . nb_t  : Nombre de tâches dans la pile id_ts


int get_toptask(int id_ts);

   PRE : . id_ts : exist_tstack(id_ts) et get_tstacklevel(id_ts) > 0

   POST: . id_t  : Identifiant de la tâche au sommet de la pile


int get_firsttask(int id_ts);

   PRE : . id_ts : exist_tstack(id_ts) et get_tstacklevel(id_ts) > 0

   POST: . id_t  : Identifiant de la tâche au bas de la pile


int get_nexttask(int id_ts, int id_t);

   PRE : . id_ts : exist_tstack(id_ts)

   POST: . id_t  : Identifiant de la tâche au-dessus de celle identifiée par
                   (id_ts,id_t). 0 si (id_ts,id_t) est la top task


char *get_taskname(int id_t);

   PRE : . id_t  : exist_task(id_t)

   POST: . name : Nom de la tâche identifiée par (id_ts, id_t)
```

```
int get_tasktype(int id_ts,int id_t);
```

PRE : . id_t : exist_task(id_t)

POST: . type : Type (DECOMP,LOGIC,TERMIN) de la tâche identifiée par (id_ts, id_t)

```
int get_tasklevel(int id_ts,int id_t);
```

PRE : . id_t : exist_task(id_t)

POST: . level : Niveau dans l'arbre de la tâche identifiée par (id_ts, id_t)

4. EXPLAIN MEMORY

4.1. DESCRIPTION

4.2. CONSTANTES SYMBOLIQUES

4.3. STRUCTURES DE DONNEES

4.4. LISTE DES PRIMITIVES

4.4.1. EXTERNES

4.4.2. INTERNES

4.5. SPECIFICATION DES PRIMITIVES

IV. N I V E A U D I A L O G C O N T R O L

Niveau assurant le lien entre les fonctions de l'application (le niveau Manager du noyau) et leur présentation à l'utilisateur. Ce lien se fait au moyen d'objets structurés.

1. MAIN

1.1. DESCRIPTION

Coordinateur général de toutes les fonctions du système
En particulier : - F1 donne accès au HELP
- F9 donne accès au menu principal
- Alt + car donne accès au menu identifié par car.

1.2. LISTE DES PRIMITIVES

1.2.1. EXTERNES

```
void m_init(void);
void m_loop(void);
void m_termin(void);
```

1.2.2. INTERNES

```
void mi_menu(void);
void mi_screen(void);
void ml_dofkey(int key);
void ml_doalt(int key);
void ml_docde(int choix);
```

1.3. SPECIFICATION DES PRIMITIVES

```
void m_init(void);
```

PRE : . Routine exécutée avant tout autre.

POST: . Initialise tous les modules et les variables externes.

```
void m_loop(void);
```

PRE : . /

POST: . Boucle principale de prise en charge et d'analyse des événements provenant de l'utilisateur. Les seuls événements actuellement considérés sont les touches clavier.

```
void m_termin(void);
```

PRE : . Routine exécutée après tout autre.

POST: . Effectue les opérations de terminaison pour tous les modules.

V. NIVEAU APPLICATION

Niveau prenant en charge des fonctions visibles par l'utilisateur mais propres à l'interface utilisateur.

1. FILE

1.1. DESCRIPTION

Réalisation des sous titres du titre FILE du menu principal pris en charges par l'interface utilisateur.
C'est à dire : - Dir
- Change dir
- Osshell
- Quit

1.2. LISTE DES PRIMITIVES

1.2.1. EXTERNES

```
void changedir(void);
void osshell(void);
```

1.2.2. INTERNES

```
static int dos_cmd(void);
static int get_comspec(char *buffer);
```

1.3. SPECIFICATION DES PRIMITIVES

```
void changedir(void);
```

PRE : . /

POST: . Affiche le directory courant.
. Permet l'édition de ce directory.
. Si le directory est modifié et valide le directory édité devient courant.

```
void osshell(void);
```

PRE : . /

POST: . Effectue une sortie provisoire de l'application vers le système d'exploitation. Pour revenir à l'application taper la commande EXIT.

VI. NIVEAU STRUCTURED I/O

Niveau comprenant des modules d'entrée/sortie de haut niveau. Les objets actuellement disponibles sont des menus déroulants et des boîtes de dialogue.

1. MENU

1.1. DESCRIPTION

Module de gestion de menus déroulants.

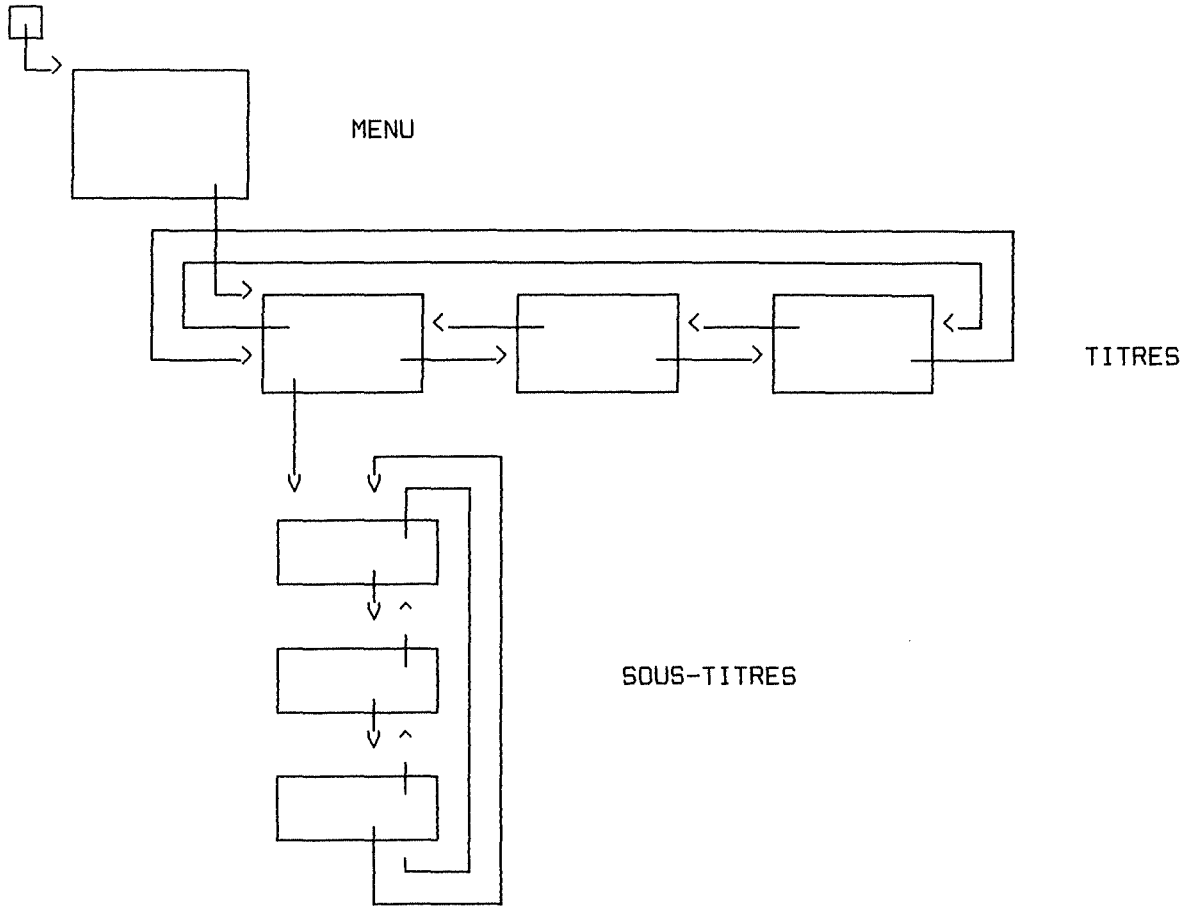
1.2. STRUCTURES DE DONNEES

1.2.1. PRINCIPE

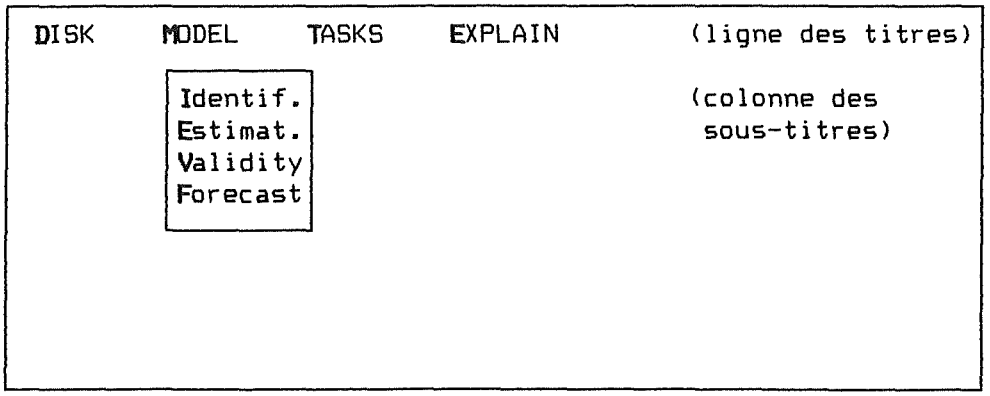
Un menu est une structure de données contenant des titres, contenant eux-mêmes des sous-titres. Titres et sous-titres sont organisés selon des anneaux bidirectionnels.

A l'utilisation les noms des titres sont affichés en permanence à l'écran. Sur demande, l'utilisateur peut dérouler un titre, c'est-à-dire afficher les noms de ses sous-titres sur une colonne sous le titre et sélectionner un sous-titre correspondant à un traitement.

1.2.2. SCHEMA DE LA STRUCTURE



1.2.3. SCHEMA DE LA PRESENTATION



1.2.4. TRADUCTION C

```
struct ST{                          /* STRUCTURE SOUS-TITRE                    */
        char      *nom;             /* Nom du sous-titre                       */
        int       num;              /* Num d'ordre (≥0) du ss-titre dans le titre
        struct ST *st_s;            /* Pointeur vers le ss-titre suivant       */
        struct ST *st_p;            /* Pointeur vers le ss-titre précédent     */
        };
typedef struct ST s_titre;


struct T{                           /* STRUCTURE TITRE                         */
        char     *nom;              /* Nom du titre                            */
        int      num;               /* Num d'ordre (≥0) du titre dans le menu  */
        int      x;                 /* Abscisse du premier car. du nom du titre*/
        int      nb_st;             /* Nombre de sous-titre dans le titre      */
        char     *chx;              /* Chaîne des premièrs car. des ss-titres  */
        int      larg;              /* Largeur du cadre contenant les ss-titres*/
        struct T *t_s;              /* Pointeur vers le titre suivant          */
        struct T *t_p;              /* Pointeur vers le titre précédent        */
        s_titre  *st_1;             /* Pointeur vers le premier sous-titre     */
        };
typedef struct T titre;

struct M{                           /* STRUCTURE MENU                          */
        int   x;                    /* Abscisse du premier car du premier titre*/
        int   y;                    /* Ordonnée de la ligne des titres         */
        int   nb_t;                 /* Nombre de titres dans le menu           */
        char  *chx;                 /* Chaîne des premiers car. des titres     */
        titre *t_1;                 /* Pointeur vers le premier titre          */
        };
typedef struct M menu;
```

1.3. LISTE DES PRIMITIVES

1.3.1. EXTERNES

```
menu    *create_menu(int x, int y);
titre   *add_titre(menu *m, char *nom);
s_titre *add_stitre(titre *t, char *nom);

void    titre_menu(menu *m);
titre   *set_titre(menu *m, int key);
titre   *reset_titre(menu *m);
s_titre *set_s_titre(titre *t, int key);
s_titre *reset_stitre(titre *t);
int     existe_titre(menu *m, int key);

int     get_chxcde(menu *m, titre *t, s_titre *s);
void    maj_ts(titre **tp, s_titre **sp, int choix);
```

1.3.2. INTERNES

```
char    *deroule_titre(int y, titre *t);
void    enroule_titre(char *scrcopy, int y, titre *t);
titre   *maj_titre(titre *t, int key);

void    allume_s_titre(int y, titre *t, s_titre *s);
void    eteind_s_titre(int y, titre *t, s_titre *s);
s_titre *maj_s_titre(titre *t, s_titre *s, int key);
```

1.4. SPECIFICATION DES PRIMITIVES

```
menu *create_menu(int x, int y);
```

PRE : . x,y : coordonnées du premier titre du menu

POST: . m : initialisé : Renvoie un pointeur vers une structure menu créée et initialisée.

```
titre *add_titre(menu *m, char *nom);
```

PRE : . m : initialisé
. nom : chaîne quelconque de caractères

POST: . t : initialisé : Renvoie un pointeur vers une structure titre créée et initialisée.
Cette structure est associée au menu m (dernier titre) et est identifiée par nom.

```
s_titre *add_stitre(titre *t, char *nom);
```

PRE : . t : initialisé
. nom : chaîne quelconque de caractères

POST: . st : initialisé : Renvoie un pointeur vers une structure s_titre créée et initialisée.
Cette structure est associée au titre t (dernier s_titre) et est identifiée par nom.

```
void titre_menu(menu *m);
```

PRE : . m : initialisé.

POST: . Affichage des noms des titres associés à m sur 1 ligne à partir de (m->x,m->y)
L'espace entre 2 titres est fixé par SEPAR

```
titre *set_titre(menu *m, int key);
```

PRE : . m : initialisé
. key : code ASCII d'un char ∈ m->chx

POST: . Renvoie un ptr vers le premier titre de m dont t->nom[0] = key

```
titre *reset_titre(menu *m);
```

PRE : . m : initialisé

POST: . Renvoie un ptr vers le premier titre du menu m

```
s_titre *set_s_titre(titre *t, int key);
```

PRE : . t : initialisé
. key : code ASCII d'un char ∈ t->chx

POST: . Renvoie un ptr vers le premier s_titre de t dont st->nom[0] = key

```
s_titre *reset_stitre(titre *t);
```

PRE : . t : initialisé

POST: . Renvoie un ptr vers le premier s_titre du titre t

```
int existe_titre(menu *m, int key);
```

PRE : . m : initialisé
. key : code ASCII d'un char quelconque

POST: . Renvoie VRAI s'il existe un titre de m identifié par le caractère key , FAUX sinon.

```
int get_chxcde(menu *m, titre *t, s_titre *s);
```

PRE : . m : initialisé
. t : initialisé
. s : initialisé

POST: . Déroule t
. Allume s
. Attend le choix de l'utilisateur
ESC : Abandon sans action
Cursor+CR : Changement de titre ou sous titre actif
Char : Changement de sous titre actif
. Enroule le titre actif
. Renvoie un identifiant du choix (t,s)
-1 : si ESC
s->num : sinon : 8 bits gauche identifient t
8 bits droite identifient s

```
void maj_ts(titre **tp, s_titre **sp, int choix);
```

PRE : . tp : *tp : initialisé
. sp : *sp : initialisé
. choix : identifiant (t,s)

POST: . *tp, *sp mis à jour : en sortie, ils pointent respectivement vers le titre et sous titre identifiés par choix.

2. DIALOG BOX

2.1. DESCRIPTION

Module de gestion des boîtes de dialogue.

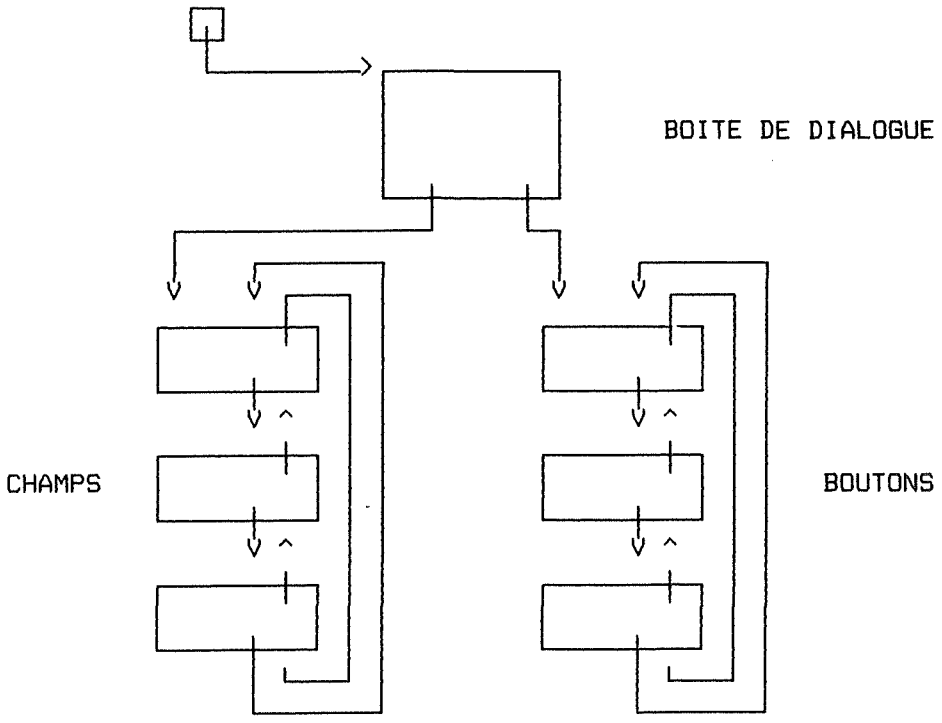
2.2. STRUCTURES DE DONNEES

2.2.1. PRINCIPE

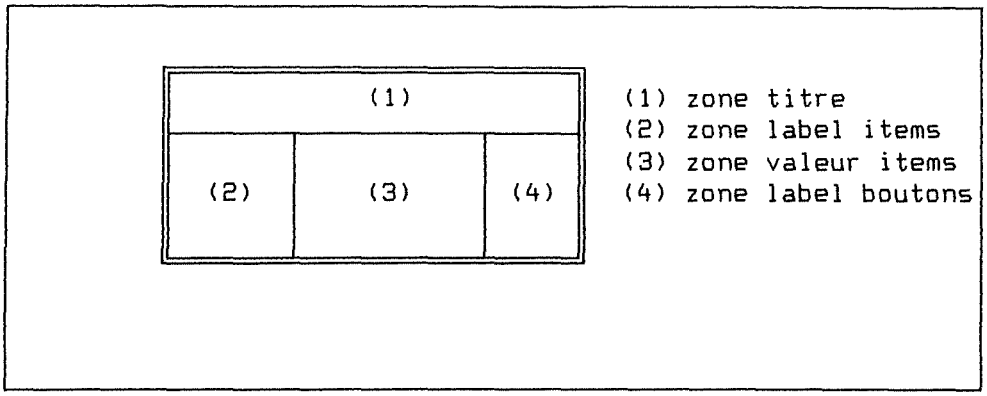
Un boîte de dialogue est une structure de données contenant des champs (items et valeurs d'item) et des boutons. Champs et boutons sont des structures de données organisées selon des anneaux bidirectionnels.

A l'utilisation, la boîte apparaît à l'écran comme un cadre découpé en plusieurs zone : une pour un titre, une pour les champs et une pour les boutons. Sur demande, l'utilisateur peut sélectionner un champ pour éditer sa valeur d'item ou un bouton correspondant à un traitement.

2.2.2. SCHEMA DE LA STRUCTURE



2.2.3. SCHEMA DE LA PRESENTATION



2.2..4. TRADUCTION C

```
struct B{
        char      *label;          /* Label du bouton                          */
        int       num;             /* Num. d'ordre (≥0) du bouton dans la boîte
        struct B  *b_s;            /* Pointeur vers le bouton suivant          */
        struct B  *b_p;            /* Pointeur vers le bouton précédent        */
        };
typedef struct B button;


struct F{
        char      *label;          /* Label du champ                           */
        int       num;             /* Num. d'ordre (≥0) du champ dans la boîte
        int       len;             /* Longueur maximale de l'item              */
        char      item[ITEMLEN];   /* Valeur de l'item                         */
        struct F *f_s;             /* Pointeur vers le champ suivant           */
        struct F *f_p;             /* Pointeur vers le champ précédent         */
        };
typedef struct F field;

struct X{
        int   x1;                  /* Abs. du coin sup. gauche de la boîte     */
        int   y1;                  /* Ord. du coin sup. gauche de la boîte     */
        int   dx1;                 /* Largeur des espaces réservés au labels   */
        int   dx2;                 /* des items, aux valeurs des items et aux  */
        int   dx3;                 /* labels des boutons.                      */
        int   dy;                  /* nb de lignes de la boîte = max(nb_b,nb_f)
        char  titre[TITRELEN];     /* Titre de la boîte                        */
        int   nb_b;                /* Nombre de boutons dans la boîte          */
        int   nb_f;                /* Nombre de champs  dans la boîte          */
       button *b_1;                /* Pointeur vers le premier bouton          */
       field  *f_1;                /* Pointeur vers le premier champ           */
        };
typedef struct X box;
```

2.3. LISTE DES PRIMITIVES

2.3.1. EXTERNES

```
box    *create_box(int x1, int y1, char *titre);
button *add_but(box *x, char *label);
field  *add_field(box *x, char *label, int len, va_list arg_list, ...);

int out_box(box *x);
```

2.3.2. INTERNES

```
char   *display_box(box *x);
void   allume_but(int x, int y, button *b);
void   eteind_but(int x, int y, button *b);
button *maj_but(box *x, button *b, int key);
void   del_box(char *scrcopy, box *x);
```

2.4. SPECIFICATION DES PRIMITIVES

```
box *create_box(int x1, int y1, char *titre);
```

PRE : . x1, y1 : coordonnées du coin supérieur gauche de la boîte
. titre : chaîne de char telle que strlen(titre)<TITRELEN

POST: . x : créé et initialisé. Renvoie un pointeur vers une structure de données de type box initialisée. Le titre de x est titre. Un bouton ESC est ajouté à toute boîte.

```
button *add_but(box *x, char *label);
```

PRE : . x : initialisé
. label : chaîne quelconque de caractères

POST: . b : créé et initialisé. Renvoie un pointeur vers un structure button initialisée. b est associé à x et porte le label label.

```
field  *add_field(box *x, char *label, int len, va_list arg_list, ...)
```

PRE : . x : initialisé
. label, item : chaîne quelconque de caractères
. len : entier <= ITEMLEN
. arg_list : liste de deux arguments : un format typé (control string) et une valeur d'item. Les règles sont celles de la primitive printf() du langage C.

POST: . f : créé et initialisé. Renvoie un pointeur vers une structure field initialisée. f est associé à x, porte le label label, et contient un item dont le type et la valeur sont définis dans arg_list et de longueur maximale len.

```
int out_box(box *x);
```

PRE : . x : initialisé

POST: . Affiche la boîte x et ses composants (field et button)
. Attend les instructions de l'utilisateur (choix d'un bouton)
. Renvoi le numéro de bouton choisi et détruit la boîte. Le bouton ESC a toujours me numéro 0.

VII. NIVEAU ELEMENTARY I/O

Niveau comprenant des modules d'entrée/sortie de bas niveau. Les spécifications externes des primitives de ces modules n'ont pas été reprises dans ce dossier.

1. OUTPUT

1.1. DESCRIPTION

Gestion des sorties écran de bas niveau
Cela comprend : - affichage de cadres,
- gestion des messages textuels,
- gestion de l'indicateur d'état.

1.2. CONSTANTES SYMBOLIQUES

COORDONNEES	MESSAGES	STATUS
X1MSG1	MSGF1F9	ERROR
X2MSG1	MSGWAITCMD	READY
X1MSG2	MSGPRESSCR	WAIT
X2MSG2	MSGPRESSESC	
YMSG		
X1STATUS		
X2STATUS		
YSTATUS		

1.3. STRUCTURES DE DONNEES

```
struct msgstack{
                char *msg;
    struct msgstack *prec;};

typedef struct msgstack MS;
```

1.4. LISTE DES PRIMITIVES

1.4.1. EXTERNES

```
void init_output(void);

void frame(int x1,int y1,int x2,int y2);

void stdmsg1(int msg1);
void stdmsg2(int msg2);
void writemsg1(char *str1);
void writemsg2(char *str2);
void clearmsg1(void);
void clearmsg2(void);
void clearmsg(void);
void savemsg1(void);
void savemsg2(void);
void restoremsg(void);

void writestatus(int code);
void clearstatus(void);
```

1.4.2. INTERNES

```
void initmsgstatus(void);
```

2. INPUT

2.1. DESCRIPTION

Gestion des entrées écran de bas niveau
Cela comprend : - entrée d'une chaîne avec possibilités d'édition

2.2. LISTE DES PRIMITIVES

EXTERNES

```
void initedit(void);
void clearedit(char *string);
int  editstring(char *s,char *legal,int maxlength);
void indicins(int ins);
```

VIII. NIVEAU DEVICES CONTROL

Ce niveau comprend des modules d'interface entre les niveaux supérieurs de la hiérarchie et les périphériques d'entrée/sortie. Les spécifications externes des primitives de ces modules n'ont pas été reprises dans ce dossier.

1. DISK

1.1. DESCRIPTION

Gestion du contrôleur de disques.

1.2. LISTE DES PRIMITIVES

EXTERNES

```
char *getdir(char *path);
```

2. DISPLAY

2.1. DESCRIPTION

Gestion du contrôleur d'écran.

2.2. CONSTANTES SYMBOLIQUES

Des constantes symboliques ont été définies pour:

- les couleurs,
- les attributs vidéo,
- les modes vidéo.

2.3. LISTE DES PRIMITIVES

EXTERNES

```
void init_display(void);

void setvideomode(int mode);
void setcursor(int l1,int l2);
void gotoxy(int x,int y);
int  wherex(void);
int  wherey(void);
void setvideopage(int page);
void scroll(int dir,int lines,int attr,int x1,int y1,int x2,int y2);
void scrollup(int lines,int x1,int y1,int x2,int y2);
void scrolldown(int lines,int x1,int y1,int x2,int y2);
void clrscr(int x1,int y1,int x2,int y2);
int  whatatcurs(void);
int  getcharatcurs(void);
int  getattratcurs(void);
void dot(int x,int y);
int  getvideomode(void);
void writevideomem(int x, int y, char *str);
void fillattr(char attr, int x,int y, int nb);
char *savescreen(int x1, int y1, int x2, int y2);
void restorescreen(char *scrcopy, int x1, int y1, int x2, int y2);
void refreshscreen(char *scrcopy, int x1, int y1, int x2, int y2);
```

3. KEYBRD

3.1. DESCRIPTION

Gestion du clavier.

3.2. CONSTANTES SYMBOLIQUES

Des constantes symboliques ont été définies pour différencier les touches particulières du clavier.

3.4. LISTE DES PRIMITIVES

EXTERNES

```
int getkey(void);
int is_F_key(int key);
int is_curs_key(int key);
```

IX. N I V E A U U T I L I T Y

Niveau comprenant des modules d'utilité générale.

1. UTILITAIRES GENERAUX

DESCRIPTION

Module contenant des macros d'utilité générale :

is_in(v,x1,x2)
cursornorm
cursoroff
cursoredit
cursorins
beep

2. ERROR

2.1. DESCRIPTION

Module de gestion des erreurs quelle que soit leur origine (HARD ou SOFT).

2.2. CONSTANTES SYMBOLIQUES

```
IGNORE 0
RETRY  1
ABORT  2
```

2.3. STRUCTURES DE DONNEES

```
struct DOSERROR{
       int exterror;
       char class;
       char action;
       char locus;};

typedef struct DOSERROR DE;

extern int errno;
extern char *sys_errlist[];
extern int sys_nerr;
```

2.4. LISTE DES PRIMITIVES

2.4.1. EXTERNES

```
void init_error(void);

int  harderrorhandler(int errval, int ax, int bp, int si);
void stderrorhandler(void);
```

2.4.2. INTERNES

```
char *suggestaction(char action);
void harderrormsg(int code);
void writeerrormsg(char *msg1, char *msg2);
void clearerrormsg(void);
```

2.5. SPECIFICATION DES PRIMITIVES

```
void init_error(void);
```

PRE : . Routine appelée une fois avant l'exécution de tout autre susceptible de provoquer une erreur.

POST: . Etablit une routine de gestion des erreurs futurs.

```
int harderrorhandler(int errval, int ax, int bp, int si);
```

PRE : Routine appelée par MS-DOS lors d'un erreur HARD (int 24)
. errval : code de l'erreur : 0..12 (low byte of DI reg)
. ax : valeur du registre AX positionné par MS-DOS
≥ 0 si disk error (ax & 0x00FF = failing drive number)
< 0 si device error
. bp, si : valeur des registres BP,SI positionnés par MS-DOS
Pointent vers le "device driver header of the failing drive"

POST: . Recherche toute l'information sur l'erreur
. Affiche deux messages :
- msg1 : ce que l'utilisateur est censé faire,
- msg2 : information concernant l'erreur
. Attend une intervention de l'utilisateur (ESC)
. Renvoie le code de l'erreur

```
void stderrorhandler(void);
```

PRE : Routine appelée si une erreur est détectée de manière SOFT

POST: . Recherche toute l'information sur l'erreur
. Affiche deux messages :
- msg1 : ce que l'utilisateur est censé faire,
- msg2 : information concernant l'erreur
. Attend une intervention de l'utilisateur (ESC)

REFERENCES

BIBLIOGRAPHIQUES

REFRENCES BIBLIOGRAPHIQUES

1. INTELLIGENCE ARTIFICIELLE

AIKINS J.S.
Prototypical Knowledge for Expert System
in Artificial Intelligence, 120, 1983, pp 163-210

BARR A., FEIGENBAUM
The Handbook of Artificial Intelligence
Los Altos, CA, William Kaufmann Inc, 1981

BOBROW D.G., MITTAL S., STEFIK M.J.
Expert Systems : Perils and Promise
in Communications of the ACM, Vol 29, No 9, 1986, pp 880-894

BROWNSTON L., FARRELL R., KANT E., MARTIN N.
Programming Expert Systems in OPS5 : An Introduction to Rule-Based Programming
Addison-Wesley, Reading, Ma., 1985, 459 p

HAMMOND P, SERGOT M.J.
APES : Augmented PROLOG for Expert Systems
Logic Based Systems Ltd, Richmond

HART P.
Peter Hart talks about Expert Systems
in IEEE EXPERT, Vol 1, No 1, Spring 1986, pp 96-99

JACKSON P.
Introduction to Expert Systems

LIBERT G.
Un Système Expert ? Pourquoi ? Comment ?
Exposé, journée "Systèmes Expert en Agronomie" organisée par le groupe de contact FNRS "Statistique et Informatique en Agronomie" à Gembloux, le 11 mai 1987

MICHIE D.
Expert Systems
in The Computer Journal, Vol 23, No 4, 1980, pp 369-376

MYERS W.
Introduction to Expert Systems
in IEEE EXPERT, Vol 1, No 1, Spring 1986, pp 100-109

NAU D.S.
Expert Computer Systems
in Computer, Vol 16, no 2, 1983, pp 83-85

RICH E.
Introduction to Artificial Intelligence

SCHOR M.I.
Declarative Knowledge Programming : Better than Procedural ?
in IEEE EXPERT, Vol 1, No 1, Spring 1986, pp 36-43

VAN LAMSWEERDE A.
Techniques d'Intelligence Artificielle
Notes de cours, Fac. Un. Notre Dame de la Paix, Namur, 1987

WATERMAN D.A.
A Guide to Expert Systems

2. DEVELOPPEMENT DE LOGICIELS

BODART F.
Interface Homme-Machine
Notes de cours, Fac. Un. Notre Dame de la Paix, Namur, 1987

BORLAND
TURBO C : User's and Reference Guide
Borland International Inc., California, 1987, 299 et 385 p

CHANDELON M.
Implémentation du Modèle APEX

COUTAZ J.
The Construction of User Interfaces
Lab. de Génie Informatique, France, 1987, 52 p

DUNCAN R.
Advanced MS-DOS : The Microsoft Guide foe Assembly Language and C Programmers
Microsoft Press, Washington, 1986, 467 p

KELLEY A., POHL I.
A Book on C : An Introduction to Programming in C
The Benjamin/Cummings Publishing Company Inc., 1982 , 362 p

KERNIGHAN B.W., RITCHIE D.M. traduit par BUFFENOIR T.
Le Langage C
Masson, Paris, 1986, 219 p

PARNAS D.L.
Use of Abstract Interfaces in the Development of Software for Embedded Computer Systems
in NRL report 8047, 1977, pp 106-135

SHNEIDERMAN B.
Designing the User Interface : Strategies for Effective Human Computer Interaction
Addison-Wesley, Reading Mass., 1987, 448 p

VAN LAMSWEERDE A.
Méthodologie de Développement de Logiciels
Notes de cours, Fac. Un. Notre Dame de la Paix, Namur, 1987

3. STATISTIQUE

BERK K.
Statistical Computing Software Reviews
in The American Statistician, Vol 40, No 3, 1986, pp 223-237

BOARDMAN T.J.
The Future of Statistical Computing on Desktop Computers
in The American Statistician, Vol 36, No 1, Feb 1982, pp 49-58

BOX G.E.P., COX D.R.
An Analysis of Transformations
J. Roy. Stat. Soc., B, 26, 1964, pp 211-243

BOX G.E.P., JENKINS G.M.
Time Series Analysis Forecasting and Control
Holden Day, San Francisco, 1976, 575 p

CHAMBERS J.M.
Statistical Computing : History and Trends
in The American Statistician, Vol 34, No 4, 1980, pp 238-243

JENKINS G.M.
Forecasting Seminar : Univariate Stochastic Models
Notes de séminaire

LIBERT G.
Analyse de Séries Chronologiques : Elaboration Automatique et Continue de Prévisions
Thèse de doctorat, Fac. Polytechnique, Mons, 1981, 244 p

LIBERT G.
Construction Automatique et Continue de Prévisions
in R.A.I.R.O., vol 16, n°4, 1981, pp 333-347

LIBERT G.
An Automatic Procedure for Box-Jenkins Model Building
in European Journal of Operational Research, 16, 1984, pp 95-103

MULLER M.E.
Aspects of Statistical Computing : What Packages for the 1980's Ought to Do
in The American Statistician, Vol 34, No 3, 1980, pp 159-168

TIBAUX J.P.
Recherche Opérationnelle
Notes de cours, Ec. Hautes Etudes Commerciales, Liège, 1986

??
Choix et Valeur des Méthodes de Prévision
??

Software Notices, in CSQ Computational Statistics Quarterly
BMDP Statistical Software, Vol 1, Issue 1, 1984, pp 85-88