National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Simulation, estimation des paramètres et prédiction pour un processus de Kendall.

par

Christian Mangin

B.Sc., M.Sc.

Thèse déposée à

l'École des Études Supérieures et de la Recherche

en accomplissement partiel

des exigences du diplôme de

Maîtrise es Sciences

Programme de Science des Systèmes

Université d'Ottawa

Ottawa, Ontario

Février, 1994

© Christian Mangin,1994

National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-95936-3

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Résumé

Le processus de Kendall a été proposé par Nucho (1979) comme une description adéquate de la fluctuation de la demande entre deux points. Nous utilisons une simulation pour tester une méthode d'estimation des paramètres du processus. Nous développons un algorithme pour calculer un intervalle de prédiction de la demande à 95% de certitude, à t unités de temps dans le futur, à partir d'une réalisation du processus t^* unités de temps dans le passé. Nous vérifions l'applicabilité de ce modèle sur une base de données décrivant les fluctuations de la demande sur un réseau commercial.

Remerciements

Je tiens à remercier le docteur David McDonald pour tout le soutien et l'aide qu'il m'a prodigué pour la réalisation de ce projet.

Je voudrais aussi remercier Bell Northern Research pour l'intérêt et le soutien qu'ils ont apporté au projet.

Table des Matières

1 Introduction 1

1.1 Définition du problème 1

1.2 Le modèle de Kendall 2

1.3 Estimation des paramètres 3

2 Simulation d'un processus de Kendall 5

2.1 La simulation 5

2.2 Validation du calcul 7

2.3 Calcul des estimateurs 9

2.4 Distribution des estimateurs 11

2.5 Taux de succès des prédictions 13

2.6 Qualité des estimateurs et taux de succès 17

3 Estimateurs et Intervalles de prédictions 23

3.1 L'intervalle de confiance élargi 23

3.2 Détermination de r 26

3.3 Sensibilité du paramètre r 27

3.4 Validation: Taux de succès des prédictions 33

3.5 Conclusion 40

4 Application du modèle de Kendall **41**

4.1 Les données 41

4.2 Niveau client-client 42

4.3 Niveaux de regroupement supérieurs 51

4.4 Regroupement au niveau ville-ville 53

4.5 Regroupement sur les types de circuits 56

5 Conclusion **59**

A Programme de simulation **61**

A.1 Algorithmes de simulation 62

A.2 Variables globales et fonctions utilitaires 74

A.3 Routines graphiques 79

A.4 Routines d'entrées / sorties 92

A.5 Routines des fenêtres de 'dialogue' 100

A.6 Routine de gestion de l'interface 108

A.7 Initialisation et programme principal 113

B Algorithme: calcul de r **116**

C Thesis Long Summary **118**

Liste des Figures

2.1 Simulation d'un processus de Kendall 6

2.2 Approximation de $\sum_{k=1}^{N} W_k \eta_k$: Méthode 1 10

2.3 Approximation de $\sum_{k=1}^{N} W_k \eta_k$: Méthode 2 10

2.4 Distribution de $\hat{\lambda}$ 12

2.5 Distribution de $\hat{\mu}$ 12

2.6 Distribution réelle et intervalle estimé 14

2.7 Taux de succès des prédictions 15

2.8 Proportion des trajectoires complètes à l'intérieur des bornes. 16

2.9 Taux de succès des prédictions: Estimation sur 10 semaines 17

2.10 Taux de succès des prédictions: Estimation sur 20 semaines 18

2.11 Taux de succès des prédictions: Estimation sur 40 semaines 19

2.12 Largeur de l'intervalle de confiance de 95%: Estimation sur 10 semaines. 20

2.13 Largeur de l'intervalle de confiance de 95%: Estimation sur 20 semaines. 21

2.14 Largeur de l'intervalle de confiance de 95%: Estimation sur 40 semaines. 22

3.1 Élargissement de l'intervalle de confiance 24

3.2 Paramètre τ versus t/t^*: exemple 1 28

3.3 Paramètre τ versus t/t^*: exemple 2 29

3.4 Paramètre τ versus t/t^*: exemple 3 30

3.5 Paramètre r versus t/t^*: exemple 4 31
3.6 Paramètre r versus t/t^*: exemple 5 32
3.7 Taux de succès des prédictions: Estimation sur 10 semaines 34
3.8 Taux de succès des prédictions: Estimation sur 20 semaines 35
3.9 Taux de succès des prédictions: Estimation sur 40 semaines 36
3.10 Intervalle de confiance de 95%: Estimation sur 10 semaines 37
3.11 Intervalle de confiance de 95%: Estimation sur 20 semaines 38
3.12 Intervalle de confiance de 95%: Estimation sur 40 semaines 39

4.1 Niveau client-client: Prédiction sans suppression de saut (A) 44
4.2 Niveau client-client: Prédiction avec suppression de saut (A) 44
4.3 Niveau client-client: Prédiction sans suppression de saut (B) 45
4.4 Niveau client-client: Prédiction avec suppression de saut (B) 45
4.5 Prédiction au niveau client-client: exemple 1 46
4.6 Prédiction au niveau client-client: exemple 2 46
4.7 Prédiction au niveau client-client: exemple 3 47
4.8 Prédiction au niveau client-client: exemple 4 47
4.9 Prédiction au niveau client-client: exemple 5 48
4.10 Prédiction au niveau client-client: exemple 6 48
4.11 Prédiction au niveau client-client: exemple 7 49
4.12 Prédiction au niveau client-client: exemple 8 49
4.13 Prédiction au niveau client-client: exemple 9 50
4.14 Prédiction au niveau client-client: exemple 10 50
4.15 Qualité des prédictions aux niveau client-client 52
4.16 Qualité des prédictions aux niveau ville-ville 52
4.17 Qualité des prédictions aux niveau province-province 53

4.18 Prédiction au niveau ville-ville: exemple 1 54
4.19 Prédiction au niveau ville-ville: exemple 2 54
4.20 Prédiction au niveau ville-ville: exemple 3 55
4.21 Prédiction au niveau ville-ville: exemple 4 55
4.22 Regroupement par type de circuit: exemple 1 56
4.23 Regroupement par type de circuit: exemple 2 57
4.24 Regroupement par type de circuit: exemple 3 57
4.25 Regroupement par type de circuit: exemple 4 58
4.26 Regroupement par type de circuit: exemple 5 58

Chapitre 1

Introduction

1.1 Définition du problème

Pour une compagnie de télécommunication, un problème important consiste à prévoir l'évolution de la demande entre deux points. Lorsque la demande dépasse la capacité du réseau, le surplus doit être réacheminé par d'autre voies, ce qui entraîne généralement des coûts beaucoup plus élevé pour le service. Pour éliminer ces coûts une approche consisterait à installer une grande quantité d'équipements à tous les endroits où l'on appréhende un excès possible de la demande. Mais le résultat serait que la plus grande partie de ces équipements risqueraient d'être sous-utilisés. Pour des raisons d'efficacité il serait donc plus raisonnable d'installer une quantité d'équipements qui serait tout juste suffisante pour répondre à la demande pour disons 95% des cas. Pour obtenir des prédictions sur la demande future, une méthode consisterait à prendre les données du passé et utiliser une technique de régression. Cependant en utilisant un modèle stochastique, il devient possible d'intégrer certaines propriétés que ces systèmes devraient avoir intuitivement. Dans ce travail, nous allons utiliser un modèle stochastique appelé "modèle de Kendall". Nous al-

lons présenter ce modèle, nous allons voir comment ses paramètres peuvent être déterminés. Nous allons utiliser la simulation sur ordinateur, pour vérifier cette méthode d'estimation. Nous verrons pourquoi la formule de la variance du modèle est insuffisante pour un intervalle de confiance de 95%. Nous verrons comment ce problème peut être corrigé à l'aide d'un algorithme que nous avons développé. Finalement nous allons appliquer ce modéle sur des données d'un réseau commercial.

1.2 Le modèle de Kendall

Nous allons utiliser le processus de Kendall pour décrire la fluctuation de la demande entre deux points. Dans ce modèle, le taux de fluctuations de la demande est proportionnel au niveau de cette demande. Soit $X(t)$ la demande au temps t. Si $N(t)$ compte le nombre de nouvelles connections, jusqu'au temps t et $M(t)$ compte le nombre de déconnections jusqu'au temps t alors nous avons:

$$dX(t) = dN(t) - dM(t)$$

Nous dénotons l'intensité de ces processus ponctuels par $\lambda X(t)$ et $\mu X(t)$. Lorsque $\lambda > \mu$, le processus stochastique $X(t)$ avec valeur initiale $n(0)$ est transitoire. Puisque la probabilité d'absorption à 0 est négligeable, nous supposons que 0 est un point de réflection. La moyenne $m(t)$ et la variance $v(t)$ augmentent selon les formules suivantes (Nucho 1979):

$$m(t) = n(0)e^{(\lambda-\mu)t}$$

$$v(t) = n(0)\frac{\lambda+\mu}{\lambda-\mu}e^{2t(\lambda-\mu)}[1 - e^{-(\lambda-\mu)t}]$$

Si $\lambda = \mu$ alors

$$m(t) = n(0)$$

$$v(t) = 2\lambda n(0)t$$

On peut aussi démontrer que si $\lambda > \mu$ le processus $X(t)$ est asymptotiquement normal, i.e.:

$$\frac{X(t) - m(t)}{\sqrt{v(t)}} \rightarrow N(0,1).$$

Le but de notre analyse est l'estimation des paramètres λ et μ à partir d'une réalisation du processus sur une période de longueur t^* dans le passé. Avec les estimateurs $\hat{\lambda}$ et $\hat{\mu}$ nous voulons donner un intervalle de prédiction pour la demande future t unités de temps plus tard.

1.3 Estimation des paramètres

Dans cette section nous définissons des estimateurs pour les paramètres λ et μ. Supposons que pendant la période $[0, t^*]$ il y a eu un total de N transactions. Dénotons les temps entre deux transactions consécutives par $\{ \eta_k : k = 1, .., N \}$. Nous dénotons le nombre en place durant l'intervalle k par W_k. Si $W_k = n$ alors η_k est un nombre aléatoire avec distribution exponentielle et espérance $1/[n(\lambda + \mu)]$. Soit $Y_k := \exp(-W_k\eta_k)$ alors nous avons pour $0 \leq t \leq 1$:

$$P(Y_k \leq t | W_k = n) = P(\eta_k \leq -\frac{1}{n} \ln t | W_k = n) = t^{\lambda+\mu}$$

Nous pouvons en conclure que les variables Y_k sont indépendantes et identiquement distribuées. On peut aussi montrer que

$$\hat{\beta} := -\frac{(\sum_{k=1}^{N} \ln(Y_k)}{N} = \frac{\sum_{k=1}^{N} W_k\eta_k}{N}$$

est un estimateur de vraisemblance maximale pour $\beta := 1/(\lambda + \mu)$. L'estimateur $\hat{\beta}$ est asymptotiquement normal puisque les variables $\{Y_k = W_k\eta_k\}$ sont indépendantes

et identiquement distribuées avec distribution exponentielle et espérance égale à β. Nous avons donc

$$\sqrt{N}\frac{\hat{\beta} - \frac{1}{\lambda+\mu}}{\hat{\beta}} \to N(0,1) \tag{1.3.1}$$

Nous pouvons aussi avoir un estimateur indépendant pour le ratio $\lambda/(\lambda+\mu)$.

Définissons $\{\xi_k : k = 1, \cdots, N\}$ par $+1$ pour une connection et -1 pour une déconnection. Alors l'estimateur de vraisemblance maximale pour $\lambda/(\lambda+\mu)$ est:

$$\hat{\rho} := \sum_{k=1}^{N} \frac{\xi_k}{2N} + \frac{1}{2}$$

L'estimateur $\hat{\rho}$ est asymptotiquement normal:

$$\sqrt{N}\frac{\hat{\rho} - \frac{\lambda}{\lambda+\mu}}{\sqrt{\hat{\rho}(1-\hat{\rho})}} \to N(0,1).$$

Il suffit d'un peu d'arithmétique pour trouver que $\hat{\alpha} = (2\hat{\rho} - 1)/\hat{\beta}$ est un estimateur de vraisemblance maximale indépendant pour $\alpha := \lambda - \mu$. On peut aussi montrer que:

$$\sqrt{N}\hat{\beta}[\hat{\alpha} - (\lambda - \mu)] \to N(0,1) \tag{1.3.2}$$

Finalement les estimateurs de vraisemblance maximale pour λ et μ sont:

$$\hat{\lambda} := \frac{\sum_{k=1}^{N}\{\xi_k = 1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

$$\hat{\mu} := \frac{\sum_{k=1}^{N}\{\xi_k = -1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

Chapitre 2

Simulation d'un processus de Kendall

2.1 La simulation

Pour étudier le processus de Kendall nous avons utiliserons une simulation sur ordinateur. En partant d'une demande initiale $X[0]$ nous voulons obtenir la valeur de la demande à des intervalles d'une semaine, ainsi que du nombre total de connections et de déconnections qui se sont produites durant la semaine. On dénote par t_i le temps en jours du début de l'$i^{ème}$ journée. Si $X[i]$ est la demande au temps t_i, et si U_1 et U_2 sont des variables aléatoires pseudo-uniformes sur $[0,1]$ alors les fonctions cumulative de distributions du nombre de connections C_i et du nombre de déconnections D_i sont:

$$F_1(C_i) = \sum_{j=0}^{C_i} e^{-\lambda X[i]} \frac{(\lambda X[i])^j}{j!}$$

$$F_2(D_i) = \sum_{j=0}^{D_i} e^{-\mu X[i]} \frac{(\mu X[i])^j}{j!}$$

et la demande au temps t_{i+1} est:

$$X[i+1] = X[i] + F_1^{-1}(U_1) - F_2^{-1}(U_2)$$

Avec des valeurs prédéterminées de λ et μ (données en $[jours]^{-1}$) et une valeur

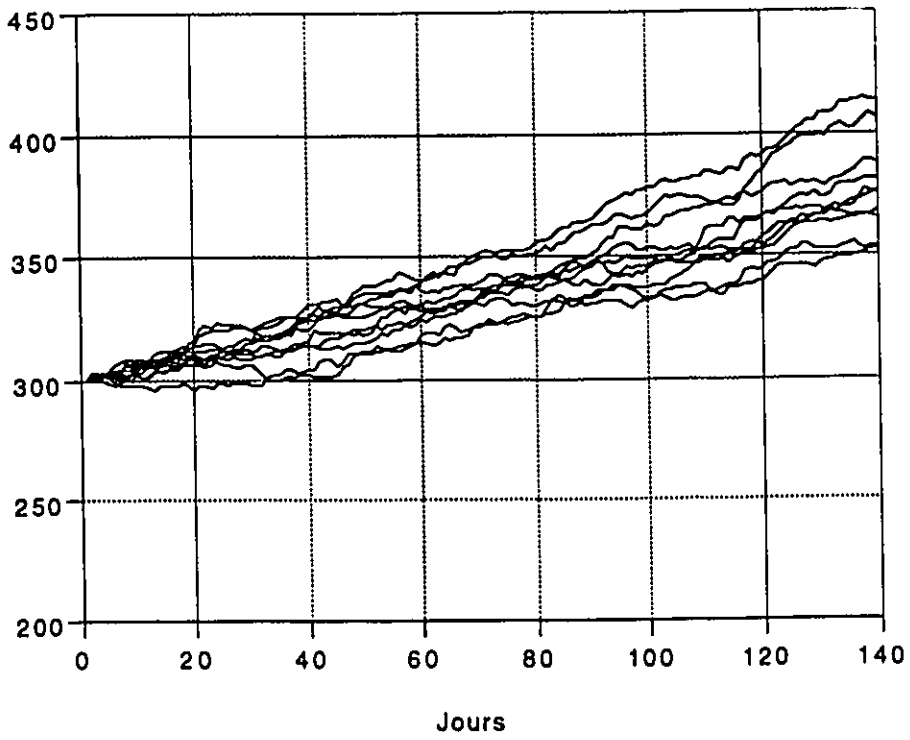

Figure 2.1: Simulation d'un processus de Kendall

initiale pour la demande nous pouvons simuler l'évolution de la demande sur un nombre donné de jours. Nous avons développé un programme pour nous permettre de générer ces trajectoires. À la fin de chaque semaine simulée, un enregistrement du niveau de la demande et du nombre total de transactions générées pendant la semaine est produit. C'est avec ces enregistrements que nous calculerons nos estimateurs et testerons nos prédictions.

Ce programme possède une interface qui permet de modifier aisément les conditions de la simulation. L'utilisateur peut décider de la longueur des périodes d'estimations et de prédictions, c'est-à-dire du nombre de semaines sur lequel on se base pour calculer les estimateurs, et du nombre de semaines sur lesquelles les prédictions sont testées. L'utilisateur peut aussi fixer la valeur de départ de la simu-

lation, le nombre de semaines à simuler, la valeurs des paramètres du processus ainsi le nombre de trajectoires à générer. La figure 2.1 illustre le genre de trajectoires qui sont générés. Ce programme a été écrit avec Think Pascal 4.0 sur Apple MacIntosh et est présenté intégralement dans l'appendice A.

2.2 Validation du calcul

Nous présentons maintenant une validation de l'algorithme de simulation. En utilisant les paramètres λ et μ de la simulation et les formules de la moyenne et de la variance nous avons calculé le pourcentage de succès des prédictions pour 2 écart-types. Les colonnes du tableau I indique la valeur des paramètres choisis. Chaque colonne du tableau II montre l'évolution du taux de succès des prédictions sur 39 semaines calculé sur un total de 1000 trajectoires. On voit que le taux de succès se maintient toujours assez près de 95%.

Tableau I

Paramètres	S1	S2	S3	S4	S5	S6	S7
λ	0.00440	0.00840	0.01540	0.00170	0.00095	0.00046	0.02320
μ	0.00310	0.00610	0.01410	0.00140	0.00078	0.00021	0.02160
X_0	600	900	600	900	900	900	300

Tableau II

Semaine	S1	S2	S3	S4	S5	S6	S7
1	0.942	0.967	0.954	0.953	0.960	0.939	0.954
2	0.956	0.970	0.961	0.948	0.956	0.958	0.950
3	0.952	0.969	0.960	0.946	0.950	0.962	0.954
4	0.946	0.958	0.944	0.948	0.956	0.944	0.965
5	0.951	0.956	0.946	0.967	0.954	0.965	0.960
6	0.955	0.956	0.943	0.960	0.955	0.955	0.967
7	0.965	0.957	0.951	0.958	0.950	0.953	0.966
8	0.963	0.956	0.951	0.957	0.956	0.941	0.965
9	0.963	0.951	0.957	0.953	0.956	0.950	0.959
10	0.970	0.953	0.958	0.955	0.960	0.953	0.958
11	0.960	0.948	0.956	0.942	0.960	0.953	0.951
12	0.962	0.950	0.955	0.951	0.964	0.955	0.957
13	0.964	0.953	0.963	0.952	0.963	0.952	0.958
14	0.956	0.952	0.962	0.946	0.960	0.950	0.965
15	0.960	0.944	0.962	0.948	0.963	0.947	0.960
16	0.959	0.942	0.967	0.950	0.964	0.950	0.957
17	0.963	0.944	0.960	0.950	0.972	0.958	0.956
18	0.960	0.949	0.962	0.954	0.965	0.956	0.958
19	0.962	0.943	0.965	0.950	0.968	0.952	0.960
20	0.958	0.943	0.966	0.949	0.964	0.946	0.954
21	0.955	0.943	0.961	0.947	0.963	0.941	0.954
22	0.961	0.943	0.963	0.945	0.971	0.941	0.960
23	0.952	0.944	0.966	0.947	0.969	0.942	0.959
24	0.955	0.943	0.965	0.949	0.966	0.941	0.957
25	0.958	0.942	0.958	0.947	0.969	0.938	0.957
26	0.953	0.937	0.957	0.948	0.971	0.942	0.955
27	0.960	0.937	0.952	0.947	0.972	0.943	0.955
28	0.953	0.936	0.953	0.946	0.974	0.942	0.960
29	0.951	0.937	0.954	0.942	0.971	0.943	0.949
30	0.951	0.940	0.948	0.948	0.970	0.941	0.959
31	0.950	0.939	0.951	0.949	0.971	0.943	0.958
32	0.952	0.939	0.952	0.951	0.963	0.934	0.953
33	0.955	0.939	0.953	0.953	0.963	0.936	0.954
34	0.959	0.936	0.954	0.952	0.960	0.944	0.951
35	0.955	0.939	0.955	0.950	0.959	0.940	0.954
36	0.958	0.944	0.956	0.954	0.961	0.944	0.958
37	0.959	0.944	0.951	0.956	0.959	0.943	0.962
38	0.967	0.942	0.954	0.952	0.957	0.946	0.961
39	0.959	0.943	0.958	0.951	0.960	0.946	0.955

2.3 Calcul des estimateurs

Les estimateurs de vraisemblance maximale sont définies par

$$\hat{\lambda} := \frac{\sum_{k=1}^{N}\{\xi_k = 1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

$$\hat{\mu} := \frac{\sum_{k=1}^{N}\{\xi_k = -1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

La somme des ξ_k ne pose aucun problème puisqu'il s'agit tout simplement du nombre total de transactions (connections pour $\hat{\lambda}$, déconnections pour $\hat{\mu}$) qui se sont produites pendant l'intervalle de temps. Cependant nous ne savons rien des temps η_k entre deux transactions consécutives. En pratique, nous ne connaissons que le nombre en place à des temps fixes (à la fin de la semaine), ainsi que le nombre total de transactions dans les intervalles (d'une semaine). Il est possible cependant d'obtenir une bonne approximation de $\sum_k W_k \eta_k$. Supposons que pour un nombre de temps donnés $t_1, ..., t_L$ séparés par un intervalle de temps fixe Δt, nous connaissons le nombre en place $X[i]$, alors:

$$\sum_{i=1}^{L} X[i]\Delta t \approx \sum_{k=1}^{N} W_k \eta_k$$

Comme l'illustre la figure 2.2, cette approximation s'apparente à une technique d'intégration numérique d'une fonction.

Les estimateurs peuvent donc être évalués par:

$$\hat{\lambda} := \frac{\sum_{i=1}^{L} C_i}{\sum_{i=1}^{L} X[i]\Delta t}$$

$$\hat{\mu} := \frac{\sum_{i=1}^{L} D_i}{\sum_{i=1}^{L} X[i]\Delta t}$$

où C_i et D_i sont les nombres de transactions dans les intervalles correspondants. Il est possible d'améliorer légèrement cette approximation. Supposons que $X[i]$ et $X[i+1]$ sont deux nombres en place successifs. Dans notre première méthode

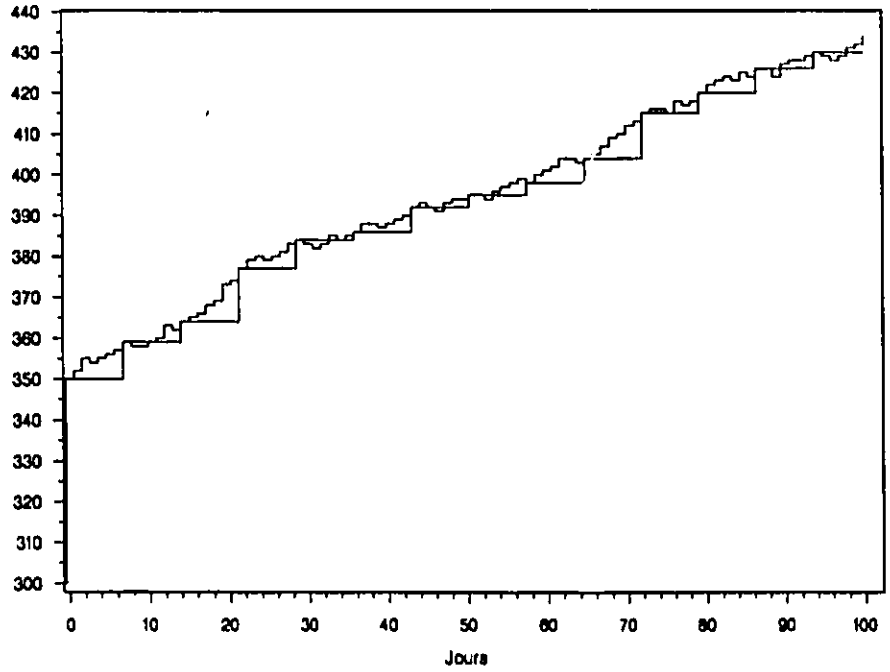

Figure 2.2: Approximation de $\sum_{k=1}^{N} W_k \eta_k$: Méthode 1

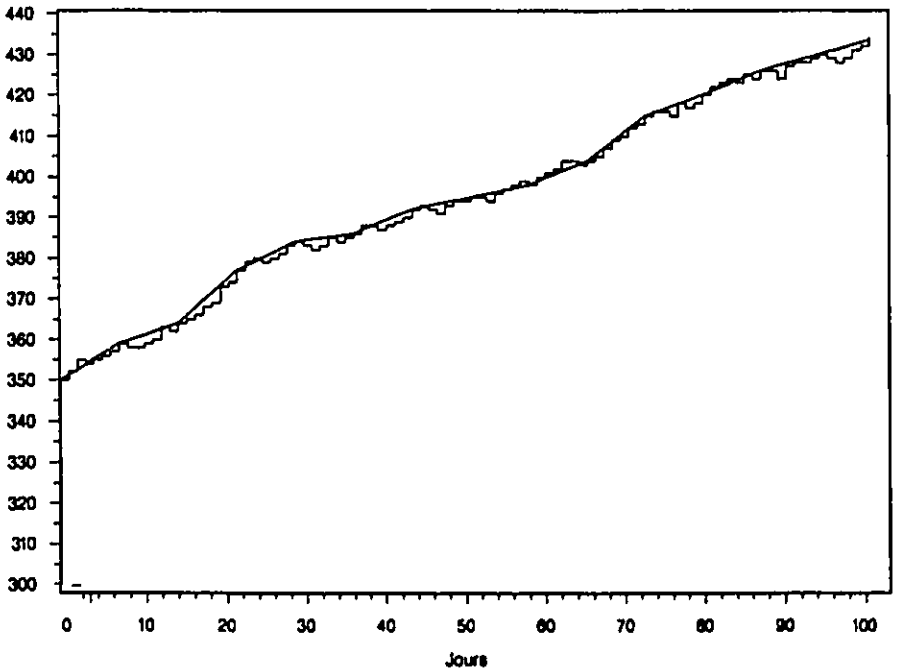

Figure 2.3: Approximation de $\sum_{k=1}^{N} W_k \eta_k$: Méthode 2

d'interpolation, nous supposons que $X(t) = X[i]$ sur tout l'intervalle entre t_i et t_{i+1}. Nous pouvons aussi supposer que le nombre en place évolue selon l'espérance d'un processus de Kendall (voir figure 2.3),

$$X(t) = X[i]e^{h_i(t-t_i)}$$

h_i étant spécifié par la condition à la limite:

$$X[i+1] = X[i]e^{h_i\Delta t}$$

Il suffit alors d'intégrer entre t_i et t_{i+1}

$$\hat{X}_i = \int_{t_i}^{t_{i+1}} X[i]e^{h_i(t-t_i)}dt$$

pour obtenir:

$$\hat{X}_i = X[i]\frac{e^{h_i\Delta t} - 1}{h_i}$$

Lorsque h_i est petit, l'expression peut se simplifier à

$$\hat{X}_i = X[i]\Delta t(1 + h_i\Delta t/2)$$

Les estimateurs corrigés sont donc:

$$\hat{\lambda} = \frac{\sum_{i=1}^{L} C_i}{\sum_{i=1}^{L} X[i]\Delta t(1 + \frac{h_i\Delta t}{2})}$$

$$\hat{\mu} = \frac{\sum_{i=1}^{L} D_i}{\sum_{i=1}^{L} X[i]\Delta t(1 + \frac{h_i\Delta t}{2})}$$

2.4 Distribution des estimateurs

Pour chaque trajectoire générée, nous calculons les estimateurs $\hat{\lambda}$ et $\hat{\mu}$. Un premier résultat est que nous pouvons constater que les estimateurs tendent à être normalement distribués. Les figures 2.4 et 2.5 montrent un exemple d'histogramme de ces estimateurs. (1000 trajectoires générés avec $\lambda = 3.2 \times 10^{-3}[jours]^{-1}$ et $\mu = 1.6 \times 10^{-3}[jours]^{-1}$.)

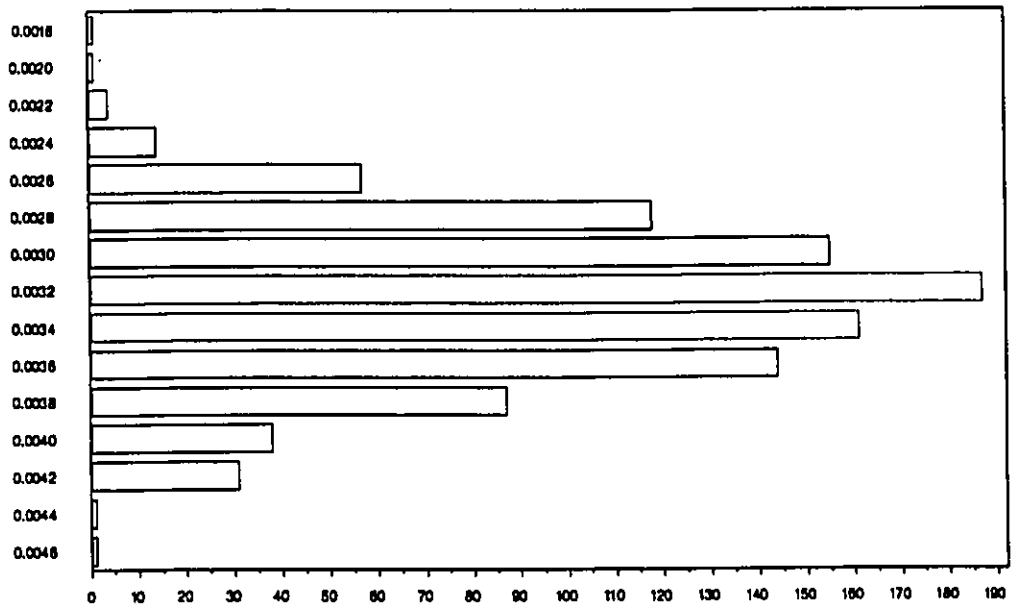

Figure 2.4: Distribution de $\hat{\lambda}$

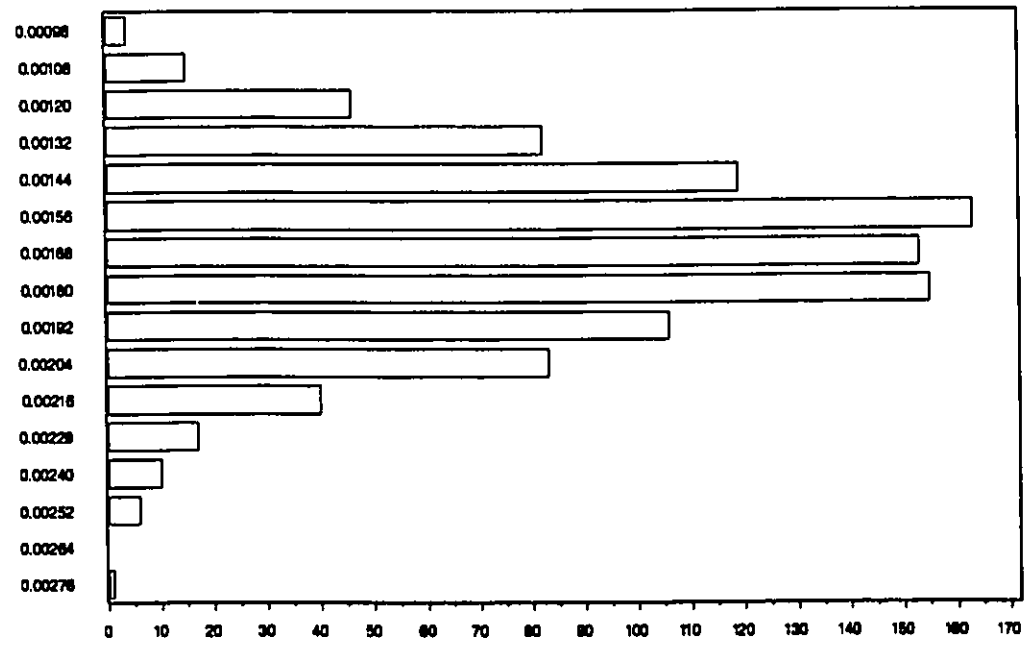

Figure 2.5: Distribution de $\hat{\mu}$

2.5 Taux de succès des prédictions

On se sert des estimateurs $\hat{\lambda}$ et $\hat{\mu}$ pour calculer les intervalles de prédictions t unités de temps dans l'avenir. Puisque ce ne sont pas les valeurs exactes de λ et μ, il s'ensuit que:

- La moyenne prédite

$$\hat{m}(t) = X_0 e^{(\hat{\lambda}-\hat{\mu})t},$$

ne correspondra généralement pas à la moyenne réelle de la distribution:

$$m(t) := X_0 e^{(\lambda-\mu)t}.$$

- Les intervalles de deux écart-types calculés, $\hat{m}(t) \pm 2\hat{\sigma}(t)$, ne seront pas parfaitement de la bonne largeur, $m(t) \pm 2\sigma(t)$, et ne seront pas centrés sur la moyenne réelle de la distribution.

La conséquence est que en général, nos intervalles de prédictions pour deux écart-types capturent moins de 95% de la distribution. La figure 2.6 illustre cette situation.

La figure 2.7 nous montre le taux de succès de nos prédictions par semaine. Nous définissons le taux de succès comme la fraction des trajectoires pour lesquelles le nombre en place ou niveau de la demande s'est maintenu à l'intérieur des bornes de deux écart-types pour une semaine donnée.

On voit que ce taux de succès tend à diminuer avec le temps. Il ne faut pas confondre le taux de succès des prédictions avec la proportion des trajectoires complètes qui sont restées à l'intérieur des bornes après un nombre donné de semaines. Cette proportion est en général beaucoup plus basse (voir figure 2.8).

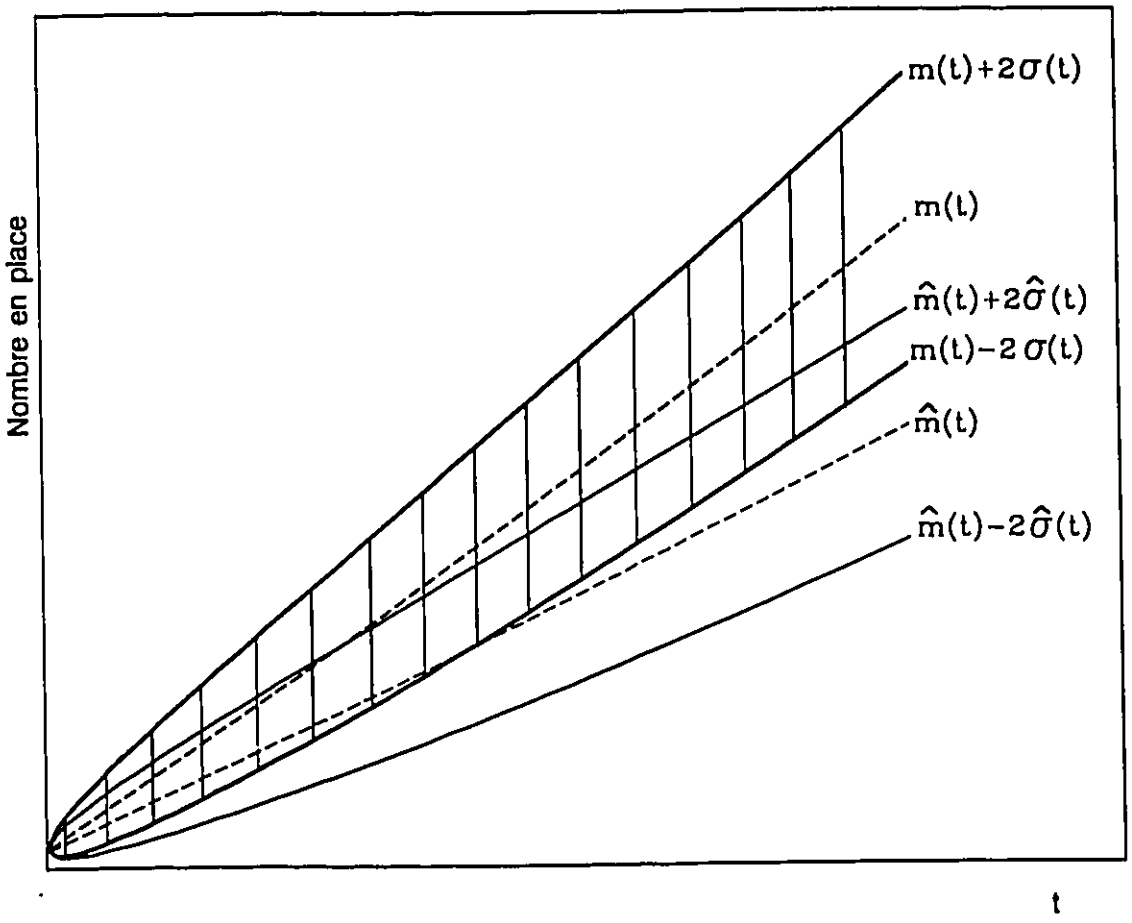

Figure 2.6: Distribution réelle et intervalle estimé

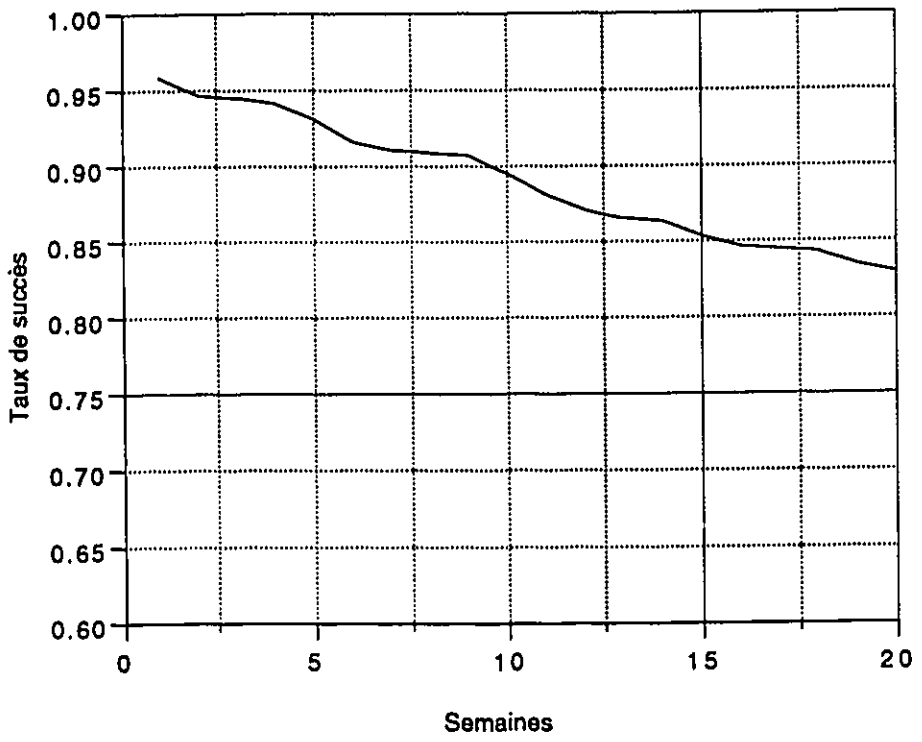

Figure 2.7: Taux de succès des prédictions

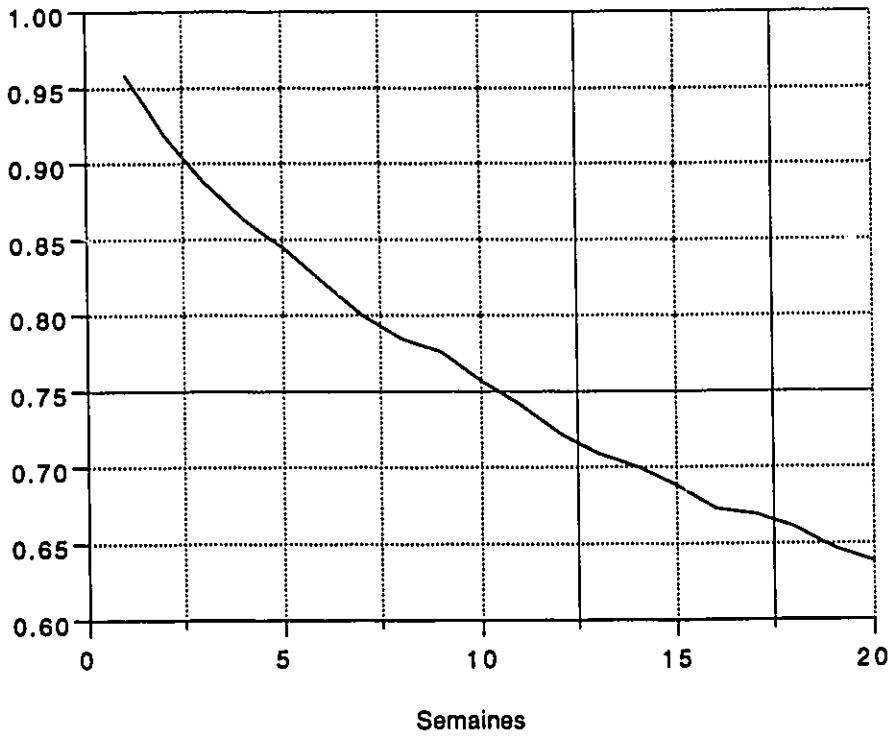

Figure 2.8: Proportion des trajectoires complètes à l'intérieur des bornes.

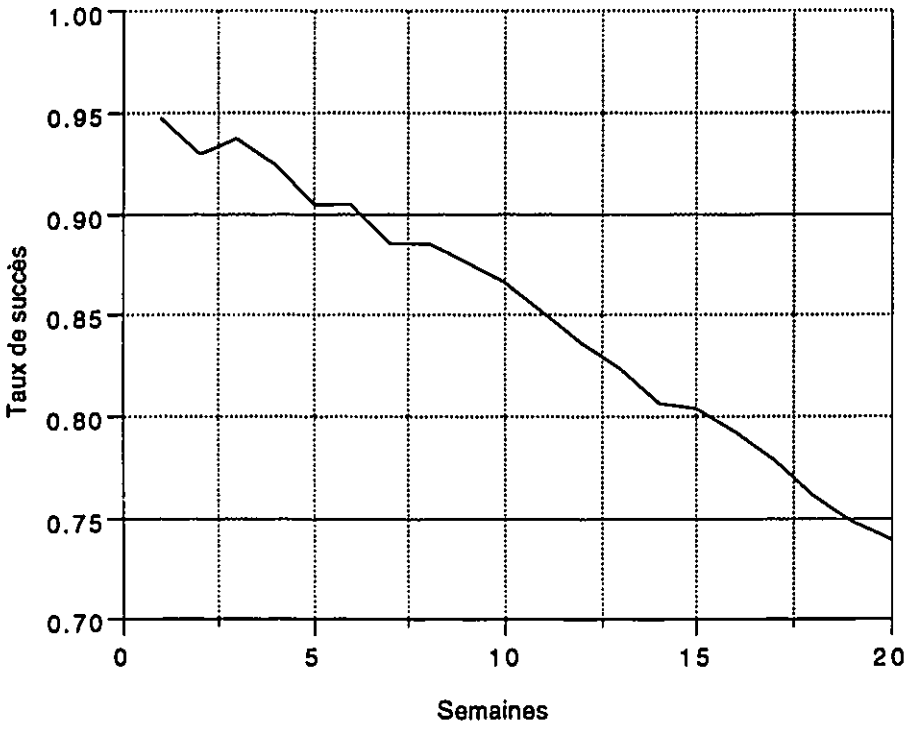

Figure 2.9: Taux de succès des prédictions: Estimation sur 10 semaines

2.6 Qualité des estimateurs et taux de succès

Il y a une relation directe entre le taux de succès de nos prédictions et la qualité des estimateurs. On peut observer que plus la longueur de la période sur laquelle nos estimateurs sont calculés est large, moins rapide est la chute de notre taux de succès. Les figures 2.9, 2.10 et 2.11 illustre cette relation. Dans ces trois cas nous avons utilisé: $\lambda = 2.5 \times 10^{-3}[jours]^{-1}$, $\mu = 1.1 \times 10^{-3}[jours]^{-1}$, $X_0 = 400$ et le nombre de trajectoires générées est de 1000.

Dans chacun des cas nous voyons que ne pouvons pas obtenir un taux de succès constant dans le temps avec les expressions de Nucho. Il est donc nécessaire d'élargir ces intervalles.

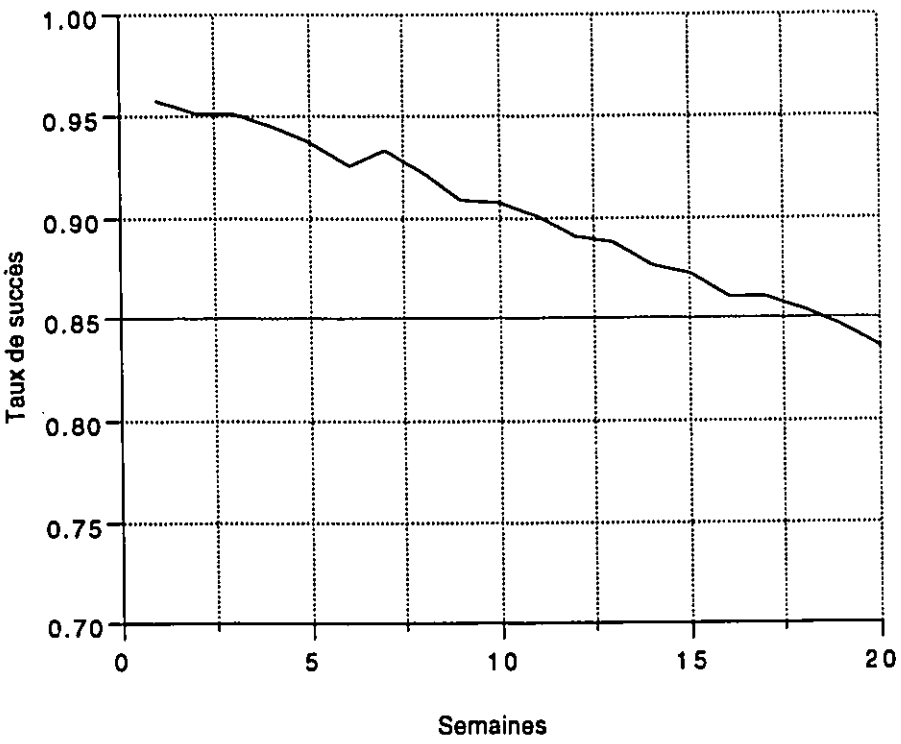

Figure 2.10: Taux de succès des prédictions: Estimation sur 20 semaines

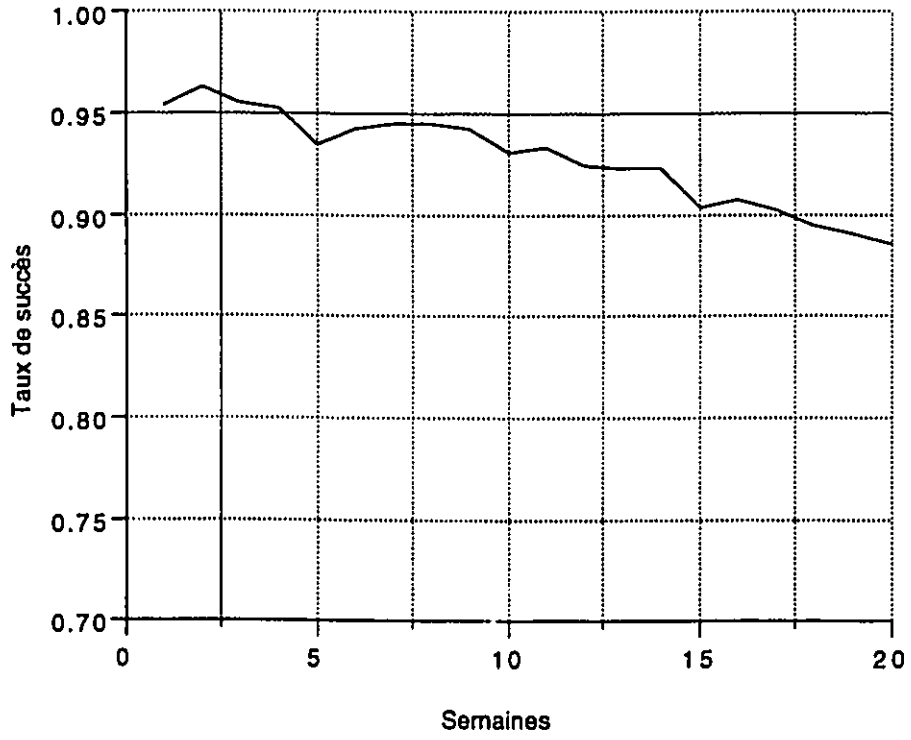

Figure 2.11: Taux de succès des prédictions: Estimation sur 40 semaines

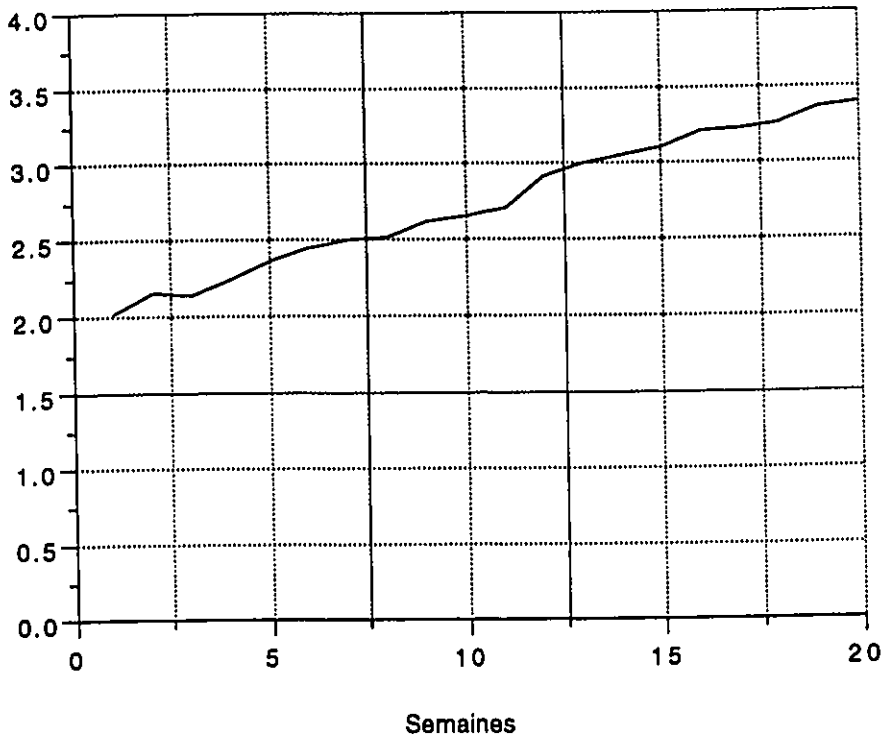

Figure 2.12: Largeur de l'intervalle de confiance de 95%: Estimation sur 10 semaines.

Les figures 2.12, 2.13 et 2.14 donnent une idée de l'élargissement nécessaire. Dans ces trois graphiques l'échelle verticale est en écart-types (en utilisant la formule de Nucho avec $\lambda = \hat{\lambda}$ et $\mu = \hat{\mu}$). La ligne indique la largeur minimale de l'intervalle de confiance qu'il aurait fallu pour avoir 95% de succès, en supposant un élargissement symétrique autour de la moyenne estimée.

En conclusion, si on veut obtenir un intervalle de confiance qui puisse capturer 95% de la distribution en moyenne, et que cette fraction reste constante dans le temps, il faut trouver une façon d'élargir les bornes de nos prédictions. Le chapitre 3 est consacré à cette question.

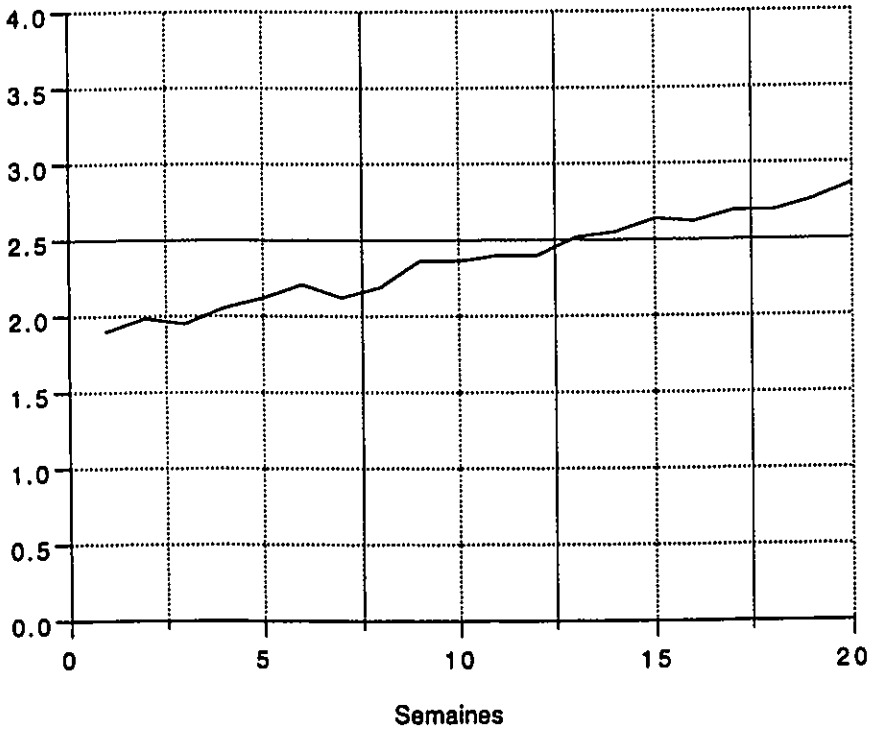

Figure 2.13: Largeur de l'intervalle de confiance de 95%: Estimation sur 20 semaines.

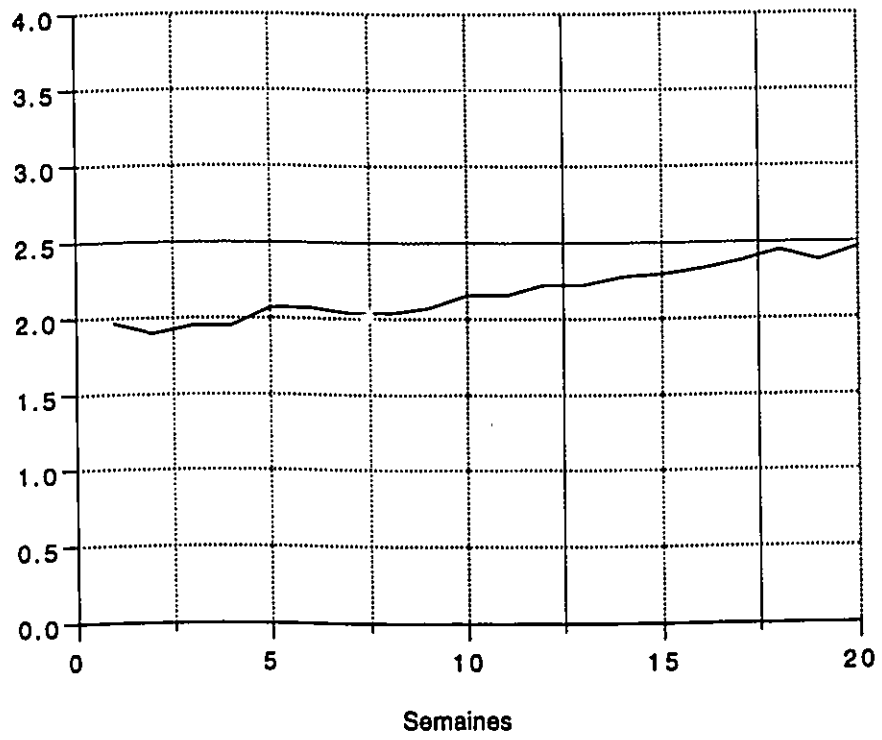

Figure 2.14: Largeur de l'intervalle de confiance de 95%: Estimation sur 40 semaines.

Chapitre 3

Estimateurs et Intervalles de prédictions

3.1 L'intervalle de confiance élargi

Comment peut-on obtenir un intervalle de confiance de 95% avec les estimateurs que l'on dispose ? En général, l'expression pour la variance de Nucho est insuffisante puisqu'elle permet de capturer en moyenne 95% de la distribution seulement si l'on connait exactement les paramètres du processus. Cet chapitre est donc consacré à trouver un algorithme qui permettra de compenser l'erreur des estimateurs par un élargissement de l'intervalle de confiance de Nucho. On cherche un algorithme qui nous donne un intervalle de confiance avec les propriétées suivantes:

- L'intervalle de confiance élargie tend à se réduire à l'intervalle donné par la formule de Nucho lorsque l'écart-type des estimateurs tend vers zéro.
- L'intervalle de confiance doit être symétrique autour de la moyenne estimée.

Si les estimateurs sont $\hat{\lambda}$ et $\hat{\mu}$, la moyenne estimée par la formule de Nucho est:

$$\hat{m}(t) = X_0 e^{(\hat{\lambda}-\hat{\mu})t}$$

On peut garantir un intervalle de confiance supérieur ou égal à 95% avec:

$$\sigma_e(t) = \sigma(t) + \frac{|m(t) - \hat{m}(t)|}{2}$$

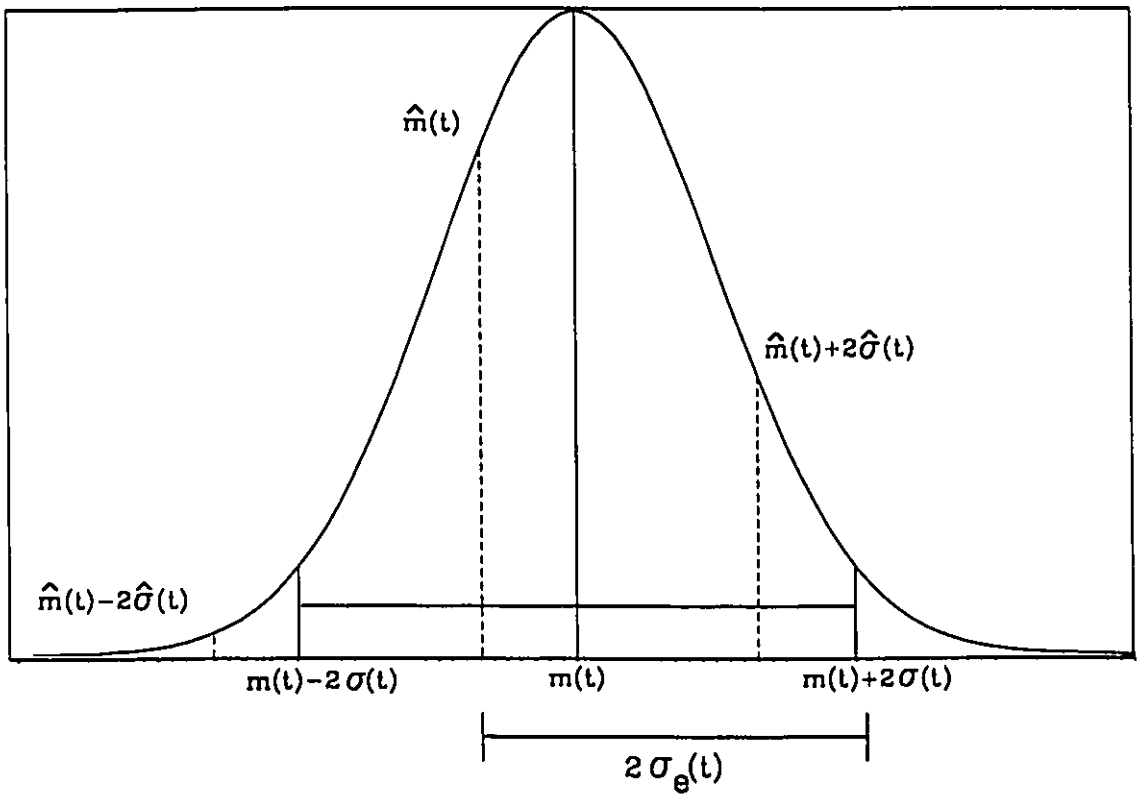

Figure 3.1: Élargissement de l'intervalle de confiance

Il serait donc possible de trouver une valeur adéquate pour $\sigma_e(t)$ en remplaçant les valeurs de $\sigma(t)$ et $m(t)$ par une *surestimation* de $\hat{m}(t)$ et $\hat{\sigma}(t)$. Pour ce faire nous allons remplacer les estimateurs $\hat{\lambda}$ et $\hat{\mu}$ par $\hat{\alpha}$ et $\hat{\beta}$:

$$\hat{\alpha} = \hat{\lambda} - \hat{\mu}$$

$$\hat{\beta} = \frac{1}{\hat{\lambda} + \hat{\mu}}$$

Les formules de Nucho s'écrivent alors:

$$\hat{m}(t) = X_0 e^{\hat{\alpha}t}$$

$$\hat{v}(t) = \frac{X_0}{\hat{\alpha}\hat{\beta}} e^{\hat{\alpha}t}(e^{\hat{\alpha}t} - 1)$$

L'avantage d'utiliser ces nouvelles variables est que nous possédons maintenant un estimé de l'écart-type de ces estimateurs (voir équation 1.3.1 et 1.3.2):

$$\hat{\sigma}_{\hat{\alpha}} = \frac{\hat{\beta}}{\sqrt{N}}$$

$$\hat{\sigma}_{\hat{\beta}} = \frac{1}{\sqrt{N}\hat{\beta}}$$

Nous pouvons obtenir une surestimation de $\hat{m}(t)$ en surestimant $\hat{\alpha}$. Soit:

$$\hat{\alpha}_s = \hat{\alpha} + r\hat{\sigma}_{\hat{\alpha}}$$

où r est un nombre non-négatif à déterminer. Alors

$$\hat{m}_s(t) = X_0 e^{\hat{\alpha}_s t}$$

est une surestimation de $\hat{m}(t)$. De la même façon, on peut surestimer l'écart-type $\hat{\sigma}(t)$ en utilisant:

$$\hat{\beta}_s = \hat{\beta} - r\hat{\sigma}_{\hat{\beta}}$$

et

$$\hat{\sigma}_s(t) = X_0 \frac{1}{\hat{\alpha}_s \hat{\beta}_s} e^{\hat{\alpha}_s t}(e^{\hat{\alpha}_s t} - 1)$$

Alors l'intervalle de confiance étendu est donné par:

$$\hat{\sigma}_e(t) = \hat{\sigma}_s(t) + \frac{|\hat{m}_s(t) - \hat{m}(t)|}{2}$$

Notons que lorsque $\hat{\sigma}_{\hat{\alpha}} \to 0$ et $\hat{\sigma}_{\hat{\beta}} \to 0$ il s'ensuit que $\hat{m}_s(t) \to \hat{m}(t)$ et $\hat{\sigma}_s(t) \to \hat{\sigma}(t)$ et nous retrouvons à la limite l'expression de Nucho:

$$\hat{\sigma}_e(t) = \hat{\sigma}(t)$$

et l'intervalle est donc:

$$\hat{m}(t) \pm 2\hat{\sigma}_e(t).$$

3.2 Détermination de r

Le problème est maintenant de trouver le r minimal qui permettra de capturer en moyenne 95% de la distribution. Idéalement on voudrait que ce paramètre ne dépende pas des valeurs de α et β (voir section 1.3), autrement notre approche perdrait son utilité. Nous utilisons le fait que les estimateurs $\hat{\alpha}$ et $\hat{\beta}$ sont indépendants et asymptotiquement normal (equation 1.3.1 et 1.3.2). En moyenne la proportion de la distribution capturée est:

$$I = \int_{-\infty}^{+\infty} N(\hat{\alpha}; \alpha, \sigma_\alpha^2) \int_{-\infty}^{+\infty} N(\hat{\beta}; \beta, \sigma_\beta^2) \int_{\hat{m}-2\hat{\sigma}_e}^{\hat{m}+2\hat{\sigma}_e} N(x; m, v) dx d\hat{\beta} d\hat{\alpha}$$

où $N(x; \mu, \sigma^2)$ est une densité normale avec moyenne μ et variance σ^2. Dans cette expression $m = X_0 e^{\alpha t}$ et $v = X_0 \frac{1}{\alpha\beta} e^{\alpha t}(e^{\alpha t} - 1)$. On utilisera, pour calculer σ_e, la valeur moyenne de l'écart-type:

$$\bar{\sigma}_{\hat{\alpha}} = \frac{\hat{\beta}}{\sqrt{E(N)}}$$

$$\bar{\sigma}_{\hat{\beta}} = \frac{1}{\sqrt{E(N)\hat{\beta}}}$$

avec:

$$E(N) = \int_0^{t^*} (\lambda + \mu) E(X(t)) dt = \frac{\lambda + \mu}{\lambda - \mu} X_0 (e^{(\lambda - \mu)t^*} - 1)$$

où t^* est la période de temps sur lequel les estimateurs sont évalués. Cette dernière équation nous donne la relation entre la qualité des estimateurs et la période d'estimation $[0,t^*]$. Par une méthode itérative, on peut déterminer la valeur de r, pour des valeurs de α,β, t, t^*, et X_0 donnés. (voir appendice B)

3.3 Sensibilité du paramètre r

Nous avons tout d'abord constaté que les valeurs de r tendent à être les mêmes pour des valeurs constantes du ratio t/t^*. De plus pour une valeur fixe de t/t^*, la valeur de r ne montre qu'une faible dépendance sur les valeurs de X_0, α, β. Autrement dit, de grande variation des valeurs de X_0, α ou β n'ont que d'effets très limités sur la valeur de r. Lorsque l'on trace un graphique de r en fonction t/t^* pour différentes valeurs de λ, μ, et t^*, on obtient une série de courbes très similaires. Il est alors intéressant de voir si on ne pourrait pas trouver une fonction qui serait une bonne approximation de toutes ces courbes. Puisque la valeur de r semble augmenter comme un logarithme, nous avons expérimenté avec plusieurs fonctions de la forme $\hat{r}(t/t^*) = ln(1 + p(t/t^*))$, $p(t/t^*)$ étant une fonction strictement croissante avec $p(0) = 0$. Nous avons trouvé que $p(t/t^*) = \sqrt{t/t^*}$ donnait de bons résultats. Nous avons testé cette fonction avec un grand nombre de combinaisons de X_0,λ,μ et t^*. Dans tous les cas, $\hat{r}(t) = ln(1 + \sqrt{t/t^*})$ reste une bonne approximation de r. Les figures 3.2 à 3.6 sont des exemples de valeurs de r calculées par la méthode itérative et comparées avec la fonction $\hat{r}(t/t^*)$. Dans chacune de ces figures, les points sont les valeurs calculées de r et la courbe représente la fonction $\hat{r}(t/t^*)$.

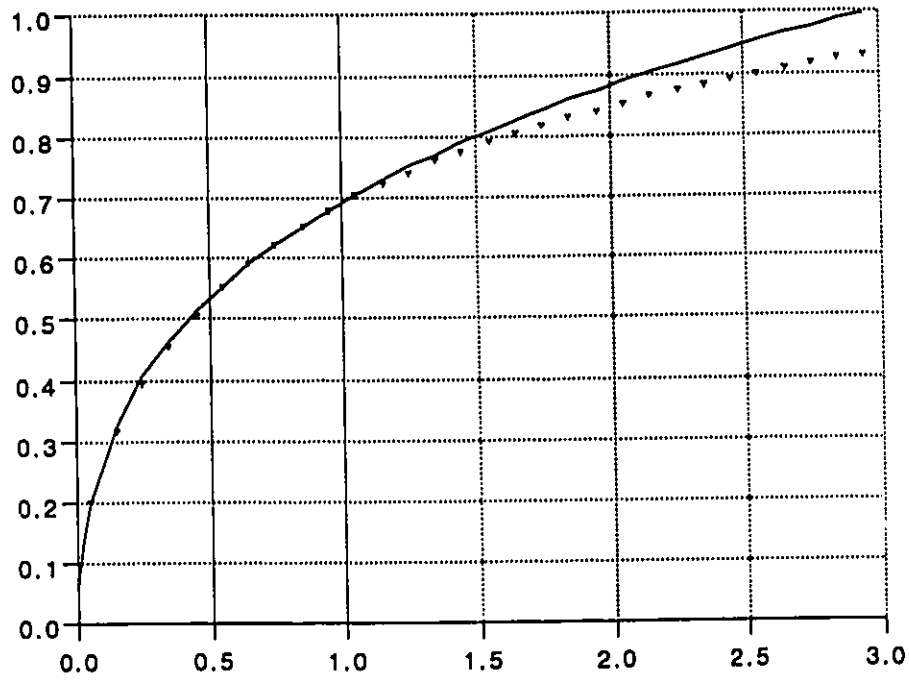

Figure 3.2: Paramètre r versus t/t^*: exemple 1

$\lambda = 3.5 \times 10^{-3}[jours]^{-1}$, $\mu = 2.1 \times 10^{-3}[jours]^{-1}$,

$t^* = 200$ Jours, $X_0 = 400$

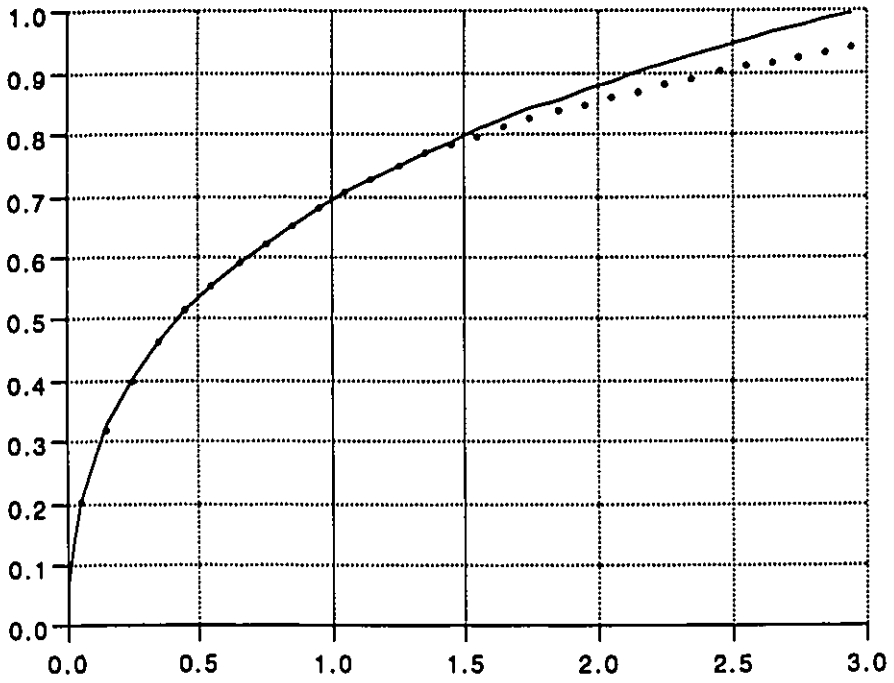

Figure 3.3: Paramètre r versus t/t^*: exemple 2

$\lambda = 1.5 \times 10^{-3}[jours]^{-1}$, $\mu = 1.1 \times 10^{-3}[jours]^{-1}$,

$t^* = 100$ Jours, $X_0 = 400$

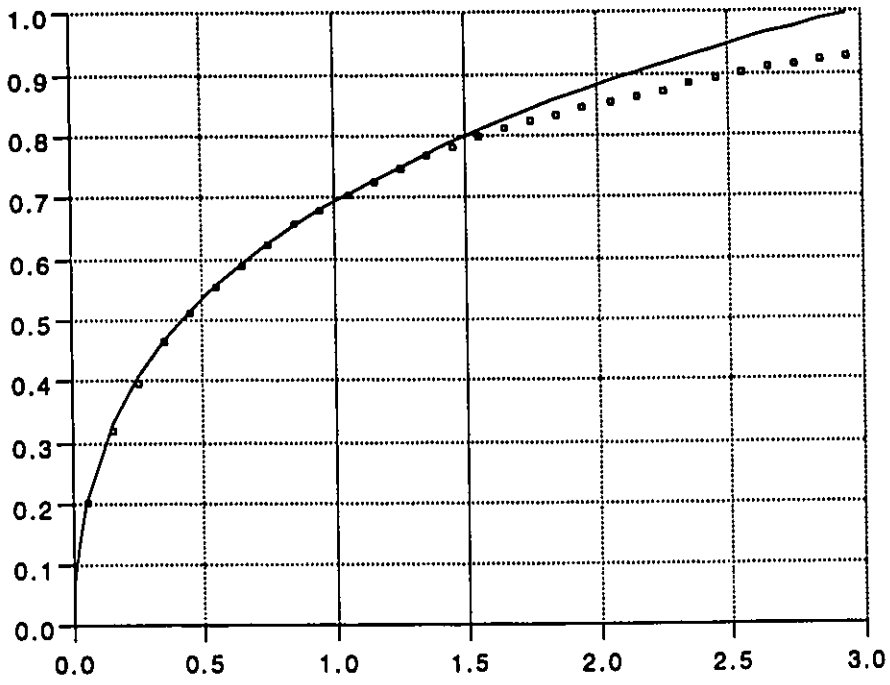

Figure 3.4: Paramètre r versus t/t^*: exemple 3

$\lambda = 5.0 \times 10^{-4}[jours]^{-1}$, $\mu = 2.4 \times 10^{-4}[jours]^{-1}$,

$t^* = 200$ Jours, $X_0 = 400$

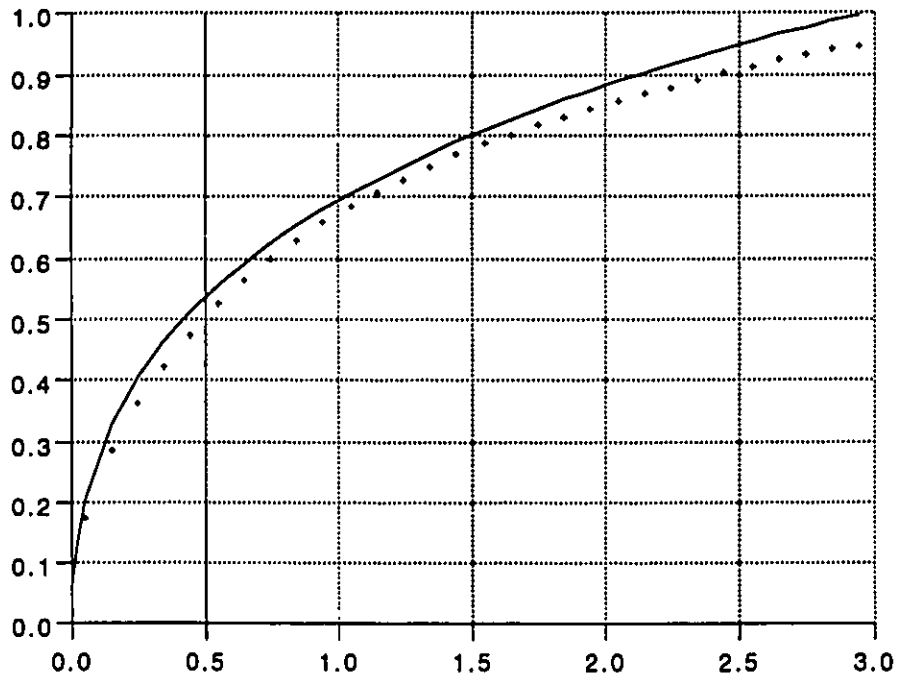

Figure 3.5: Paramètre r versus t/t^*: exemple 4

$\lambda = 1.2 \times 10^{-2}[jours]^{-1}$, $\mu = 7.2 \times 10^{-3}[jours]^{-1}$,

$t^* = 100$ Jours, $X_0 = 1000$

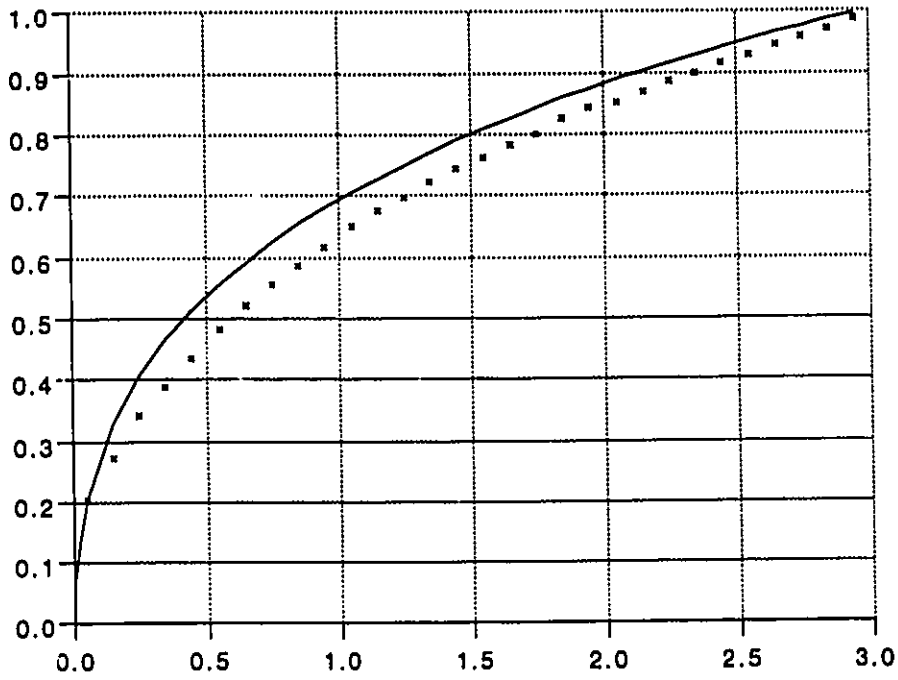

Figure 3.6: Paramètre r versus t/t^*: exemple 5

$\lambda = 2.7 \times 10^{-2}[jours]^{-1}$, $\mu = 1.4 \times 10^{-2}[jours]^{-1}$,

$t^* = 200$ Jours, $X_0 = 1000$

L'algorithme se résume donc à ceci:

$$\hat{\alpha} = \hat{\lambda} - \hat{\mu}$$

$$\hat{\beta} = \hat{\lambda} + \hat{\mu}$$

$$\hat{\alpha}_s(t) = \hat{\alpha} + ln(1 + \sqrt{t/t^*})\hat{\sigma}_{\hat{\alpha}}$$

$$\hat{\beta}_s(t) = \hat{\beta} - ln(1 + \sqrt{t/t^*})\hat{\sigma}_{\hat{\beta}}$$

$$\hat{m}(t) = X_0 e^{\hat{\alpha}t}$$

$$\hat{m}_s(t) = X_0 e^{t\hat{\alpha}_s(t)}$$

$$\hat{\sigma}_s(t) = X_0 \frac{1}{\hat{\alpha}_s(t)\hat{\beta}_s(t)} e^{t\hat{\alpha}_s(t)}(e^{t\hat{\alpha}_s(t)} - 1)$$

$$\hat{\sigma}_e(t) = \hat{\sigma}_s(t) + \frac{|\hat{m}_s(t) - \hat{m}(t)|}{2}$$

L'intervalle $(\hat{m}(t) - 2\hat{\sigma}_e(t), \hat{m}(t) + 2\hat{\sigma}_e(t))$ capture en moyenne 95% de la distribution.

3.4 Validation: Taux de succès des prédictions

Maintenant que nous disposons d'une méthode pour élargir l'intervalle de confiance, nous pouvons vérifier la validité de cette méthode en l'appliquant à notre simulation. Nous observons effectivement que les taux de succès des prédictions se maintiennent autour de 95%. Les figures 3.7- 3.9 peuvent être directement comparées aux figures 2.9- 2.11

Les figures 3.10- 3.12 peuvent être comparées aux figures 2.12- 2.14. Les points indique l'intervalle minimal capturant 95% des trajectoires. La ligne pleine indique la largeur de l'intervalle de confiance élargie pour deux écart-types. L'axe vertical est en écart-types de Nucho, avec $\lambda = \hat{\lambda}$ et $\mu = \hat{\mu}$. Notez comment l'élargissement de l'intervalle dépend de la qualité des estimateurs, c'est-à-dire de la longueur de la période d'estimation.

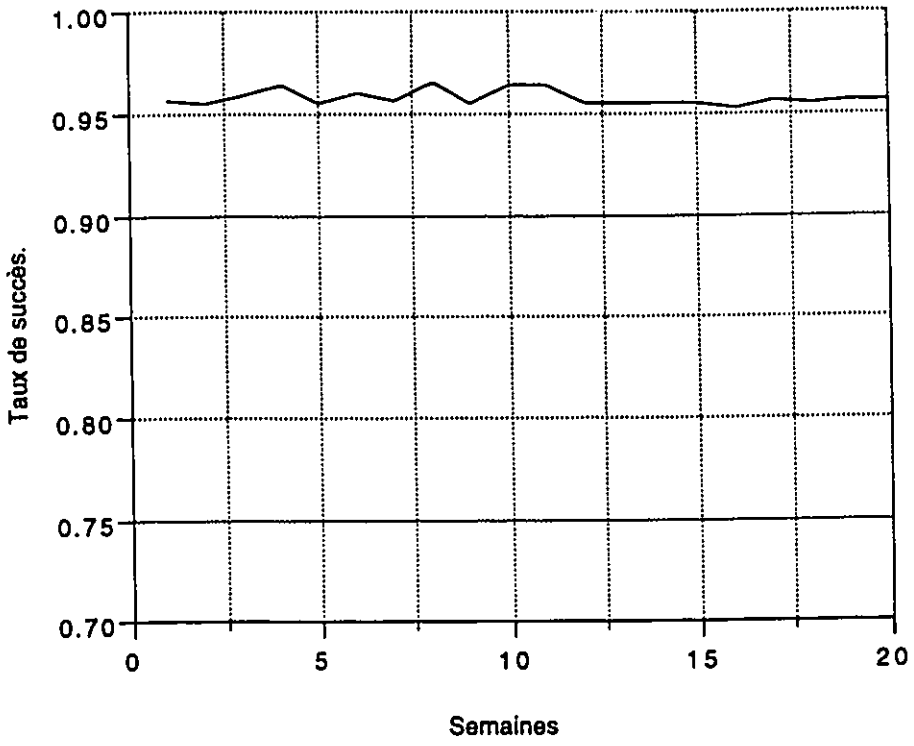

Figure 3.7: Taux de succès des prédictions: Estimation sur 10 semaines

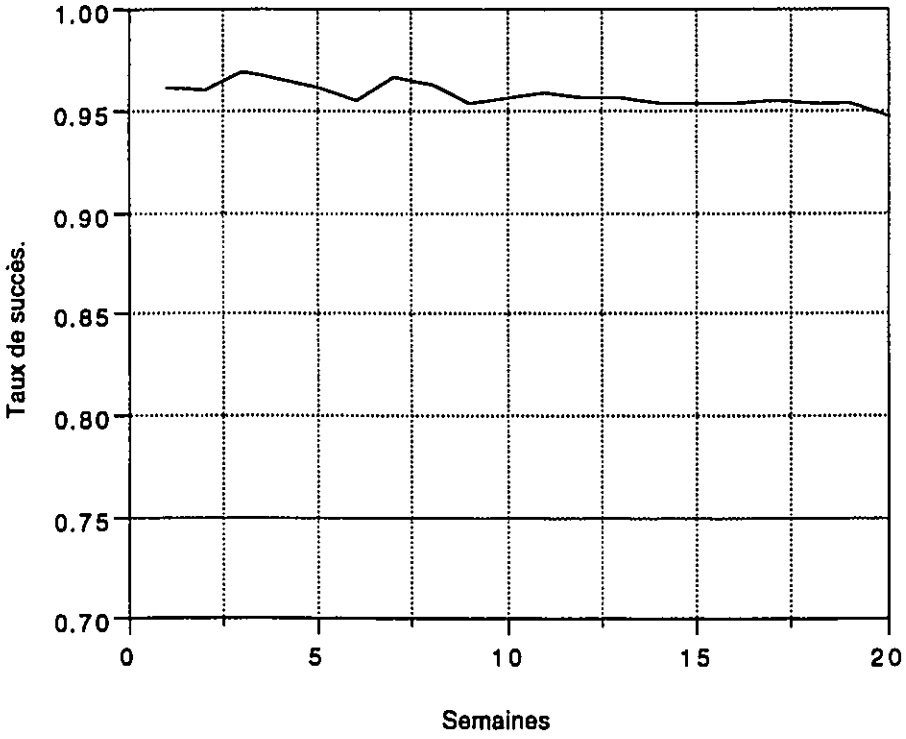

Figure 3.8: Taux de succès des prédictions: Estimation sur 20 semaines

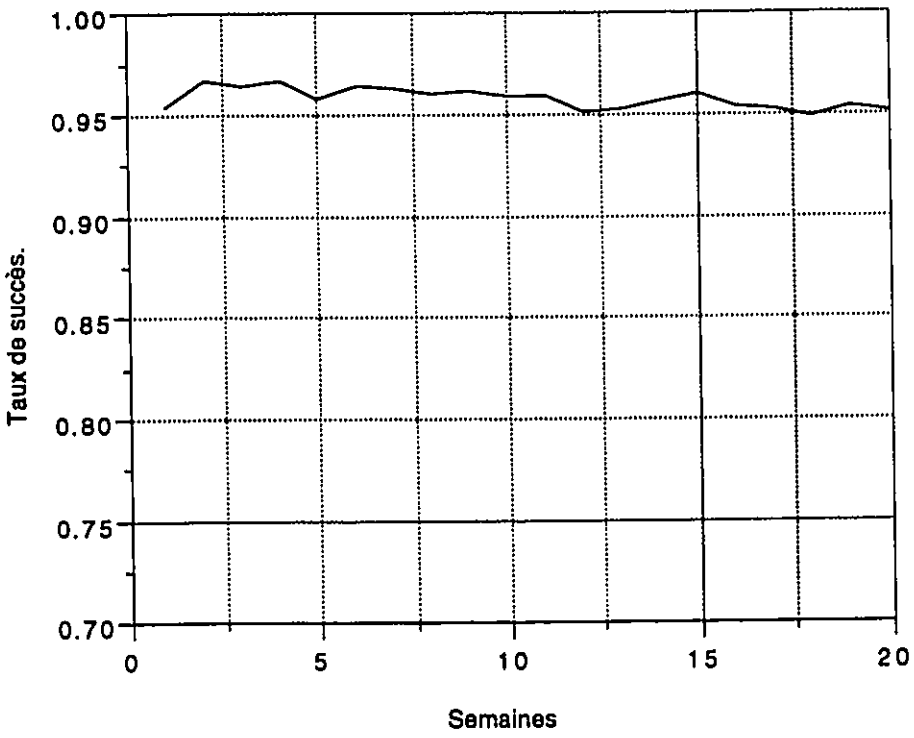

Figure 3.9: Taux de succès des prédictions: Estimation sur 40 semaines

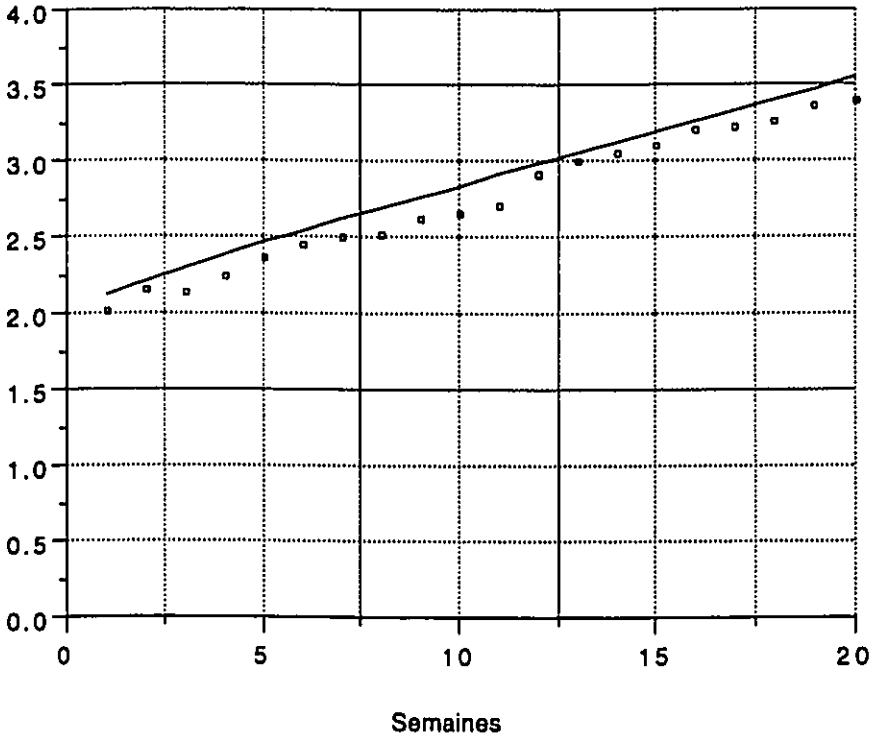

Figure 3.10: Intervalle de confiance de 95%: Estimation sur 10 semaines

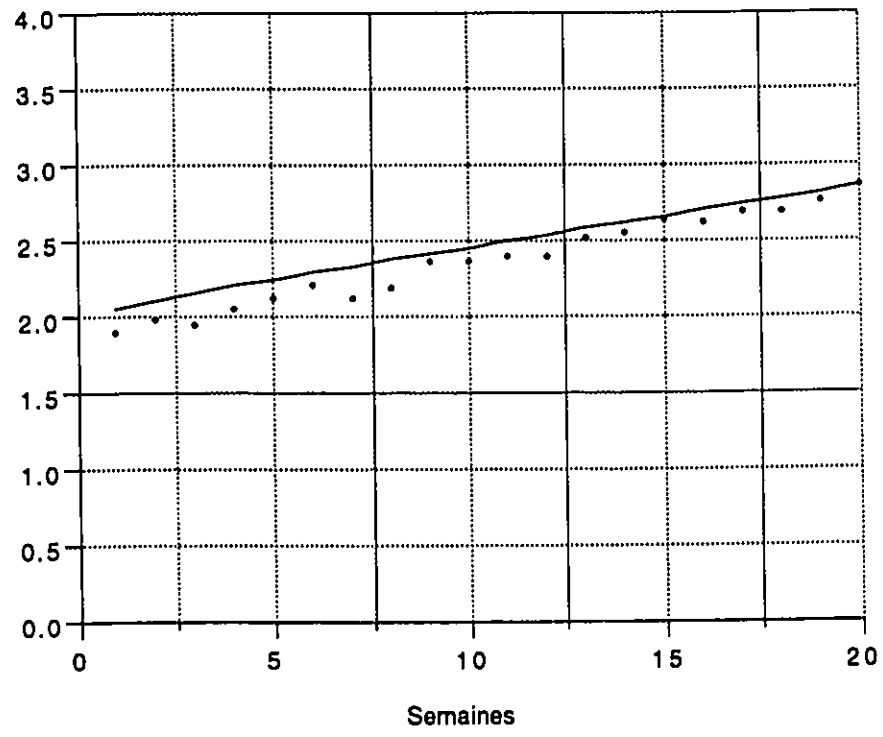

Figure 3.11: Intervalle de confiance de 95%: Estimation sur 20 semaines

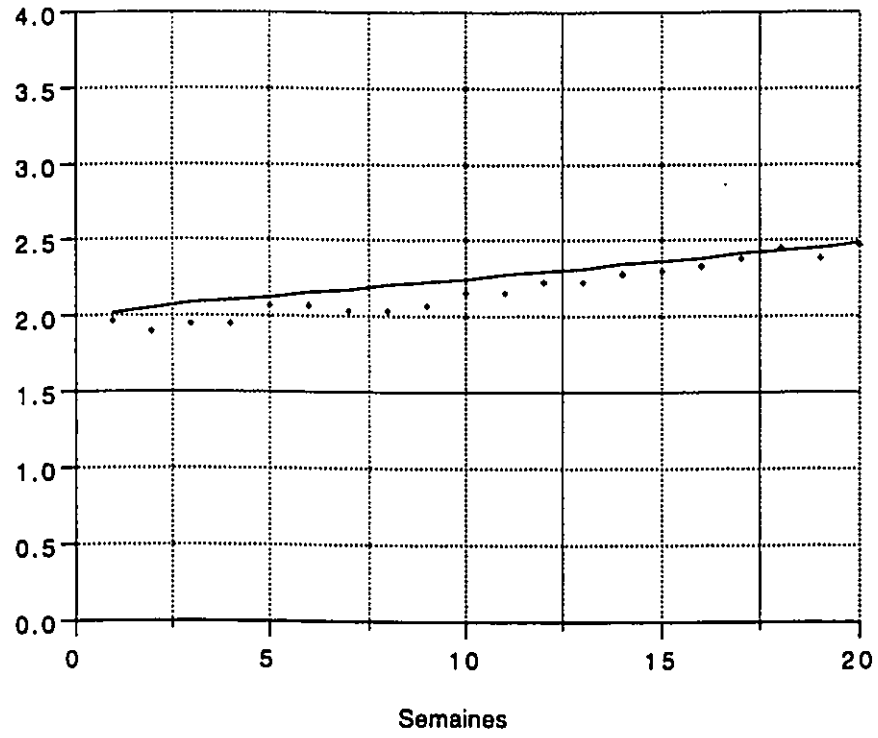

Figure 3.12: Intervalle de confiance de 95%: Estimation sur 40 semaines

3.5 Conclusion

Nous avons donc une méthode pour estimer les paramètres d'un processus de Kendall, et à partir de ces estimateurs et de leurs écart-types correspondants, nous pouvons trouver un intervalle de confiance de 95% pour nos prédictions. Dans le chapitre 4, nous allons utiliser ces méthodes avec des données réelles et nous tenterons de voir jusqu'à quel point ce modèle est applicable.

Chapitre 4

Application du modèle de Kendall

4.1 Les données

Dans ce chapitre, nous présentons les résultats de notre analyse s'appliquant à un réseau commercial de télécommunication. Le travail s'est effectué essentiellement sur le système d'analyse statistique SAS, dans les environnement CMS et Aix-Windows. Cette base de données consiste en une série d'enregistrements hebdomadaires des transactions qui se sont produites entre Février 1992 et Mars 1993. Chaque enregistrement contient un champ pour le client de départ, le client d'arrivée, le type de circuit impliqué, le genre de transaction (connection ou déconnection), ainsi que la grandeur et la date de cette transaction. L'identificateur du client est aussi divisé en 4 sous-champs, soit la province, la ville, le bâtiment, et un identificateur local du client. Il est donc possible de regrouper les données à quatre niveaux différents, soit les niveaux client-client, bâtiment-bâtiment, ville-ville, province-province. On peut également faire des regroupements sur les types de circuits. Nous disposons aussi d'une liste de nombres en place au niveau client-client à la date du 15 Septembre 1992.

Dans un premier temps, nous avons procédé à un tri pour éliminer toutes les

transactions n'ayant pas de nombre en place correspondant au 15 Septembre 1992. Il est possible qu'une bonne partie de ces cas soit causée par une mauvaise identification du client puisqu'il y a une large fraction de nombres en place au 15 Septembre avec aucune transaction correspondante dans l'année 1992-93.

4.2 Niveau client-client

Pour chaque client-client nous avons reconstruit la trajectoire du nombre en place en fonction de la date. Nous avons observé qu'un grand nombre de ces trajectoires contenait très peu de données. Pour notre analyse au niveau client-client, nous n'avons considéré que les cas qui avait au minimum 20 transactions. La méthode d'analyse que nous avons employé est la suivante: Nous prenons l'intervalle de temps séparant la première transaction de la dernière transaction et nous coupons cet intervalle en deux partie égales. Nous nous servons des données appartenant à la première partie de l'intervalle pour évaluer les estimateurs. Nous utilisons des données de la seconde moitié de l'intervalle pour tester nos prédictions.

La première série de croquis est un échantillonnage parmis 237 graphiques que nous avons produits pour le niveau client-client. Les courbes en pointillé correspondent à la moyenne prédite et au bornes maximale et minimale de l'intervalle de confiance de 95%. Nous avons observé que les prédictions étaient plutôt bonnes pour un grand nombre de ces trajectoires. Cependant nous constatons que dans plusieurs cas nos prédictions échouent complètement. Il semble y avoir deux causes principales d'échecs des prédictions. Il y a les sauts de la demandes et les changement de tendance des fluctuations. Par "saut" nous entendons une transaction d'un ordre de grandeur nettement plus élevé que la moyenne. On s'est aperçu que lorsqu'un saut se produisait dans la période d'estimation ceci avait des conséquences souvent

désastreuses pour nos prédictions. Ceci est illustré par l'exemple de la figure 4.1. On voit à quel point la tendance de nos prédictions peut être affectée par un seul saut. Pour régler ce problème nous avons modifié notre algorithme pour supprimer les sauts dans la période d'estimation. Lorsque la grandeur d'une transaction excède une certaine limite, nous supprimons la transaction du calculs des estimateurs. Une limite de deux écarts-types pour la grandeur de la transaction nous a semblé un critère adéquat. L'effet de cette suppression est évident si on compare les figures 4.1 et 4.3 aux figures 4.2 et 4.4. Globalement on observe une nette amélioration du taux de succès de nos prédictions lorsque cette suppression est utilisée.

Les changement de tendances ne sont pas prévu par le modèle de Kendall. Ceci impliquerait que les paramètres λ et μ changent de valeurs avec le temps. Nos méthodes d'estimation de ces paramètres ne sont plus valides dans un tel cas. Cependant si les paramètres restent relativement fixes pendant une certaine période de temps puis changent brusquement de valeurs (certains des graphiques suggèrent exactement cela), alors le modèle de Kendall pourrait toujours être applicable. Le problème serait alors de trouver une période de temps adéquate pour calculer les estimateurs, peut-être à l'aide d'un test de statistique non-paramétrique (McDonald (1991), Lehmann(1951)). Nous n'étudierons pas cette question dans cette thèse. Ainsi donc malgré les sauts et les changements de tendance, nous arrivons tout de même à un taux de succès global de 77% pour nos prédictions.

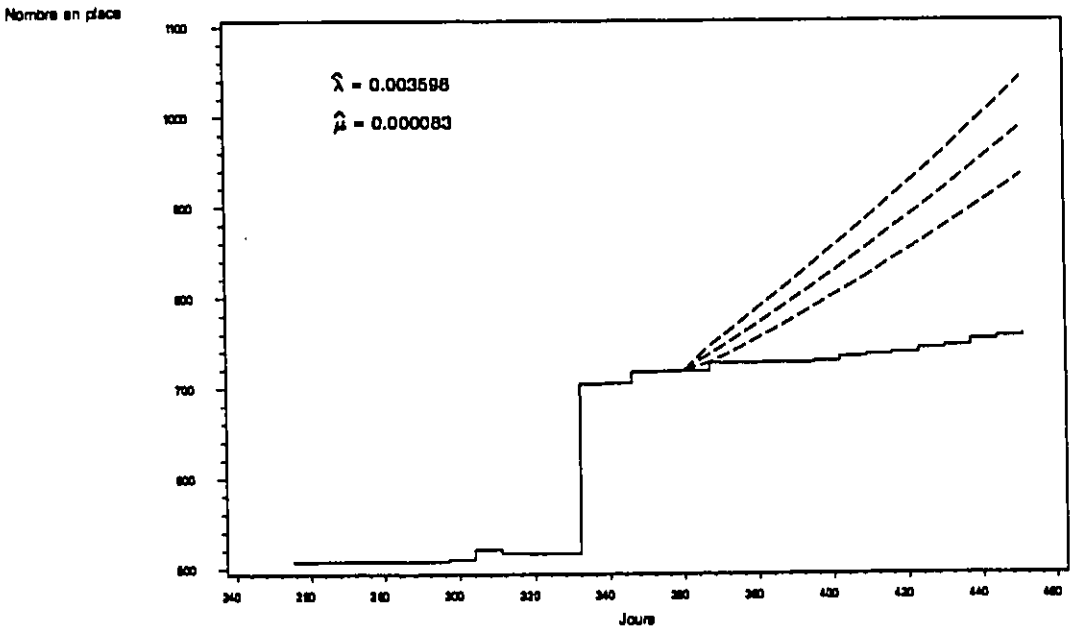

Figure 4.1: Niveau client-client: Prédiction sans suppression de saut (A)

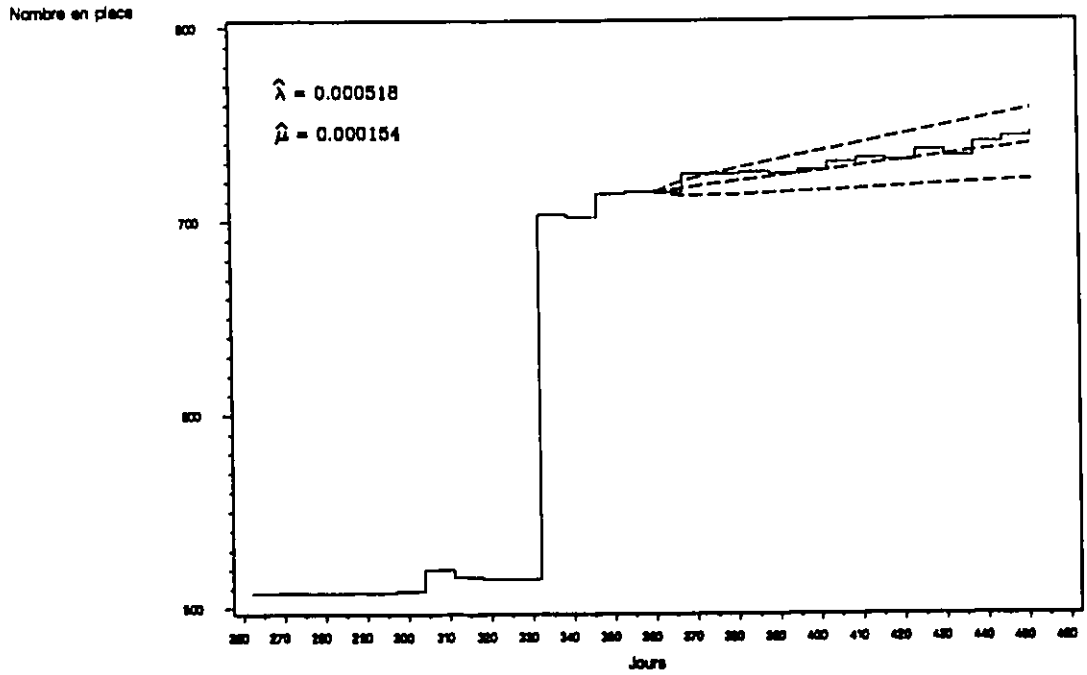

Figure 4.2: Niveau client-client: Prédiction avec suppression de saut (A)

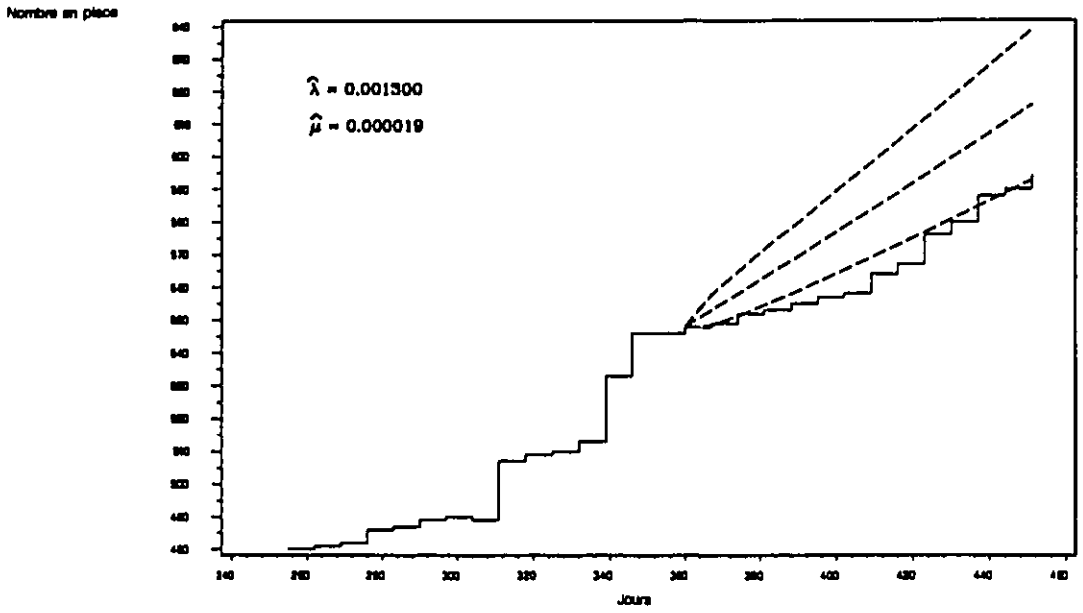

Figure 4.3: Niveau client-client: Prédiction sans suppression de saut (B)

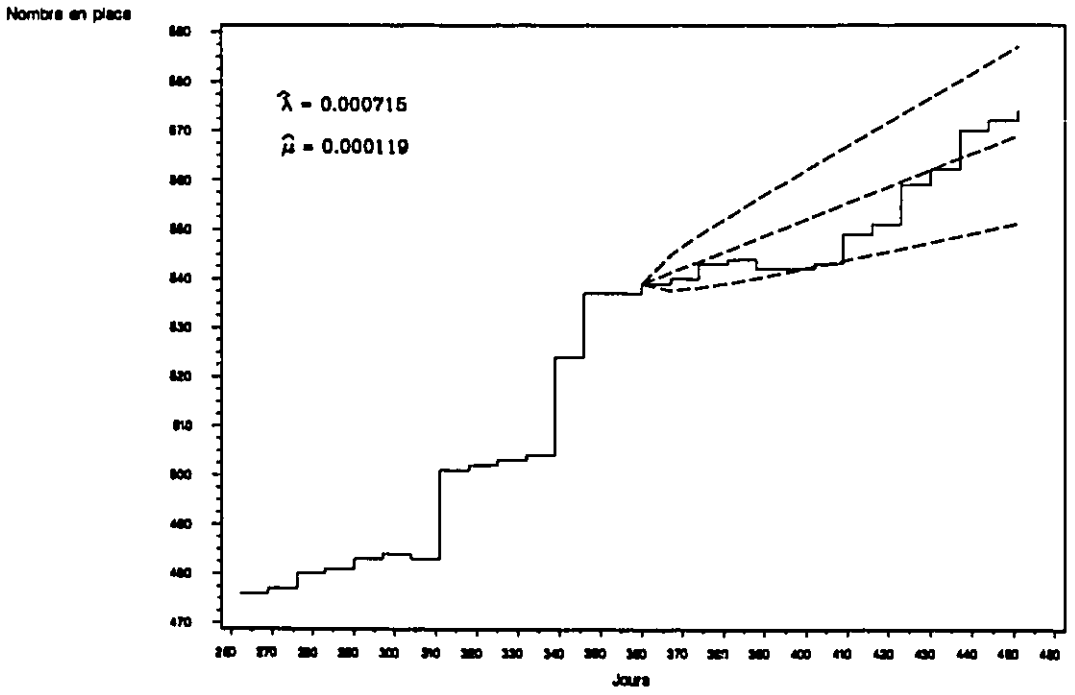

Figure 4.4: Niveau client-client: Prédiction avec suppression de saut (B)

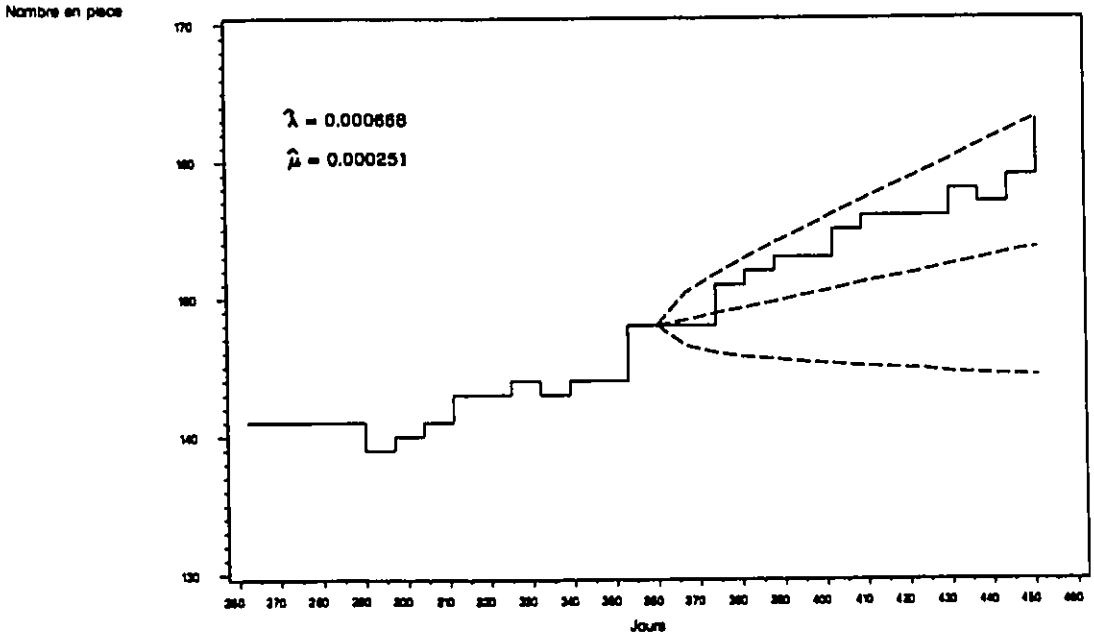

Figure 4.5: Prédiction au niveau client-client: exemple 1

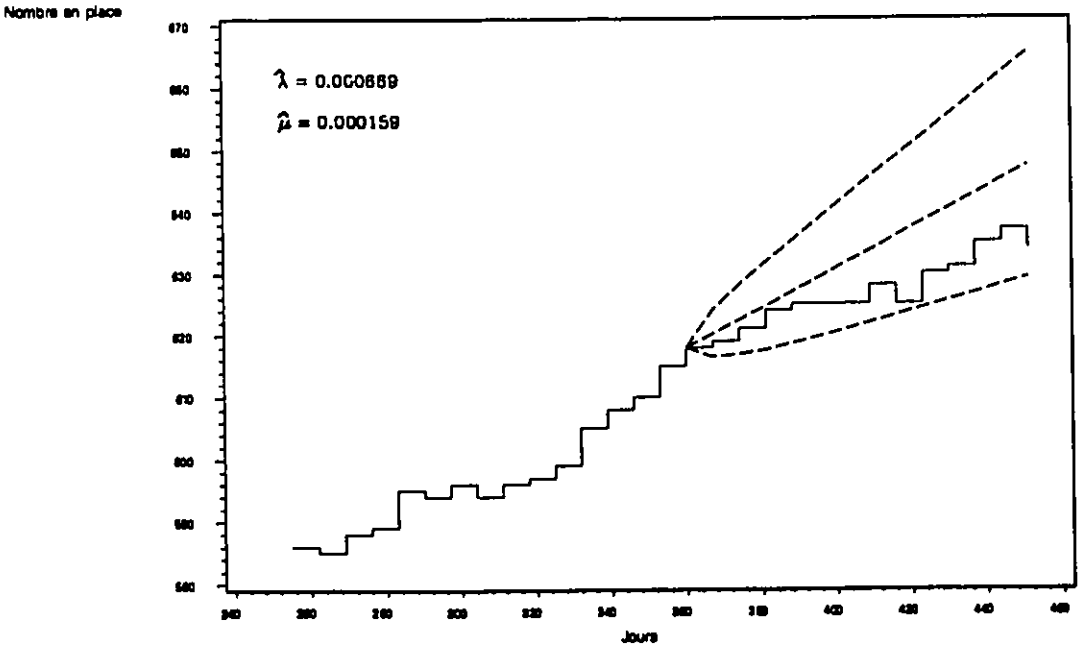

Figure 4.6: Prédiction au niveau client-client: exemple 2

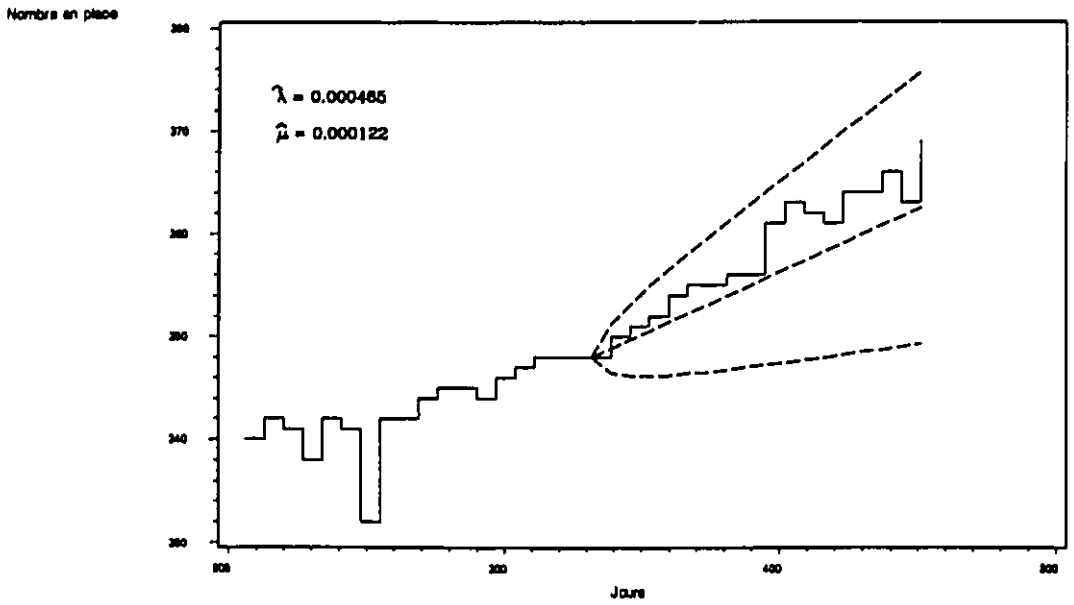

Figure 4.7: Prédiction au niveau client-client: exemple 3

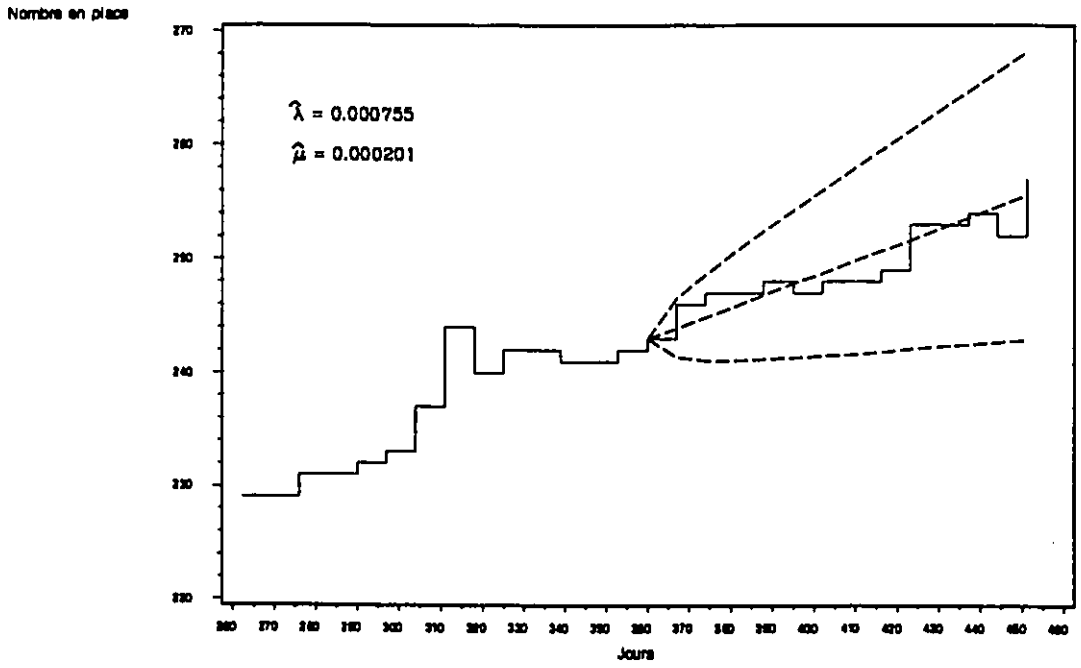

Figure 4.8: Prédiction au niveau client-client: exemple 4

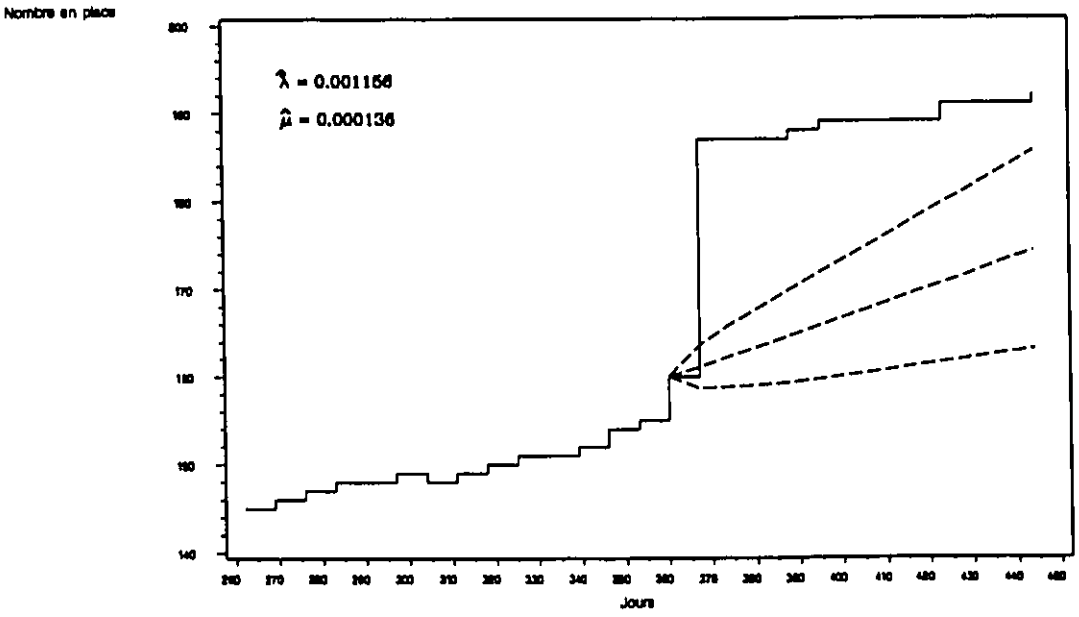

Figure 4.9: Prédiction au niveau client-client: exemple 5

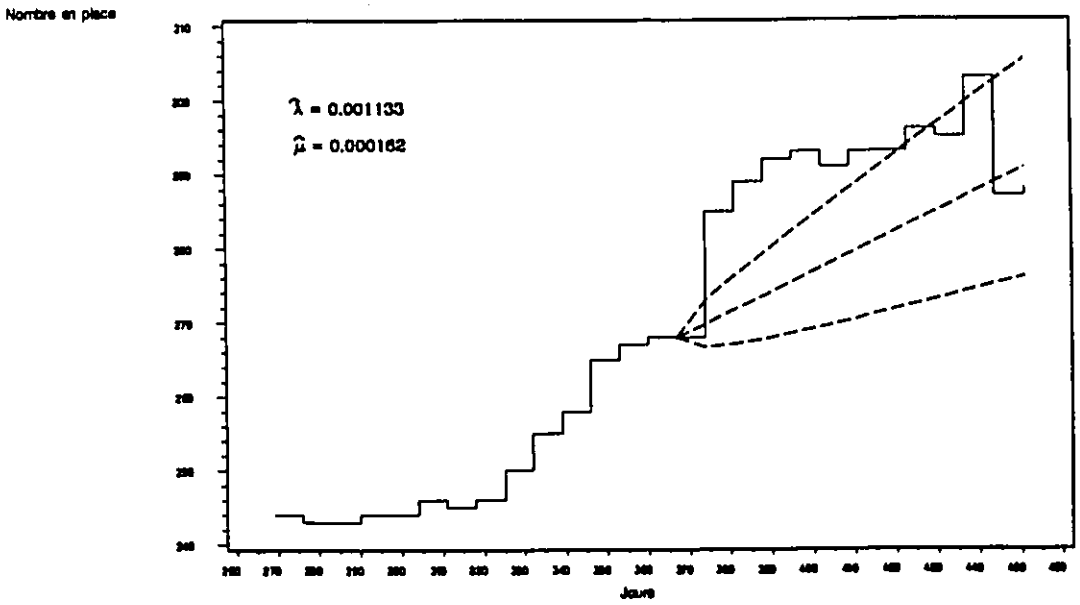

Figure 4.10: Prédiction au niveau client-client: exemple 6

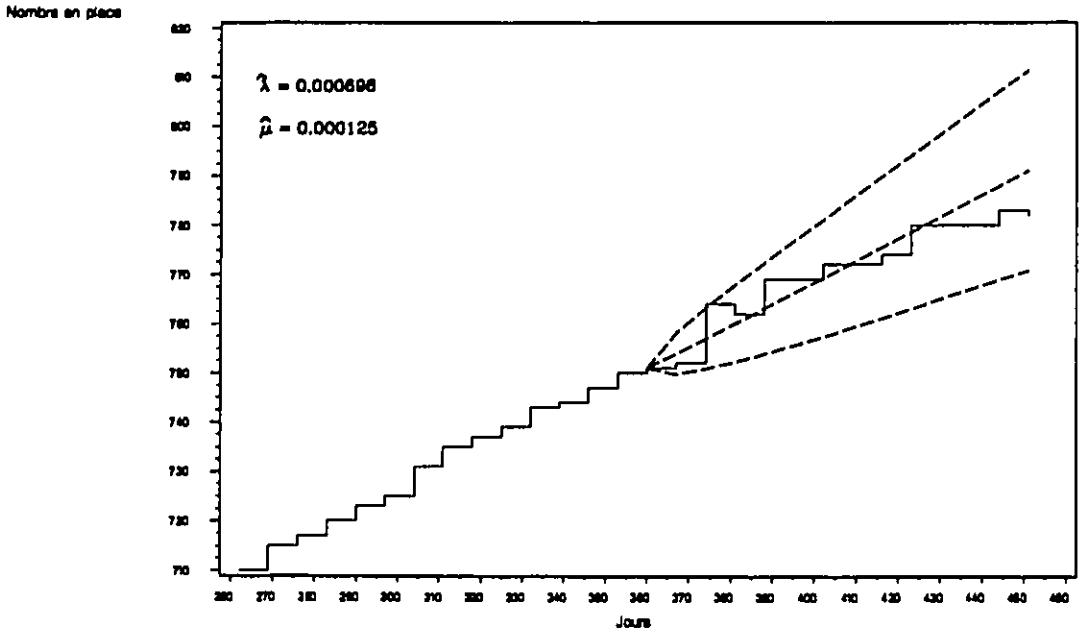

Figure 4.11: Prédiction au niveau client-client: exemple 7

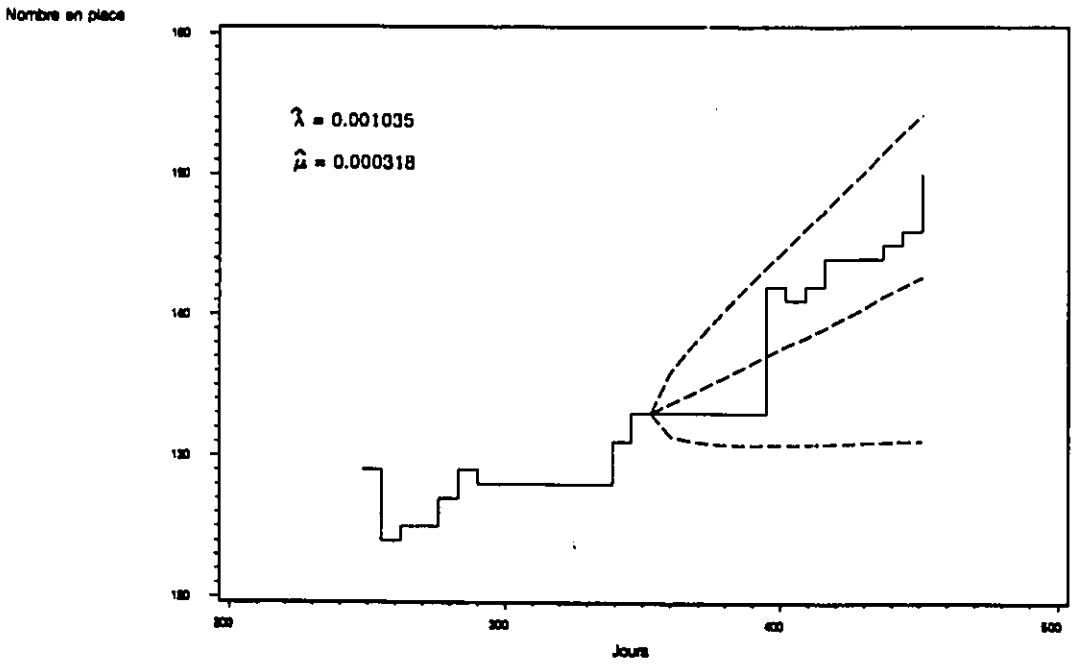

Figure 4.12: Prédiction au niveau client-client: exemple 8

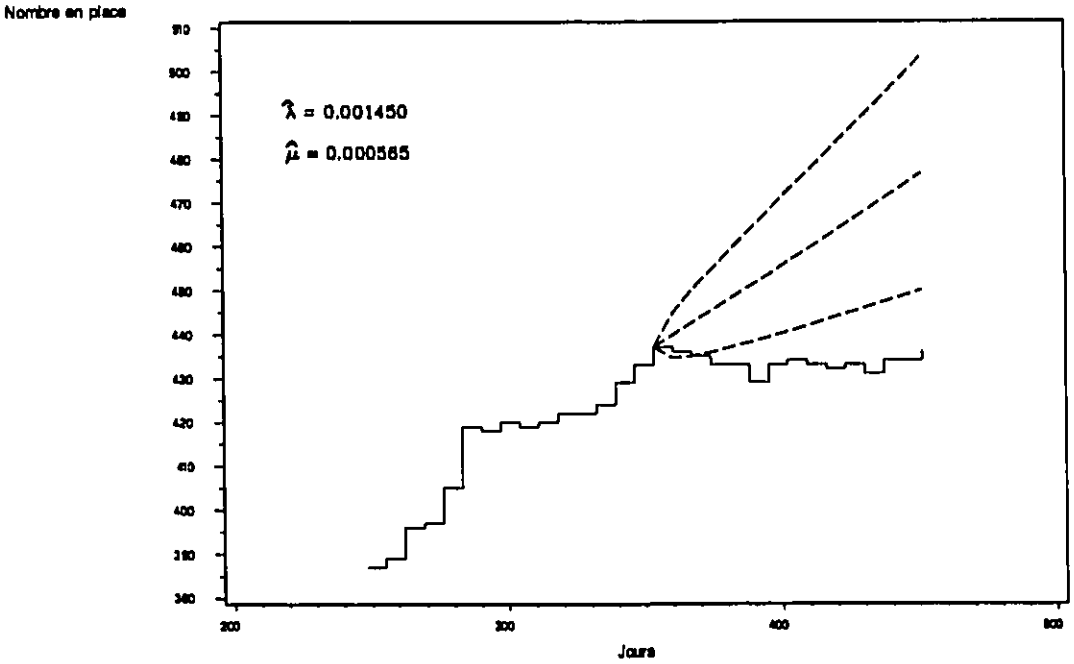

Figure 4.13: Prédiction au niveau client-client: exemple 9

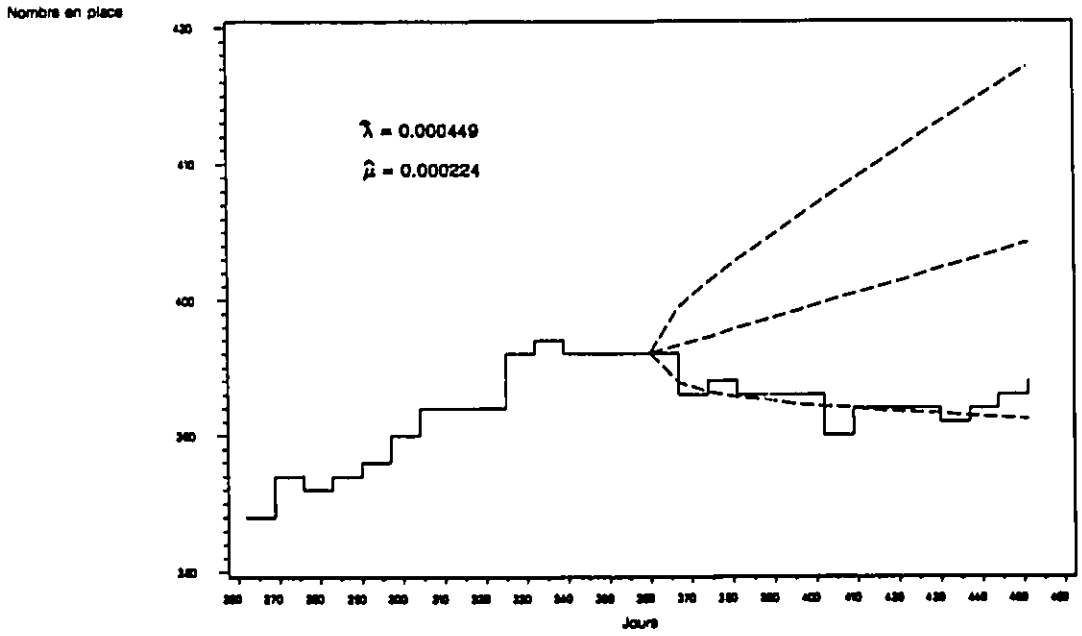

Figure 4.14: Prédiction au niveau client-client: exemple 10

4.3 Niveaux de regroupement supérieurs

Dans la section précedente nous n'avons utilisé que la fraction des clients qui avaient générés au moins 20 transactions sur la période étudiée. En regroupant les données au niveaux bâtiment-bâtiment et ville-ville, nous pouvons utiliser les données qui n'ont pas encore été analysées tout en utilisant le même critère d'un minimum de 20 transactions pour chaque niveaux. Les résultats à première vue sont peu encourageants. Le taux de succès des prédictions tend à diminuer avec un niveau croissant de regroupement. Pour illustrer la qualité globale de nos prédictions à chaque niveau, nous utilisons $\hat{X} = (X_{observé} - X_{prédit})/\hat{\sigma}_e$. La distribution de $\hat{X}$ autour de 0 est une mesure de la dispersion des observations par rapport à nos prédictions. Les histogrammes de $\hat{X}$ des figures 4.15, 4.16 et 4.17 illustrent la détérioration de nos prédictions. (Notez que nous regroupons les clients mais pas les types de circuits)

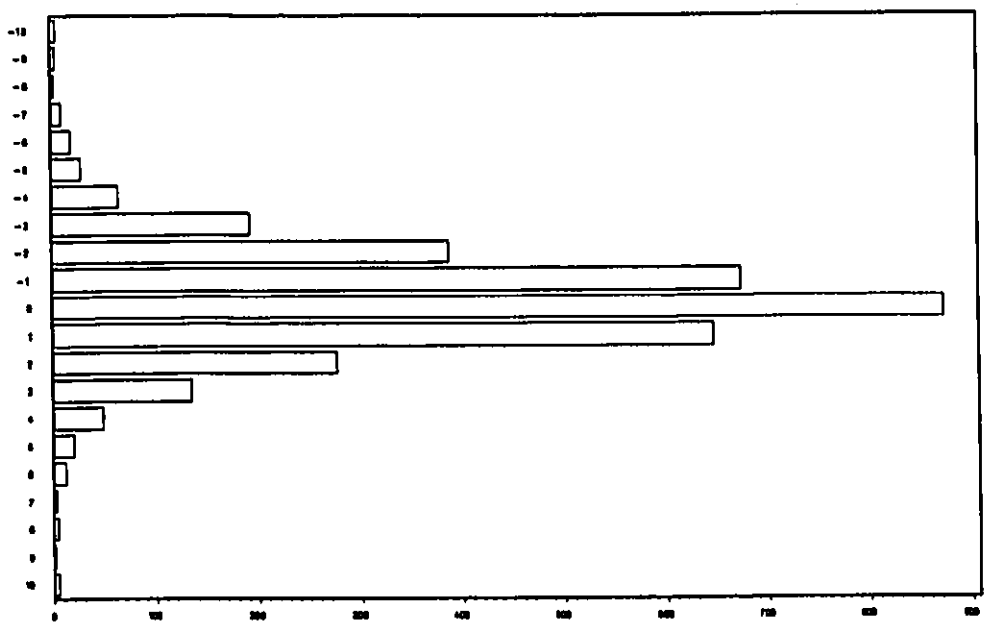

Figure 4.15: Qualité des prédictions aux niveau client-client

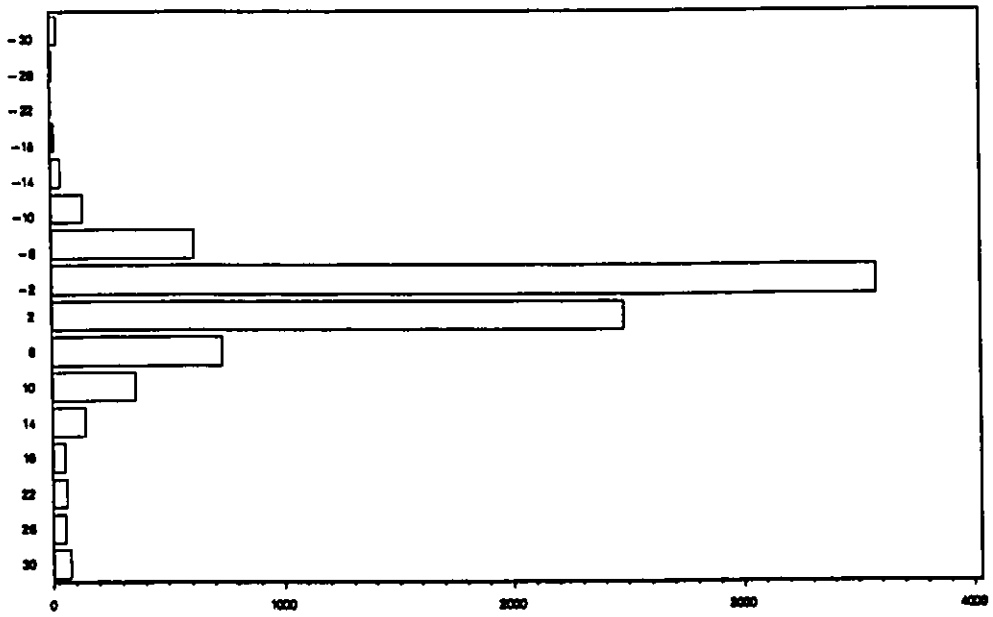

Figure 4.16: Qualité des prédictions aux niveau ville-ville

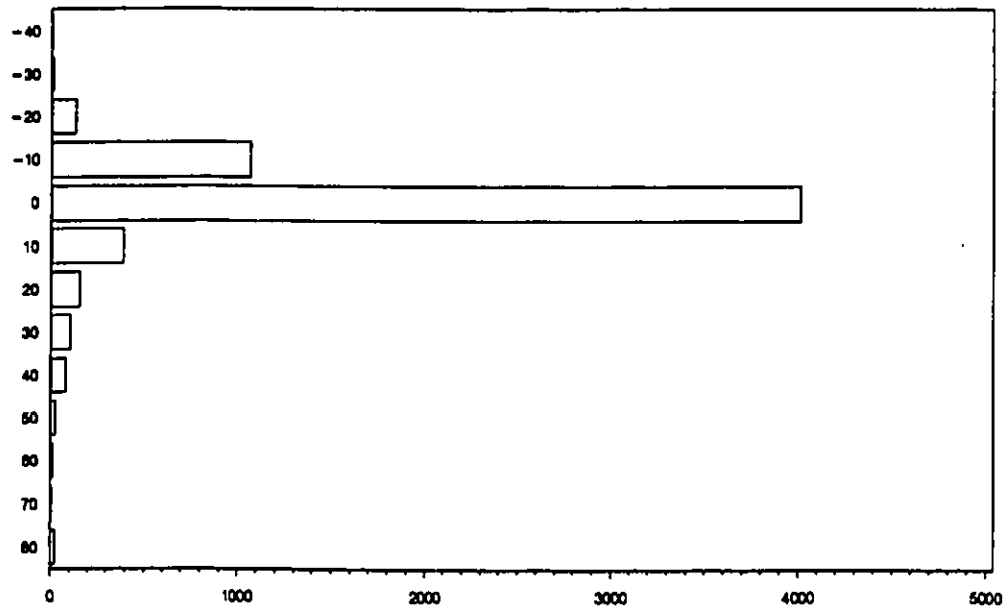

Figure 4.17: Qualité des prédictions aux niveau province-province

4.4 Regroupement au niveau ville-ville

Nous avons constaté que très souvent les fluctuations du nombre en place "intra-urbain", c'est-à-dire la somme du nombre en place de tout les clients à l'intérieur d'une même ville, change brusquement de tendance à partir d'une certaine époque de l'année. Ce phénomène est observé pour un grand nombre de villes. Ce phénomène est lié au clients ayant peu de transactions puisqu'il s'observe beaucoup moins au niveau client-client, c'est-à-dire les clients avec 20 transactions ou plus. Nous constatons que ce phénomène est aussi lié au type de circuit (voir section suivante). Ceci pourrait refléter un changement dans la politique du donneur de service plutôt que qu'un réel changement dans la demande des clients. De plus on observe très peu ce phénomème lorsque on regarde les fluctuations des nombre en place "inter-urbain", c'est-à-dire le nombre en place total entre les clients de deux villes différentes. Les figures 4.18 à 4.20 montrent des exemples typiques de nombres en place au niveau ville-ville pour une seule ville. La figure 4.21 montre le nombre en place pour la demande à partir d'une ville vers des villes différentes.

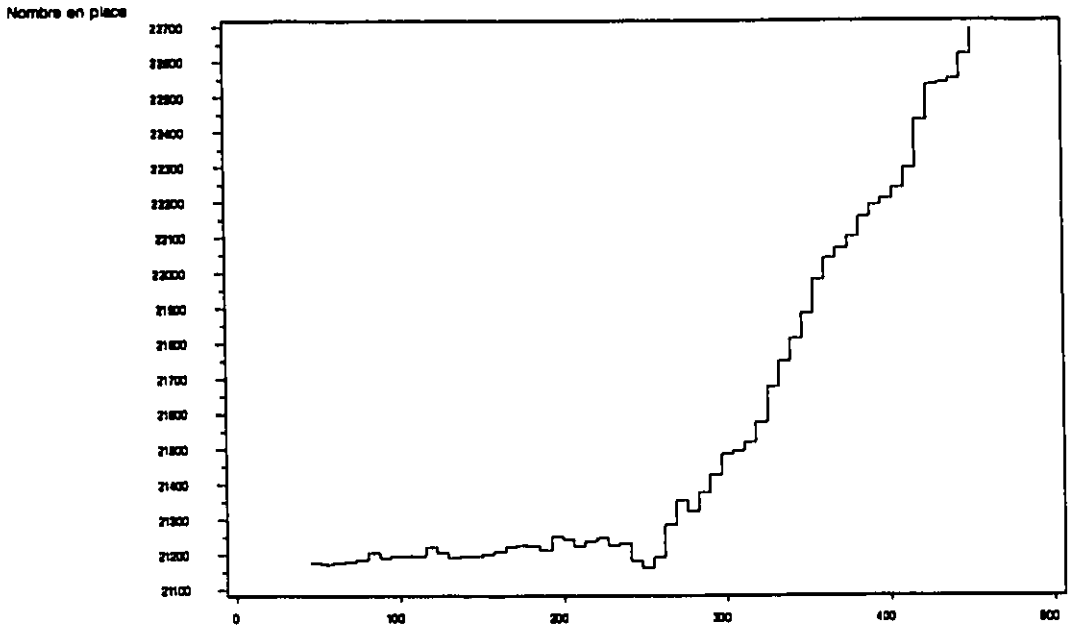

Figure 4.18: Prédiction au niveau ville-ville: exemple 1

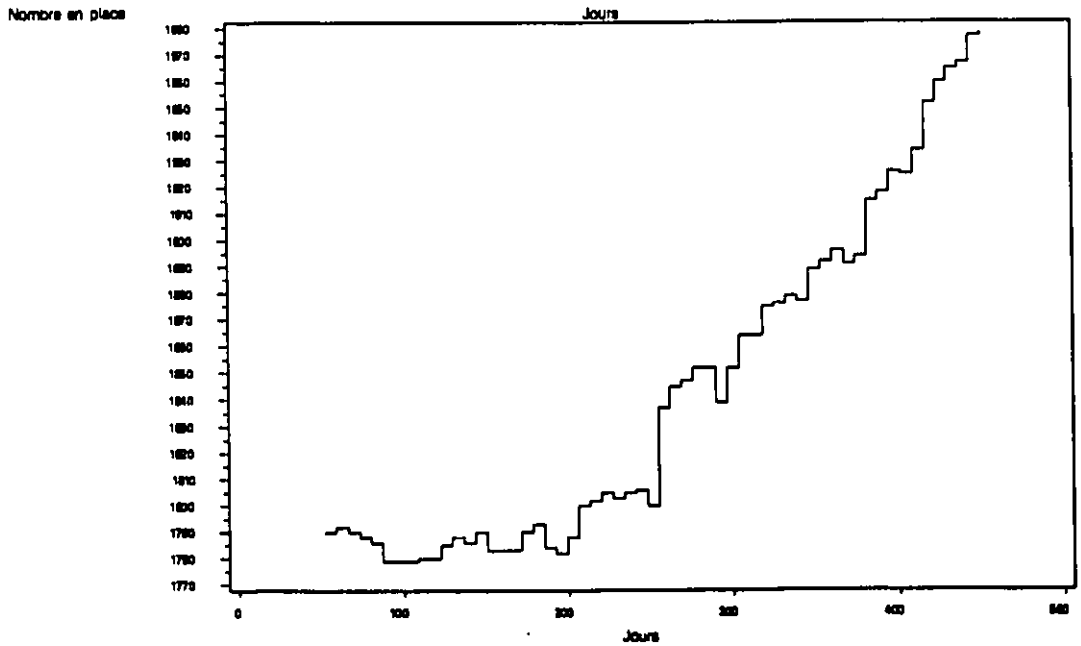

Figure 4.19: Prédiction au niveau ville-ville: exemple 2

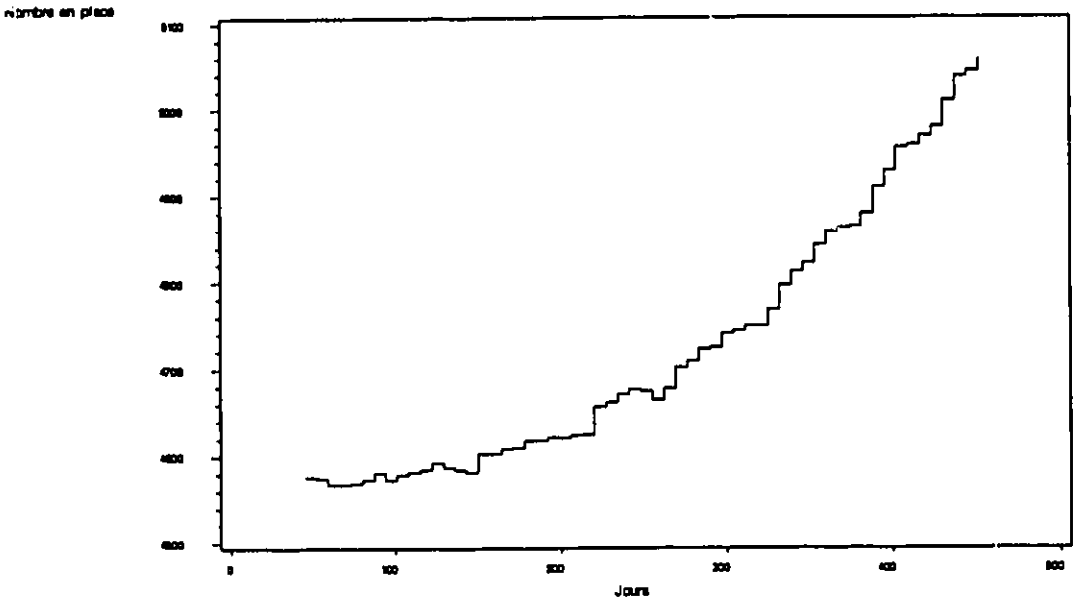

Figure 4.20: Prédiction au niveau ville-ville: exemple 3

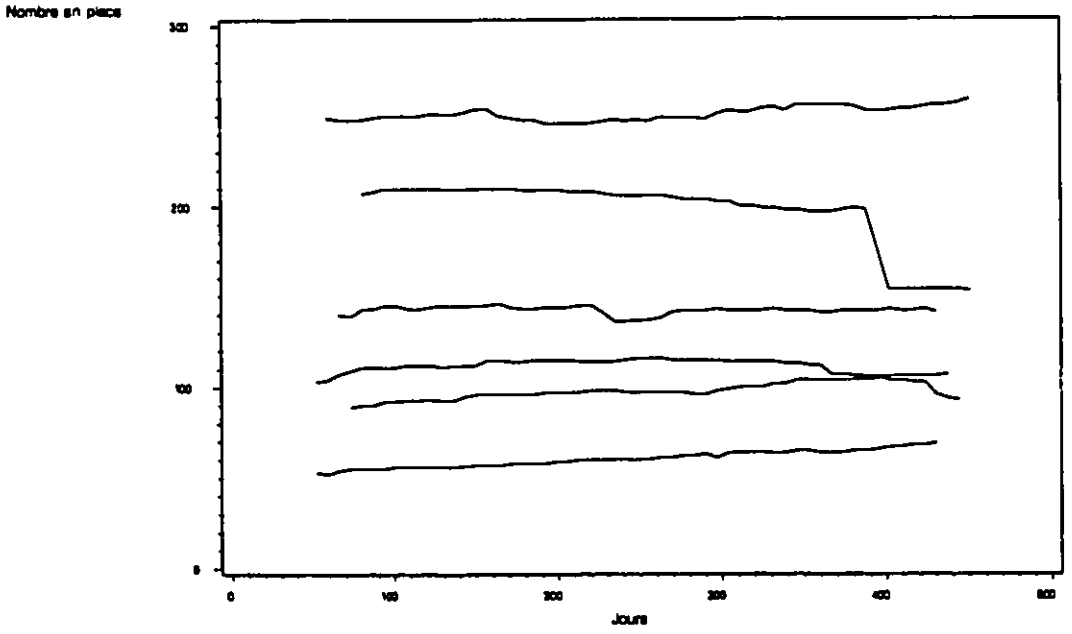

Figure 4.21: Prédiction au niveau ville-ville: exemple 4

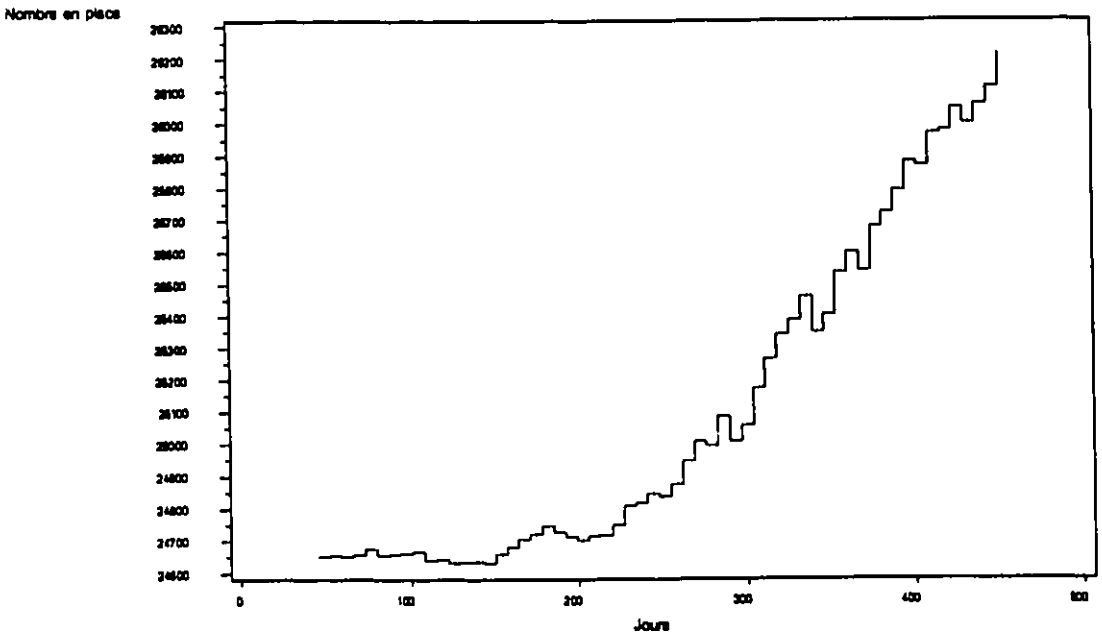

Figure 4.22: Regroupement par type de circuit: exemple 1

4.5 Regroupement sur les types de circuits

En regroupant les données de tout les clients sur les type de circuits nous avons observé des phénomènes similaires au niveau ville-ville. Dans certains cas on observe une croissance presque linéaire du nombre en place, dans d'autres cas un décroissance continuelle, et très souvent des changements brusques de tendances à des moments similaires de l'année. De fait il serait aisé de classifier les types de circuits selon les fluctuations de leur nombre en place. Encore un fois, les résultats dans beaucoup de cas semblent refléter une politique du donneur de service plutôt qu'un changement de tendance de la demande. Les figures 4.22 à 4.26 montrent des exemples typiques de nombres en place par circuit.

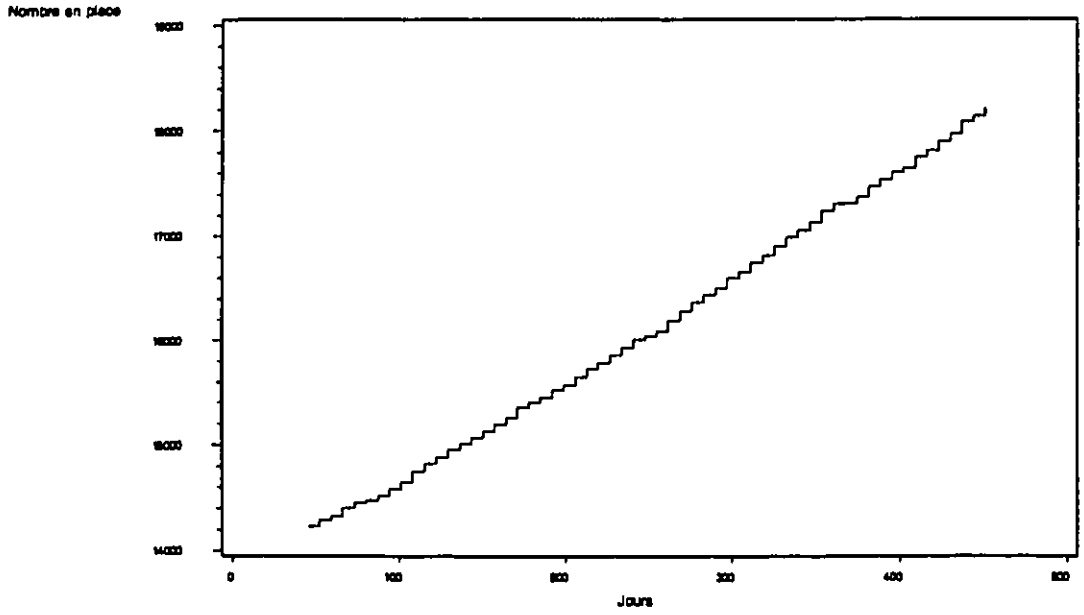

Figure 4.23: Regroupement par type de circuit: exemple 2

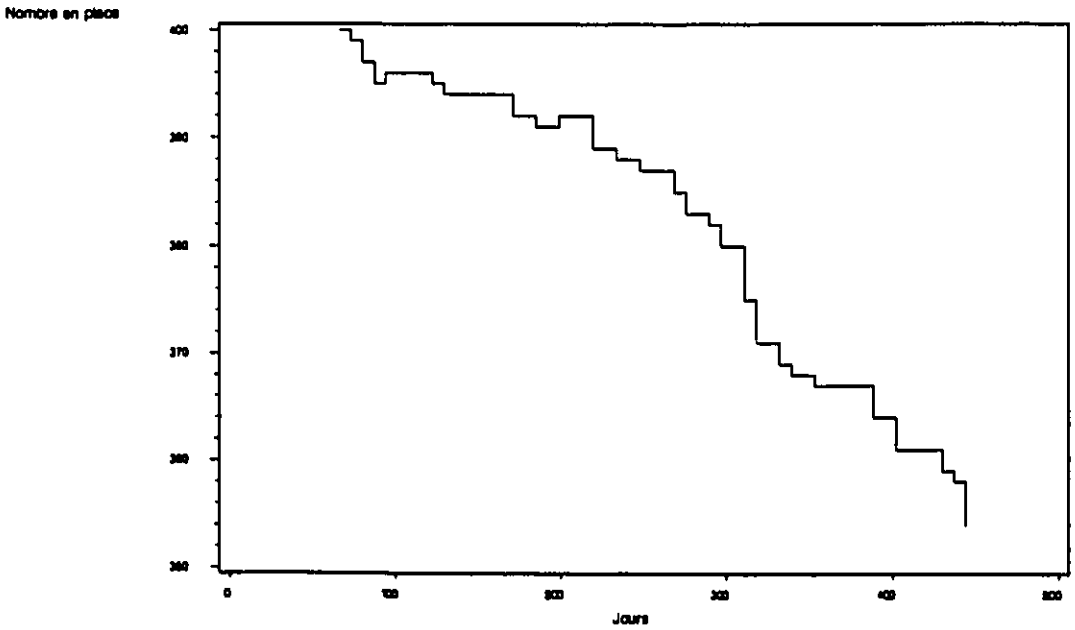

Figure 4.24: Regroupement par type de circuit: exemple 3

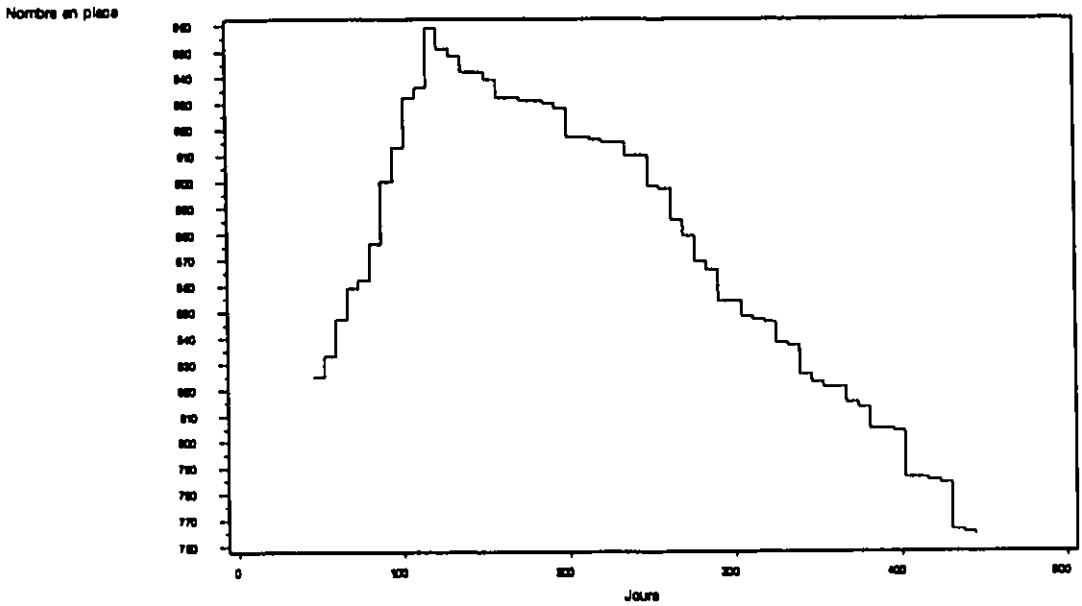

Figure 4.25: Regroupement par type de circuit: exemple 4

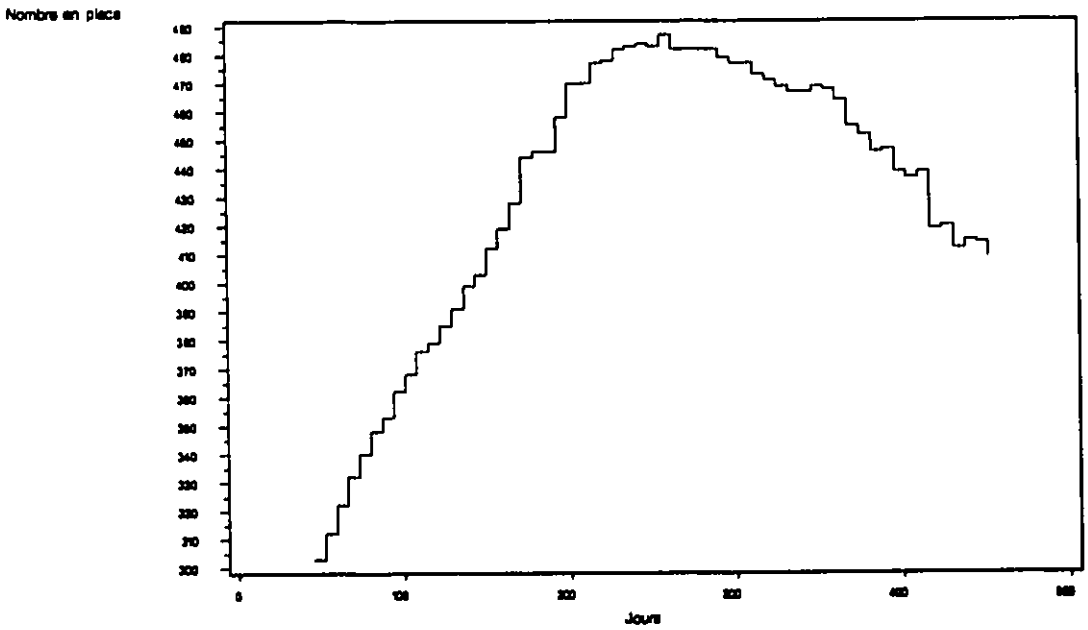

Figure 4.26: Regroupement par type de circuit: exemple 5

Chapitre 5

Conclusion

Dans cette thèse nous avons présenté le modèle de Kendall comme une représentation adéquate de la fluctuation de la demande entre deux points. Nous avons présenté une méthode d'estimation des paramètres d'un processus de Kendall. Nous avons utilisé une simulation pour tester cette méthode d'estimation. Nous avons observé que si nous utilisons les formules de Nucho pour la variance avec ces estimateurs, nous ne pouvons pas obtenir 95% de succès pour nos prédictions. Nous avons développé un algorithme qui permet d'étendre ces intervalles afin d'obtenir un intervalle de confiance de 95%. Nous avons analysé les données d'un réseau commercial. Nous obtenons un taux de succès de 77% au niveau client-client, pour les clients avec plus de 20 transactions. Les sauts et les changements de tendance de la demande contribuent à baisser le taux de succès de nos prédictions. Si un saut a eu lieu dans la période d'estimation, alors il est possible de supprimer ce saut si on utilise un critère adéquat. Un critère raisonnable est de supprimer toutes les transactions au-delà de deux écart-types.

Cependant cela ne règle pas le problème si le saut se produit dans la période de prédiction. En fait toute méthodes de prédictions échoueraient face à ce genre d'événement. Il se pourrait que les sauts soient en fait d'origine déterministique, c'est-à-dire qu'ils se produisent lors de situations bien définies. Nous pourrions alors développer une méthode pour "intégrer" les futurs sauts dans nos prédictions. Cet aspect du problème mériterait d'être étudié de plus près.

Le processus de Kendall ne prévoit pas les changements de tendances. Cependant un test statistique non-paramétrique pourrait être utilisé pour détecter un changement de tendance, permettant de choisir une période d'estimation plus approprié.

Nous avons constaté que si nous regroupons les données à des niveaux supérieurs, le taux de succès des prédictions tend à diminuer. Certains indices nous incitent à penser que des facteurs déterministiques, tel une politique du donneur de service, ont une influence non-négligeable à ces niveaux de regroupements.

Si tous ces facteurs peuvent être identifiés et intégrés au modèle nous pensons avoir un excellent outil de prédiction.

Appendice A

Programme de simulation

Le programme à été écrit en Think Pascal version 4.0, sur MacIntosh tournant sur le système 7. La présente version compilée requiert la présence d'une puce numérique. La mémoire requise par le programme dépend avant tout de la taille de la simulation. Une simulation demandant 1000 trajectoires générées sur une période de 365 jours requiert environ 400Ko de mémoire vive. Une allocation de mémoire de 1 Meg est donc généralement suffisante. Le programme est intégralement donné dans les sections suivantes. Les routines on été regroupées par thème.

- Section A.1 : Simulation. Calcul des estimateurs, prédictions et statistiques.
- Section A.2 : Declarations de types et variables globales. Fonctions utilitaires.
- Section A.3 : Routines graphiques.
- Section A.4 : Routines d'entrées / sorties.
- Section A.5 : Routines des fenêtres 'dialogue'.
- Section A.6 : Routines de gestion de l'interface.
- Section A.7 : Initialisation et programme principal.

A.1 Algorithmes de simulation

```
unit Simulation;
interface
 uses
  SANE, Global, Tfile, SimGraph, StatGraph, Estimators, Validation, Statistics;
 procedure KENDALL_PROCESS (X0: integer; lambda, mu: real);
implementation
{*********************************************************}
{ This function returns a random number with a Poisson distribution with mean 'lambda'  }
{ The function calculate the cumulative mass function 'cmf' (by summing the probability mass  }
{ function 'pmf') }
{ until it exceeds the 'test' value.  'test' is a pseudo-uniform random number generated over }
{ [0,1) }

function Poisson (lambda, test: real): integer;
  var
   cmf, pmf: real;
   x: integer;
 begin
  pmf := exp(-lambda);
  cmf := pmf;
  x := 0;
  while (cmf < test) do
   begin
    x := x + 1;
    pmf := pmf * lambda / x;
    cmf := cmf + pmf;
   end;
  Poisson := x;
 end;

{*********************************************************}
{ This procedure generate the trajectory of a Kendall process.  }
{ 'actuals' is the initial number at the start of the trajectory. }
{ 'lambda' and 'mu' are the parameters of the process. }
{  'x','xin','xout' are arrays where the daily history is recorded. 'x' contains the number of }
{ actuals }
{ 'xin' the daily number of connections, and 'xout' the daily number of disconnections. }

 procedure KENDALL_SIMULATION (x, xin, xout: LongIntPtr; actuals: LongInt; lambda, mu: real);
```

```
 const
  RANDOM_MAX = 32768.0;
 var
  t, A, D: Integer;
  JumpTime: Integer;
  value: real;
begin
 randSeed := TickCount;
 WriteLongInt(x, 1, actuals);

 for t := 2 to N_DAYS do
  begin

{ 'value' is a pseudo-uniform random number used to generate a Poisson }
{ 'A' and 'D' are Poisson random numbers with mean 'lambda*actuals' and 'mu*actuals' }

   value := (Random + 32768.0) / 65536.0;
   A := Poisson(lambda * actuals, value);
   value := (Random + 32768.0) / 65536.0;
   D := Poisson(mu * actuals, value);

{ Use 'A' and 'D' to update the value of 'actuals' }

   actuals := actuals + LongInt(A - D);

   if actuals < 1 then
    actuals := 1;

{ Write the values of 'A','D' and 'actuals' to the arrays 'xin', 'xout' and 'x'}

   WriteLongInt(xin, t, LongInt(A));
   WriteLongInt(xout, t, LongInt(D));
   WriteLongInt(x, t, actuals);
  end;
end;

{*************************************************************}
{ This is the main simulation procedure  }

procedure KENDALL_PROCESS (X0: integer; lambda, mu: real);
 var
  x, xin, xout: LongIntPtr;
  weekAct, weekIn, weekOut: LongIntPtr;
  i, t, FileNo, errCode: Integer;
  lambda1, mu1, s_alpha, s_beta: real;
begin

{ Prepare graphic window if graphical output has been requested }
 if SimOutput then
```

```
  SimulationGraph;

{ Allocate the memory for storing the data for 1 trajectory }
  x := LongIntPtr(NewPtr(Sizeof(LongInt) * N_DAYS));
  xin := LongIntPtr(NewPtr(Sizeof(LongInt) * N_DAYS));
  xout := LongIntPtr(NewPtr(Sizeof(LongInt) * N_DAYS));
  weekAct := LongIntPtr(NewPtr(Sizeof(LongInt) * N_WEEKS));
  weekIn := LongIntPtr(NewPtr(Sizeof(LongInt) * N_WEEKS));
  weekOut := LongIntPtr(NewPtr(Sizeof(LongInt) * N_WEEKS));
  xin^ := 0;
  xout^ := 0;

{ Create the records used to store statistics for each trajectory }
  CreatePredictionRecords;

{ This routine ask the user if individual trajectories are to be saved on disk }
  TOPENFILE(Fileno, errCode);
  for i := 1 to N_RUNS do
   begin

{ Generate 1 trajectory with parameters 'lambda' and 'mu' and initial value 'X0' }
{ and store the trajectory in 'x' , 'xin', 'xout' }
    KENDALL_SIMULATION(x, xin, xout, X0, lambda, mu);

{  Regroup the data by weeks. }
{ The resultant trajectory is then stored in 'weekAct' ,'weekIn' and 'weekOut' }
    REGROUP_DATA(x, xin, xout, weekAct, weekIn, weekOut);

{ Store trajectory on disk if it has been requested }
    if errCode = NoErr then
     WRITE_TRAJECTORY(x, Fileno);
    Kendall_Validate(weekAct, lambda, mu, i);

{ Calculate the estimators 'lambda1' and 'mu1' , and estimate the standard deviation of
'alpha:=lambda1-mu1' }
{ and 'beta:=1/(lambda1+mu1)' . }
    KENDALL_ESTIMATION(weekAct, weekIn, weekOut, lambda1, mu1, s_alpha, s_beta);

{ Calculate the predictions using the estimators 'lamba1' and 'mu1', and the standard deviations
s_alpha,s_beta }
    KENDALL_PREDICTION(weekAct, lambda1, mu1, s_alpha, s_beta, i);

{ If graphic output is requested plot the trajectory. }
    if SimOutput then
     SimPlot(x);
   end;

{ Close the trajectory file if output was written to it. }
  if errCode = NoErr then
```

```
  TCLOSEFILE(Fileno);

{ Compute statistics using every trajectories. }
 KENDALL_STATISTICS;

{ Deallocate the memory requested for the trajectories. }
 DisposePredictionRecords;
 DisposPtr(Ptr(weekAct));
 DisposPtr(Ptr(weekIn));
 DisposPtr(Ptr(weekOut));
 DisposPtr(Ptr(xin));
 DisposPtr(Ptr(xout));
 DisposPtr(Ptr(x));
end;
end.


unit Estimators;
interface
 uses
  SANE, Global;
 procedure REGROUP_DATA (x, xin, xout: LongIntPtr; weekAct, weekIn, weekOut: LongIntPtr);
 procedure KENDALL_ESTIMATION (weekAct, weekIn, weekOut: LongIntPtr; var lambda, mu, s_alpha,
s_beta: real);
 procedure KENDALL_PREDICTION (weekAct: LongIntPtr; lambda, mu, s_alpha, s_beta: real; aRun:
Integer);
implementation
{*********************************************************}
{ This procedure regroup the data by weeks. }
{  The trajectory over 1 week is summarized in 3 numbers }
{ 1- The number of actuals at the beginning of the week. }
{ 2- The total number of connections during the week. }
{ 3- The total number of deconnections during the week. }

 procedure REGROUP_DATA (x, xin, xout: LongIntPtr; weekAct, weekIn, weekOut: LongIntPtr);
  var
   i, j, t: integer;
   aweekin, aweekout, aweekact: LongInt;
 begin
  for i := 1 to N_WEEKS do
   begin
    t := (i - 1) * WeekLength + 1;
    aweekin := 0;
    aweekout := 0;
    aweekact := ReadLongInt(x, t);
    for j := t to t + WeekLength - 1 do
     begin
      if j <= N_DAYS then
       begin
```

```
      aweekin := aweekin + ReadLongInt(xin, j);
      aweekout := aweekout + ReadLongInt(xout, j);
      end;
    end;
   WriteLongInt(weekIn, i, aweekin);
   WriteLongInt(weekOut, i, aweekout);
   WriteLongInt(weekAct, i, aweekact);
  end;
end;

{*********************************************************}
{ This routine is used to compute the estimators according to the method described }
{ in the thesis. 'sumins' is the total number of connections during the period. }
{ 'sumouts' is the total number of deconnections. 'sumacts' is the integral of }
{ the actuals during that period. 'h' is a (usually small) correction to the }
{ evaluation of sumacts. 'N' is the total number of transactions during the period. }
{ 'lambda' and 'mu' are the estimators. 's_alpha' and 's_beta' are the estimates for }
{ the standard deviation of 'alpha:=lambda-mu' and 'beta:=1/(alpha+beta)'. }

 procedure KENDALL_ESTIMATION (weekAct, weekIn, weekOut: LongIntPtr; var lambda, mu, s_alpha,
s_beta: real);
  var
   t, i: integer;
   N, S_hat, sumacts, h: Real;
   sumins, sumouts, nextAct, anAct: LongInt;
 begin
  sumacts := 0.0;
  for t := Start_Estimation to End_Estimation do
   begin
    anAct := ReadlongInt(weekAct, t);
    nextAct := anAct + ReadLongInt(weekIn, t) - ReadLongInt(weekOut, t);
    h := ln(nextAct / anAct);
    sumacts := sumacts + anAct * WeekLength * (1.0 + h / 2);
   end;
  sumins := 0;
  sumouts := 0;
  for t := Start_Estimation to End_Estimation do
   begin
    sumins := sumins + ReadLongInt(weekIn, t);
    sumouts := sumouts + ReadLongInt(weekOut, t);
   end;
  N := sumins + sumouts;
  lambda := sumins / sumacts;
  mu := sumouts / sumacts;
  S_hat := sumacts / (sumins + sumouts);
  s_alpha := 1.0 / (sqrt(N) * S_hat);
  s_beta := S_hat / sqrt(N);

 end;
```

```
    sigma1 := sqrt(x0 * exp1 * (exp1 - 1.0) / (alpha * beta))
   else
    sigma1 := sqrt(x0 * t * WeekLength / beta);
   r := ln(1.0 + sqrt(t / Estimation_Length));
   alpha1 := alpha + s_alpha * r;
   beta1 := beta - s_beta * r;
   exp2 := exp(alpha1 * t * WeekLength);
   if (alpha1 <> 0.0) then
    sigma2 := sqrt(x0 * exp2 * (exp2 - 1.0) / (alpha1 * beta1)) + (x0 * exp2 - x1) / 2
   else
    sigma2 := sqrt(x0 * t * WeekLength / beta1) + (x0 - x1) / 2;

{ Some statistics are calculated here }
{ 'xsim' is the number of actuals at the current point of the trajectory. }
{ 'aPrediction^.new_sigma' is a measure of the extension measured in term of the original }
{ standard deviation. }
{ 'aPrediction^.deviation' is a measure of how far the simulated history deviates from the}
{ predicted mean. }
{ 'aPrediction^.DiscardPoint' records the the first time when the history leaves the 95% }
{  predictions bounds }

   xsim := ReadLongInt(weekAct, t + Start_Prediction - 1);
   WriteReal(aPrediction^.new_sigma, t, 2 * (sigma2 / sigma1));
   WriteReal(aPrediction^.deviation, t, abs((xsim - x1) / sigma1));
   if (x1 + 2 * sigma1 < xsim) or (x1 - 2 * sigma1 > xsim) then
    if aPrediction^.DiscardPoint > t then
     aPrediction^.DiscardPoint := t;
   if (x1 + 2 * sigma2 < xsim) or (x1 - 2 * sigma2 > xsim) then
    if aPrediction^.DiscardPoint1 > t then
     aPrediction^.DiscardPoint1 := t;
  end;
 end;
end.

unit Validation;
interface
 uses
  SANE, Global;
 procedure KENDALL_VALIDATE (weekAct: LongIntPtr; lambda, mu: real; aRun: Integer);
implementation

{*************************************************************}
{ This procedure calculates the predictions based on the parameters. }
{ The theory predict that the predictions should be correct 95% of the times, so this provide}
{ us a validation of the algorithm. }
 procedure KENDALL_VALIDATE (weekAct: LongIntPtr; lambda, mu: real; aRun: Integer);
  var
   t: integer;
   x0, t0, x1, sigma, exp1, xsim: extended;
```

```
{*********************************************************}
{ This procedure calculate the predictions based on the estimators and the standard deviations.}
{ Both the standard deviation from Nucho's formula and the extension algorithm are computed. }

 procedure KENDALL_PREDICTION (weekAct: LongIntPtr; lambda, mu, s_alpha, s_beta: real;        aRun:
Integer);
  var
   t, i, n: integer;
   x0, t0, x1, sigma1, sigma2, exp1, exp2, xsim, r: extended;
   alpha, beta, gamma, alpha1, beta1, Estimation_Length: extended;
   aPrediction: PredictionRecordPtr;
 begin
  alpha := lambda - mu;
  beta := 1 / (lambda + mu);

{ Get a pointer to the current 'prediction record' }
  aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, aRun));

{ Store the current values of the estimators and standard deviations }
  aPrediction^.lambda := lambda;
  aPrediction^.mu := mu;
  aPrediction^.s_beta := s_beta;
  aPrediction^.s_alpha := s_alpha;

{ Set default values for some statistics: }
{ 'Discardpoint' and 'DiscardPoint1' record the first time the simulated trajectory lies }
{ outside the predicted bounds. The first number is for Nucho's formula, the second number is }
{ for the extension algorithm }

  aPrediction^.DiscardPoint := End_Prediction - Start_Prediction + 2;
  aPrediction^.DiscardPoint1 := End_Prediction - Start_Prediction + 2;

{ Set the starting value for the prediction }
  x0 := ReadLongInt(weekAct, Start_Prediction - 1);

  Estimation_Length := End_Estimation - Start_Estimation + 1;

{ The calculations are done in the following loop }
{ 'x1' is the predicted mean. }
{ 'sigma1' is the standard deviation given by Nucho's formula. }
{ 'r' is the extension parameter given in the thesis for the extension algorithm. }
{ 'sigma2' is the extended standard deviation. }

  for t := 1 to End_Prediction - Start_Prediction + 1 do
   begin
    exp1 := exp(alpha * t * Weeklength);
    x1 := x0 * exp1;
    if (alpha <> 0.0) then
```

```
  alpha, beta, Estimation_Length: extended;
  aPrediction: PredictionRecordPtr;
 begin
  alpha := lambda - mu;
  beta := 1 / (lambda + mu);

  aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, aRun));

  aPrediction^.vDiscardPoint := End_Prediction - Start_Prediction + 2;
  x0 := ReadLongInt(weekAct, Start_Prediction - 1);
  t0 := 0.0;

  Estimation_Length := End_Estimation - Start_Estimation + 1;

  for t := 1 to End_Prediction - Start_Prediction + 1 do
   begin
    exp1 := exp(alpha * (t - t0) * Weeklength);
    x1 := x0 * exp1;
    sigma := sqrt(x0 * exp1 * (exp1 - 1.0) / (alpha * beta));

    xsim := ReadLongInt(weekAct, t + Start_Prediction - 1);
    WriteReal(aPrediction^.vdeviation, t, abs((xsim - x1) / sigma));
    if (x1 + 2 * sigma < xsim) or (x1 - 2 * sigma > xsim) then
     if aPrediction^.vDiscardPoint > t then
      aPrediction^.vDiscardPoint := t;
   end;
 end;
end.

unit Statistics;
interface
 uses
  SANE, Global, StatGraph;
 procedure KENDALL_STATISTICS;
 procedure InsertInArray (anArray: RealPtr; ArraySize, position: Integer; value: Real);
implementation
{************************************************************}
{ This procedure is called after every trajectories of the simulation has been  generated }
 procedure KENDALL_STATISTICS;
  var
   i, j, t, SortedArraySize: Integer;
   sum1, sum2, xmax, xmin, ymin, ymax, deviation, vdeviation: real;
   hit, vhit, hit1, count, vcount, count1, sigma_sum, new_sigma: real;
   aPrediction: PredictionRecordPtr;
   aSortedArray: RealPtr;
   r: rect;
 begin

{ Deallocate the memory used for statistics from the preceding simulation }
```

```
DisposPtr(Ptr(Stat.success));
DisposPtr(Ptr(Stat.vsuccess));
DisposPtr(Ptr(Stat.success1));
DisposPtr(Ptr(Stat.trajectory));
DisposPtr(Ptr(Stat.vtrajectory));
DisposPtr(Ptr(Stat.trajectory1));
DisposPtr(Ptr(Stat.confidence95));
DisposPtr(Ptr(Stat.new_sigma));
DisposPtr(Ptr(Stat.lambda_table));
DisposPtr(Ptr(Stat.mu_table));
DisposPtr(Ptr(Stat.s_beta_table));
DisposPtr(Ptr(Stat.s_alpha_table));

{ Allocate memory for statistics of the current simulation}

 Stat.nruns := N_RUNS;
 Stat.lambda_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
 Stat.mu_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
 Stat.s_beta_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
 Stat.s_alpha_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));

 Stat.pred_interval := End_Prediction - Start_Prediction + 1;
 Stat.success := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.vsuccess := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.success1 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.trajectory := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.vtrajectory := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.trajectory1 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.confidence95 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
 Stat.new_sigma := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));

{ Allocate memory for a sorted array }
 SortedArraySize := (Stat.nruns div 20) + 1;
 aSortedArray := RealPtr(NewPtr(Sizeof(Real) * SortedArraySize));

{ Find the average value of the estimators 'lambda' and 'mu' }
{ Extract the values of the estimators and standard deviations and}
{ place these values into a separate table }
 sum1 := 0.0;
 sum2 := 0.0;
 for i := 1 to Stat.nruns do
  begin
   aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, i));
   sum1 := sum1 + aPrediction^.lambda;
   sum2 := sum2 + aPrediction^.mu;
   WriteReal(Stat.lambda_table, i, aPrediction^.lambda);
   WriteReal(Stat.mu_table, i, aPrediction^.mu);
   WriteReal(Stat.s_beta_table, i, aPrediction^.s_beta);
```

```
   WriteReal(Stat.s_alpha_table, i, aPrediction^.s_alpha);
  end;
 Stat.lambda := sum1 / Stat.nruns;
 Stat.mu := sum2 / Stat.nruns;


 Stat.sigma_lambda := 0.0;
 Stat.sigma_mu := 0.0;
 If (Stat.nruns > 1) then
  begin

{ Find the standard deviation for the values of the estimators }
   for i := 1 to Stat.nruns do
    begin
     aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, i));
     Stat.sigma_lambda := Stat.sigma_lambda + (aPrediction^.lambda - Stat.lambda) *
(aPrediction^.lambda - Stat.lambda);
     Stat.sigma_mu := Stat.sigma_mu + (aPrediction^.mu - Stat.mu) * (aPrediction^.mu - Stat.mu);
    end;
   Stat.sigma_lambda := sqrt(Stat.sigma_lambda / (Stat.nruns - 1));
   Stat.sigma_mu := sqrt(Stat.sigma_mu / (Stat.nruns - 1));
  end;

{ The following loop is used to calculate the following values for every time period of the }
{ prediction interval }

{  hit: Number of times the actuals are within the predicted 95% intervals from Nucho's}
{ formula with estimators}
{ vhit:Number of times the actuals are within the predicted 95% intervals from Nucho's  }
{ formula with parameters}
{ hit1:Number of times the actuals are within the predicted 95% extended intervals }
{ count: Number of times we observed the actuals never left the interval (Nucho with }
{ estimators) up to that time period }
{ vcount: Number of times we observed the actuals never left the interval (Nucho with }
{ parameters) up to that time period }
{ count1: Number of times we observed the actuals never left the interval (extended) up to }
{that time period }
{ The average of these values is written to the 'stat' record. }
{ Also some other values are calculated : }
{stat.new_sigma: Average relative size of the extended interval vs Nucho interval. }
{stat.confidence95: Empirical 95% confidence interval, that is the smallest interval centered}
{ on the predicted mean}
{that would yield a 95% average for the variable 'hit'.  To calculate this we create a sorted}
{ array of values found}
{in 'deviation' . The array is made so it can hold only 95% of the values so the 5% largest }
{values will be excluded.}
{ The largest value of the array then correspond to the minimum size of the interval that you }
{yield a 95% average}
{ for the value 'hit'.}
```

```
for t := 1 to Stat.pred_interval do
 begin
  hit := 0;
  vhit := 0;
  hit1 := 0;
  count := 0;
  vcount := 0;
  count1 := 0;
  sigma_sum := 0.0;
  for j := 1 to SortedArraySize do
   WriteReal(aSortedArray, j, 0.0);

  for i := 1 to Stat.nruns do
   begin
    aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, i));
    new_sigma := ReadReal(aPrediction^.new_sigma, t);
    deviation := ReadReal(aPrediction^.deviation, t);
    vdeviation := ReadReal(aPrediction^.vdeviation, t);
    if aPrediction^.DiscardPoint > t then
     count := count + 1.0;
    if aPrediction^.DiscardPoint1 > t then
     count1 := count1 + 1.0;
    if aPrediction^.vDiscardPoint > t then
     vcount := vcount + 1.0;
    if deviation <= 2.0 then
     hit := hit + 1.0;
    if vdeviation <= 2.0 then
     vhit := vhit + 1.0;
    if deviation <= new_sigma then
     hit1 := hit1 + 1.0;
    j := 1;
    while (j <= SortedArraySize) do
     begin
     if deviation > ReadReal(aSortedArray, j) then
     begin
     InsertInArray(aSortedArray, SortedArraySize, j, deviation);
     j := SortedArraySize + 1;
     end
     else
     j := j + 1;
     end;
    sigma_sum := sigma_sum + new_sigma;
   end;
  hit := hit / Stat.nruns;
  vhit := vhit / Stat.nruns;
  hit1 := hit1 / Stat.nruns;
  count := count / Stat.nruns;
  vcount := vcount / Stat.nruns;
```

```
    count1 := count1 / Stat.nruns;
    new_sigma := sigma_sum / Stat.nruns;
    WriteReal(Stat.new_sigma, t, new_sigma);
    WriteReal(Stat.success, t, hit);
    WriteReal(Stat.vsuccess, t, vhit);
    WriteReal(Stat.success1, t, hit1);
    WriteReal(Stat.trajectory, t, count);
    WriteReal(Stat.vtrajectory, t, vcount);
    WriteReal(Stat.trajectory1, t, count1);
    WriteReal(Stat.confidence95, t, ReadReal(aSortedArray, SortedArraySize));
   end;

  for i := 1 to N_Graph do
   begin
    EnableItem(theMenu[StatMenu], i);
    if (WindowPeek(StatWindow[i])^.visible) then
     ShowStatWindow(i);
   end;
  DisposPtr(Ptr(aSortedArray));
 end;

{***********************************************************}
{ This routine is used to insert a real number at a given position in an array. }

 procedure InsertInArray (anArray: RealPtr; ArraySize, position: Integer; value: Real);
  var
   i: Integer;
 begin
  i := ArraySize;
  while i > position do
   begin
    WriteReal(anArray, i, ReadReal(anArray, i - 1));
    i := i - 1;
   end;
  WriteReal(anArray, position, value);
 end;
end.
```

A.2 Variables globales et fonctions utilitaires

```
{ This unit contains the declaration of the record used, global variables and some useful routines. }
unit Global;
interface
   uses
      SANE;
   const
      DAmenu = 1;
      FileMenu = 2;
      EditMenu = 3;
      SimMenu = 4;
      StatMenu = 5;
      N_MENU = 5;
      N_Graph = 7;
   type
      LongIntPtr = ^LongInt;
      IntPtr = ^Integer;
      RealPtr = ^Real;
      BooleanPtr = ^Boolean;
      PtrTable = ^Ptr;

{ This record holds settings info about 1 plot }
      GraphRecord = record
            nx: Integer;
            ny: Integer;
            xlabel: RealPtr;
            ylabel: RealPtr;
            xdigit: Integer;
            ydigit: Integer;
         end;
      GraphRecordPtr = ^GraphRecord;

{ Record that hold some statistics about 1 simulated trajectory }
      PredictionRecord = record
            lambda: real;  { Estimator }
            mu: real;      { Estimator }
            s_alpha: real; { Standard deviation of alpha := (lambda-mu) }
            s_beta: real;  { Standard deviation of beta := 1/(lambda+mu) }
            DiscardPoint: Integer; { First time trajectory goes outside regular bounds }
            DiscardPoint1: Integer; { First time trajectory goes outside extended bounds }
```

```
            deviation: RealPtr;  { Normalized difference between expected and observed trajectory }
            new_sigma: RealPtr; { Ratio of extended intervals to regular intervals }
            vdeviation: RealPtr; { Difference between true expected and observed trajectory }
            vDiscardPoint: Integer; { First time trajectory goes outside true bound }
          end;
      PredictionRecordPtr = ^PredictionRecord;

{ Main record for holding results about a simulation. (See 'STATISTICS' procedure)  }
      StatRecord = record
            lambda: real;  { Average estimator }
            mu: real; { Average estimator }
            sigma_lambda: real; { Std deviation of average estimator }
            sigma_mu: real; { Std devidation of average estimator }

            nruns: Integer; { Number of trajectories generated }
            lambda_table: RealPtr; { Array of estimators }
            mu_table: RealPtr; { Array of estimators }
            s_alpha_table: RealPtr; { Array of std deviation of alpha }
            s_beta_table: RealPtr; { Array of std deviation of beta }

            hist_interval: Integer; { Number of  interval for histograms }
            lambda_hist: RealPtr; { Array containing frequencies of lambda }
            mu_hist: RealPtr; { Array containing frequencies of mu }
            s_alpha_hist: RealPtr; { Array containing frequencies of std deviation estimates }
            s_beta_hist: RealPtr; { Array containing frequencies of std deviation estimates }

            pred_interval: Integer; { Length of prediction interval }
            success: RealPtr; { Array : Percentage of success with estimators and regular bound }
            vsuccess: RealPtr; { Array: Percentage of success with parameters and regular bounds }
            success1: RealPtr; { Array: Percentage of success with estimators and extended bounds }
            trajectory: RealPtr; { Array: Percentage of trajectories that never left the bounds }
            vtrajectory: RealPtr; { Array: Percentage of trajectories that never left the bounds }
            trajectory1: RealPtr; { Array: Percentage of trajectories that never left the bounds }
            new_sigma: RealPtr; { Array: Ratio of extended interval to regular interval }
            confidence95: RealPtr; { Array: Empirical 95% confidence interval }
          end;

   var
{ Graphics related variables... }
      GraphWindow: WindowPtr;
      SimGraph: GraphRecord;
      SimMap: BitMap;
      SimOutput: Boolean;
      StatWindow: array[1..N_Graph] of WindowPtr;
      StatGraph: array[1..N_Graph] of GraphRecord;
{ Simulation related variables...}
      Stat: StatRecord;
      Start_Estimation, End_Estimation: Integer;
      Start_Prediction, End_Prediction: Integer;
```

```
      N_DAYS, N_RUNS, N_WEEKS, WeekLength: Integer;
      X0: LongInt;
      lambda, mu: real;
{ Interface related variables...}
      theMenu: array[1..N_MENU] of MenuHandle;
      quit: Boolean;
      theEvent: EventRecord;
      DragRect: Rect;

{ Some useful functions...}
   function ReadLongInt (x: LongIntPtr; t: Integer): LongInt;
   procedure WriteLongInt (x: LongIntPtr; t: Integer; xvalue: LongInt);
   function ReadInt (x: IntPtr; t: Integer): Integer;
   procedure WriteInt (x: IntPtr; t: Integer; xvalue: Integer);
   function ReadBoolean (x: BooleanPtr; t: Integer): Boolean;
   procedure WriteBoolean (x: BooleanPtr; t: Integer; xvalue: Boolean);
   function ReadReal (x: RealPtr; t: Integer): Real;
   procedure WriteReal (x: RealPtr; t: Integer; xvalue: real);
   function ReadPtr (Table: PtrTable; t: Integer): Ptr;
   procedure WritePtr (Table: PtrTable; t: Integer; PointerValue: Ptr);
   procedure CreatePredictionRecords;
   procedure DisposePredictionRecords;

implementation

{*************************************************************}
{ Read a long integer at a position 't' of an array }
   function ReadLongInt (x: LongIntPtr; t: Integer): LongInt;
   begin
      x := LongIntPtr(Ord(x) + (t - 1) * sizeof(LongInt));
      ReadLongInt := x^;
   end;

{*************************************************************}
{ Write a long integer at a position 't' of an array }
   procedure WriteLongInt (x: LongIntPtr; t: Integer; xvalue: LongInt);
   begin
      x := LongIntPtr(Ord(x) + (t - 1) * sizeof(LongInt));
      x^ := xvalue;
   end;

{*************************************************************}
{ Read an integer at a position 't' of an array }
   function ReadInt (x: IntPtr; t: Integer): Integer;
   begin
      x := IntPtr(Ord(x) + (t - 1) * sizeof(Integer));
      ReadInt := x^;
   end;
```

```
{*************************************************************}
{ Write an integer at a position 't' of an array }
   procedure WriteInt (x: IntPtr; t: Integer; xvalue: Integer);
   begin
      x := IntPtr(Ord(x) + (t - 1) * sizeof(Integer));
      x^ := xvalue;
   end;

{*************************************************************}
{ Read a boolean value at a position 't' of an array }
   function ReadBoolean (x: BooleanPtr; t: Integer): Boolean;
   begin
      x := BooleanPtr(Ord(x) + (t - 1) * sizeof(Boolean));
      ReadBoolean := x^;
   end;

{*************************************************************}
{ Write  a boolean value at a position 't' of an array }
   procedure WriteBoolean (x: BooleanPtr; t: Integer; xvalue: Boolean);
   begin
      x := BooleanPtr(Ord(x) + (t - 1) * sizeof(Boolean));
      x^ := xvalue;
   end;

{*************************************************************}
{ Read a real number at a position 't' of an array }
   function ReadReal (x: RealPtr; t: Integer): Real;
   begin
      x := RealPtr(Ord(x) + (t - 1) * sizeof(Real));
      ReadReal := x^;
   end;

{*************************************************************}
{ Write a real number at a position 't' of an array }
   procedure WriteReal (x: RealPtr; t: Integer; xvalue: Real);
   begin
      x := RealPtr(Ord(x) + (t - 1) * sizeof(Real));
      x^ := xvalue;
   end;

{*************************************************************}
{ Read a pointer value at a position 't' of an array }
   function ReadPtr (Table: PtrTable; t: Integer): Ptr;
   begin
      Table := PtrTable(Ord(Table) + (t - 1) * sizeof(Ptr));
      ReadPtr := Table^;
   end;
```

```
{*********************************************************}
{ Write a pointer value at a position 't' of an array }

   procedure WritePtr (Table: PtrTable; t: Integer; PointerValue: Ptr);
   begin
      Table := PtrTable(Ord(Table) + (t - 1) * sizeof(Ptr));
      Table^ := PointerValue;
   end;

{*********************************************************}
   procedure CreatePredictionRecords;
      var
         i: Integer;
         aPrediction: PredictionRecordPtr;
   begin
      PredictionTable := PtrTable(NewPtr(Sizeof(Ptr) * N_RUNS));
      for i := 1 to N_RUNS do
         begin
            aPrediction := PredictionRecordPtr(NewPtr(Sizeof(PredictionRecord)));
            aPrediction^.deviation := RealPtr(NewPtr(Sizeof(Real) * (End_Prediction -
Start_Prediction + 1)));
            aPrediction^.vdeviation := RealPtr(NewPtr(Sizeof(Real) * (End_Prediction -
Start_Prediction + 1)));
            aPrediction^.new_sigma := RealPtr(NewPtr(Sizeof(Real) * (End_Prediction -
Start_Prediction + 1)));
            WritePtr(PredictionTable, i, Ptr(aPrediction));
         end;
   end;

{*********************************************************}
   procedure DisposePredictionRecords;
      var
         i: Integer;
         aPrediction: PredictionRecordPtr;
   begin
      for i := 1 to N_RUNS do
         begin
            aPrediction := PredictionRecordPtr(ReadPtr(PredictionTable, i));
            DisposPtr(Ptr(aPrediction^.deviation));
            DisposPtr(Ptr(aPrediction^.vdeviation));
            DisposPtr(Ptr(aPrediction^.new_sigma));
            DisposPtr(Ptr(aPrediction));
         end;
      DisposPtr(Ptr(PredictionTable));
      PredictionTable := nil;
   end;
end.
```

A.3 Routines graphiques

```
{ This units contains the necessary functions and procedures for plotting graphs and histograms }
unit GraphRoutine;
interface
    uses
        SANE, Global;
    function GetInterval (range: real): real;
    procedure TickLabel (min, max: real; tight: Boolean; var nLabel: Integer; var Labels: RealPtr);
    procedure SetupAxis (xmin, xmax, ymin, ymax: real; var aGraph: GraphRecord);
    procedure DisplayAxis (var aGraph: GraphRecord; r: rect);
    procedure PlotRun (n: Integer; data: LongIntPtr; var aGraph: GraphRecord; r: rect);
    procedure PlotHist (n: Integer; data: RealPtr; var aGraph: GraphRecord; r: rect);
    function GetHistInterval (range: real): real;
    procedure MakeHistogram (xin: RealPtr; runs: Integer; var xout: RealPtr; var n: Integer; var
aGraph: GraphRecord; var xmin, xmax, ymax: Real);
implementation

{*********************************************************}
{ This function compute a nice 'range' for one axis. }

    function GetInterval (range: real): real;
        var
            n: Integer;
            norm_range, interval: real;
    begin
        SetRound(Downward);
        n := Num2Integer(ln(range) / ln(10.0));
        norm_range := range / Xpwri(10.0, n);
        if norm_range <= 1.2 then
            interval := 0.2
        else if norm_range <= 3.0 then
            interval := 0.5
        else if norm_range <= 6.0 then
            interval := 1.0
        else
            interval := 2.0;
        GetInterval := interval * Xpwri(10.0, n);
    end;
```

```
{*********************************************************}
{ This procedure computes 'nice' intervals for the labels of one axis. }
   procedure TickLabel (min, max: real; tight: Boolean; var nLabel: Integer; var Labels: RealPtr);
      var
         interval, firstlabel, lastlabel: real;
         i: Integer;
   begin
      interval := GetInterval(max - min);
      SetRound(Downward);
      firstlabel := Rint((min * 1.0001) / interval) * interval;
      SetRound(Upward);
      lastlabel := Rint((max * 0.9999) / interval) * interval;
      SetRound(ToNearest);
      nLabel := Num2Integer((lastlabel - firstlabel) / interval) + 1;
      Labels := RealPtr(NewPtr(Sizeof(Real) * nLabel));
      for i := 1 to nlabel do
         WriteReal(Labels, i, firstlabel + (i - 1) * interval);
      if tight then
         begin
            WriteReal(Labels, 1, min);
            WriteReal(Labels, nLabel, max);
         end;
   end;

{*********************************************************}
{ This routine compute the tick labels intervals and the number of digit needed for the labels }

   procedure SetupAxis (xmin, xmax, ymin, ymax: real; var aGraph: GraphRecord);
   begin
      if aGraph.xlabel <> nil then
         begin
            DisposPtr(Ptr(aGraph.xlabel));
            DisposPtr(Ptr(aGraph.ylabel));
         end;
      TickLabel(xmin, xmax, True, aGraph.nx, aGraph.xlabel);
      TickLabel(ymin, ymax, False, aGraph.ny, aGraph.ylabel);
      SetRound(Downward);
      aGraph.xdigit := 1 - Num2Integer(ln(xmax - xmin) / ln(10.0));
      if aGraph.xdigit < 0 then
         aGraph.xdigit := 0;
      if aGraph.xdigit > 8 then
         aGraph.xdigit := 8;
      aGraph.ydigit := 1 - Num2Integer(ln(ymax - ymin) / ln(10.0));
      if aGraph.ydigit < 0 then
         aGraph.ydigit := 0;
      if aGraph.ydigit > 8 then
         aGraph.ydigit := 8;
   end;
```

```
{*********************************************************}
{ This routine draws the tick marks and labels for 1 plot }

    procedure DisplayAxis (var aGraph: GraphRecord; r: rect);
        var
            format: DecForm;
            str: DecStr;
            posx, posy, i, j: integer;
            xmin, xmax, ymin, ymax: integer;
            gxmin, gxmax, gymin, gymax: Real;
    begin
        SetRound(ToNearest);
        format.style := fixedDecimal;
        TextFont(helvetica);
        TextSize(10);
        gxmin := ReadReal(aGraph.xlabel, 1);
        gxmax := ReadReal(aGraph.xlabel, aGraph.nx);
        gymin := ReadReal(aGraph.ylabel, 1);
        gymax := ReadReal(aGraph.ylabel, aGraph.ny);
        xmin := r.left;
        xmax := r.right - 1;
        ymin := r.top;
        ymax := r.bottom - 1;
        FrameRect(r);

        format.digits := aGraph.xdigit;
        for i := 1 to aGraph.nx do
            begin
                posx := Num2Integer((ReadReal(aGraph.xlabel, i) - gxmin) * (xmax - xmin) / (gxmax -
gxmin)) + xmin;
                Moveto(posx, ymax);
                Lineto(posx, ymax - 4);
                Moveto(posx, ymin);
                Lineto(posx, ymin + 4);
                Num2Str(format, ReadReal(aGraph.xlabel, i), str);
                j := i mod 2;
                if (j = 0) then
                    Moveto(posx - StringWidth(str) div 2, ymax + 10)
                else
                    Moveto(posx - StringWidth(str) div 2, ymax + 20);
                DrawString(str);
            end;
        format.digits := aGraph.ydigit;
        for i := 1 to aGraph.ny do
            begin
                posy := ymax - Num2Integer((ReadReal(aGraph.ylabel, i) - gymin) * (ymax - ymin) /
(gymax - gymin));
                Moveto(xmax, posy);
                Lineto(xmax - 4, posy);
```

```
            Moveto(xmin, posy);
            Lineto(xmin + 4, posy);
            Num2Str(format, ReadReal(AGraph.ylabel, i), str);
            Moveto(xmin - 4 - StringWidth(str), posy + 5);
            DrawString(str);
          end;
    end;

{*********************************************************}
{ This procedure plot a trajectory from the values contained in the array 'data' }

    procedure PlotRun (n: Integer; data: LongIntPtr; var aGraph: GraphRecord; r: rect);
        var
            posx, posy, i: Integer;
            xmin, xmax, ymin, ymax: Integer;
            gxmin, gxmax, gymin, gymax, width, gposx: Real;
    begin
        SetRound(ToNearest);
        ClipRect(r);
        gxmin := ReadReal(aGraph.xlabel, 1);
        gxmax := ReadReal(aGraph.xlabel, aGraph.nx);
        gymin := ReadReal(aGraph.ylabel, 1);
        gymax := ReadReal(aGraph.ylabel, aGraph.ny);
        xmin := r.left;
        xmax := r.right - 1;
        ymin := r.top;
        ymax := r.bottom - 1;
        width := (gxmax - gxmin) / n;
        gposx := gxmin;
        posx := xmin;
        Moveto(posx, ymax);
        for i := 1 to n do
            begin
                posy := ymax - Num2Integer((ReadLongInt(data, i) - gymin) * (ymax - ymin) / (gymax -
gymin));
                Lineto(posx, posy);
                gposx := gposx + width;
                posx := Num2Integer((gposx - gxmin) * (xmax - xmin) / (gxmax - gxmin)) + xmin;
                Lineto(posx, posy);
            end;
        Lineto(posx, ymax);
    end;

{*********************************************************}
{ This procedure plot an histogram from the values contained in the array 'data' }

    procedure PlotHist (n: Integer; data: RealPtr; var aGraph: GraphRecord; r: rect);
        var
            posx, posy, i: Integer;
```

```
        xmin, xmax, ymin, ymax: Integer;
        gxmin, gxmax, gymin, gymax, width, gposx: Real;
  begin
     SetRound(ToNearest);
     ClipRect(r);
     gxmin := ReadReal(aGraph.xlabel, 1);
     gxmax := ReadReal(aGraph.xlabel, aGraph.nx);
     gymin := ReadReal(aGraph.ylabel, 1);
     gymax := ReadReal(aGraph.ylabel, aGraph.ny);
     xmin := r.left;
     xmax := r.right - 1;
     ymin := r.top;
     ymax := r.bottom - 1;
     width := (gxmax - gxmin) / n;
     gposx := gxmin;
     posx := xmin;
     Moveto(posx, ymax);
     for i := 1 to n do
        begin
           posy := ymax - Num2Integer((ReadReal(data, i) - gymin) * (ymax - ymin) / (gymax -
gymin));
           Lineto(posx, posy);
           gposx := gposx + width;
           posx := Num2Integer((gposx - gxmin) * (xmax - xmin) / (gxmax - gxmin)) + xmin;
           Lineto(posx, posy);
        end;
     Lineto(posx, ymax);
  end;

{*********************************************.*************}
{ This function compute a 'nice' interval length for an histogram }

  function GetHistInterval (range: real): real;
     var
        n: Integer;
        norm_range, interval: real;
  begin
     SetRound(Downward);
     n := Num2Integer(ln(range) / ln(10.0));
     norm_range := range / Xpwrl(10.0, n);
     if norm_range <= 1.2 then
        interval := 0.05
     else if norm_range <= 3.0 then
        interval := 0.10
     else if norm_range <= 6.0 then
        interval := 0.20
     else
        interval := 0.50;
     GetHistInterval := interval * Xpwrl(10.0, n);
```

```
    end;

{*************************************************************}
{ This routine construct an histogram from the values contained in 'xin'. The frequencies }
{ are returned in xout. }

    procedure MakeHistogram (xin: RealPtr; runs: Integer; var xout: RealPtr; var n: Integer; var
aGraph: GraphRecord; var xmin, xmax, ymax: Real);
        var
            i, j: Integer;
            x, dx: Real;
    begin
        SetRound(Downward);
        xmax := ReadReal(xin, 1);
        xmin := xmax;
        for i := 1 to runs do
            begin
                x := ReadReal(xin, i);
                if x < xmin then
                    xmin := x
                else if x > xmax then
                    xmax := x;
            end;
        if xmax = xmin then
            begin
                xmax := xmax + 0.5;
                xmin := xmin - 0.5;
            end;
        dx := GetHistInterval(xmax - xmin);
        SetRound(Downward);
        xmin := Rint(xmin / dx) * dx;
        SetRound(Upward);
        xmax := Rint(xmax / dx) * dx;
        SetRound(ToNearest);
        n := Num2Integer((xmax - xmin) / dx);
        DisposPtr(Ptr(xout));
        xout := RealPtr(NewPtr(Sizeof(Real) * n));
        ymax := 0;
        for i := 1 to n do
            WriteReal(xout, i, 0);
        for i := 1 to runs do
            begin
                j := Num2Integer((ReadReal(xin, i) - xmin) / dx) + 1;
                if j = n + 1 then
                    j := n;
                x := ReadReal(xout, j) + 1.0;
                WriteReal(xout, j, x);
                if ymax < x then
                    ymax := x;
```

```
        end;
      ymax := ymax + 1.0;
   end;
end.


unit SimGraph;
interface
   uses
      SANE, Global, GraphRoutine;
   procedure SimulationGraph;
   procedure SimPlot (x: LongIntPtr);
   procedure SetSigmaGraph;
   procedure SigmaPlot (aRun: Integer);
implementation
{*************************************************************}
{ This function returns the expected maximum vertical range for the simulation plot }

   function GetSimMax: Real;
      var
         arg1, arg2, t, exp1: real;
         jump: LongInt;
         i, jumptime: Integer;
         connection: boolean;
   begin
      arg1 := lambda - mu;
      arg2 := lambda + mu;
      t := N_DAYS;
      exp1 := exp(arg1 * t);
      GetSimMax := x0 * exp1 + 2 * sqrt(x0 * exp1 * (exp1 - 1.0) * arg2 / arg1);
   end;

{*************************************************************}
{ This routine contains the instructions to draw the simulation plot window. }
   procedure SimulationGraph;
      var
         format: DecForm;
         str: DecStr;
         aPort: GrafPort;
         r: rect;
   begin
      SelectWindow(GraphWindow);
      ShowWindow(GraphWindow);
      OpenPort(@aPort);
      SetPortBits(SimMap);
      FillRect(SimMap.bounds, white);
      format.style := fixedDecimal;
      format.digits := 5;
```

```
        TextFont(Geneva);
        TextSize(12);
        Nurn2Str(format, lambda, str);
        str := Concat('Lambda=', str);
        Moveto(40, 25);
        DrawString(Str);
        Num2Str(format, mu, str);
        str := Concat('Mu=', str);
        Moveto(190, 25);
        DrawString(str);
        Str := '(Days)';
        Moveto(200, 331);
        DrawString(Str);
        SetupAxis(0.0, N_DAYS, 0.0, GetSimMax, SimGraph);
        SetRect(r, 35, 50, 276, 301);
        DisplayAxis(SimGraph, r);
        CopyBits(SimMap, GraphWindow^.portBits, SimMap.bounds, GraphWindow^.portRect, srcCopy,
nil);
    end;

{************************************************************}
{ This procedure is used to plot a trajectory in the simulation plot window. The plot is then }
{ captured as a bitmap. }
    procedure SimPlot (x: LongIntPtr);
        var
            aPort: GrafPort;
            r: rect;
    begin
        OpenPort(@aPort);
        SetPortBits(SimMap);
        SetRect(r, 35, 50, 276, 301);
        PlotRun(N_DAYS, x, SimGraph, r);
        CopyBits(SimMap, GraphWindow^.portBits, SimMap.bounds, GraphWindow^.portRect, srcCopy,
nil);
    end;

end.


unit StatGraph;
interface
    uses
        SANE, Global, GraphRoutine;
    procedure ShowStatWindow (n: Integer);
    procedure DisplayStat1;
    procedure DisplayStat2;
    procedure DisplayStat3;
    procedure DisplayStat4;
    procedure DisplayStat5;
```

```
    procedure DisplayStat6;
    procedure DisplayStat7;

implementation

{*********************************************************}
{ The  procedures named 'DisplayStat1' to 'DisplayStat7' contain the informations to plot}
{ the 7 statistical windows. }
    procedure ShowStatWindow (n: Integer);
        var
            r: rect;
    begin
        SelectWindow(StatWindow[n]);
        ShowWindow(StatWindow[n]);
        SetPort(StatWindow[n]);
        ClipRect(StatWindow[n]^.portRect);
        SetRect(r, 0, 0, 300, 380);
        FillRect(r, white);
        case n of
            1:
                DisplayStat1;
            2:
                DisplayStat2;
            3:
                DisplayStat3;
            4:
                DisplayStat4;
            5:
                DisplayStat5;
            6:
                DisplayStat6;
            7:
                DisplayStat7;
        end;
    end;

{*********************************************************}
    procedure DisplayStat1;
        var
            format: DecForm;
            xmin, xmax, ymax: Real;
            str1, str2: DecStr;
            r: rect;
    begin
        ClipRect(StatWindow[1]^.portRect);
        format.style := fixedDecimal;
        SetRound(ToNearest);
        format.digits := 2 - Num2Integer(ln(Stat.lambda - Stat.mu) / ln(10.0));
        TextFont(Courier);
```

```
        TextSize(12);
        Num2Str(format, Stat.lambda, str1);
        Num2Str(format, Stat.sigma_lambda, str2);
        str1 := Concat('Lambda=', str1, '+/- ', str2);
        Moveto(30, 15);
        DrawString(str1);
        str1 := 'Distribution of Lambda estimates';
        Moveto(35, 35);
        DrawString(str1);
        str1 := '(Lambda estimates)';
        Moveto(170, 331);
        DrawString(str1);
        SetRect(r, 35, 50, 276, 301);
        MakeHistogram(Stat.lambda_table, Stat.nruns, Stat.lambda_hist, Stat.hist_interval,
StatGraph[1], xmin, xmax, ymax);
        SetupAxis(xmin, xmax, 0.0, ymax, StatGraph[1]);
        DisplayAxis(StatGraph[1], r);
        PlotHist(Stat.hist_interval, Stat.lambda_hist, StatGraph[1], r);
    end;
{*************************************************************}
    procedure DisplayStat2;
        var
            format: DecForm;
            xmin, xmax, ymax: Real;
            str1, str2: DecStr;
            r: rect;
    begin
        ClipRect(StatWindow[2]^.portRect);
        format.style := fixedDecimal;
        SetRound(ToNearest);
        format.digits := 2 - Num2Integer(ln(Stat.lambda - Stat.mu) / ln(10.0));
        TextFont(Courier);
        TextSize(12);
        Num2Str(format, Stat.mu, str1);
        Num2Str(format, Stat.sigma_mu, str2);
        str1 := Concat('Mu=', str1, '+/-', str2);
        Moveto(30, 15);
        DrawString(str1);
        str1 := 'Distribution of Mu estimates';
        Moveto(35, 35);
        DrawString(str1);
        str1 := '(Mu estimates)';
        Moveto(200, 331);
        DrawString(str1);
        SetRect(r, 35, 50, 276, 301);
        MakeHistogram(Stat.mu_table, Stat.nruns, Stat.mu_hist, Stat.hist_interval, StatGraph[2],
xmin, xmax, ymax);
        SetupAxis(xmin, xmax, 0.0, ymax, StatGraph[2]);
        DisplayAxis(StatGraph[2], r);
```

```
      PlotHist(Stat.hist_interval, Stat.mu_hist, StatGraph[2], r);
   end;

{*********************************************************}
   procedure DisplayStat3;
      var
         str: Str255;
         r: rect;
   begin
      ClipRect(StatWindow[3]^.portRect);
      TextFont(Courier);
      TextSize(12);
      str := 'Success ratio of predictions';
      Moveto(35, 18);
      DrawString(str);
      str := '(Weeks)';
      Moveto(200, 311);
      DrawString(str);
      SetRect(r, 35, 30, 276, 281);
      SetupAxis(0.0, stat.pred_interval, 0.5, 1.0, StatGraph[3]);
      PenPat(Gray);
      PlotHist(stat.pred_interval, Stat.success1, StatGraph[3], r);
      PenPat(dkGray);
      PlotHist(stat.pred_interval, Stat.vsuccess, StatGraph[3], r);
      PenPat(Black);
      PlotHist(stat.pred_interval, Stat.success, StatGraph[3], r);
      ClipRect(StatWindow[3]^.portRect);
      DisplayAxis(StatGraph[3], r);
   end;

{*********************************************************}
   procedure DisplayStat4;
      var
         str: Str255;
         r: rect;
   begin
      ClipRect(StatWindow[4]^.portRect);
      TextFont(Courier);
      TextSize(12);
      str := 'Proportion of trajectories within bounds';
      Moveto(15, 18);
      DrawString(str);
      str := '(Weeks)';
      Moveto(200, 311);
      DrawString(str);
      SetRect(r, 35, 30, 276, 281);
      SetupAxis(0.0, Stat.pred_interval, 0.5, 1.0, StatGraph[4]);
      PenPat(Gray);
      PlotHist(Stat.pred_interval, Stat.trajectory1, StatGraph[4], r);
```

```
        PenPat(dkGray);
        PlotHist(stat.pred_interval, Stat.vtrajectory, StatGraph[4], r);
        PenPat(Black);
        PlotHist(Stat.pred_interval, Stat.trajectory, StatGraph[4], r);
        ClipRect(StatWindow[4]^.portRect);
        DisplayAxis(StatGraph[4], r);
    end;

{*********************************************************}
    procedure DisplayStat5;
        var
            format: DecForm;
            xmin, xmax, ymax: Real;
            str: DecStr;
            r: rect;
    begin
        ClipRect(StatWindow[5]^.portRect);
        format.style := fixedDecimal;
        SetRound(ToNearest);
        format.digits := 2 - Num2Integer(ln(Stat.lambda - Stat.mu) / ln(10.0));
        TextFont(Courier);
        TextSize(12);
        Num2Str(format, 1 / (Stat.lambda + Stat.mu), str);
        str := Concat('1/(Lambda+Mu)=', str);
        Moveto(30, 15);
        DrawString(str);
        str := 'Standard deviation of 1/(Lambda+Mu)';
        Moveto(15, 35);
        DrawString(str);
        str := '(Standard deviation)';
        Moveto(150, 331);
        DrawString(str);
        SetRect(r, 35, 50, 276, 301);
        MakeHistogram(Stat.s_beta_table, Stat.nruns, Stat.s_beta_hist, Stat.hist_interval,
StatGraph[5], xmin, xmax, ymax);
        SetupAxis(xmin, xmax, 0.0, ymax, StatGraph[5]);
        DisplayAxis(StatGraph[5], r);
        PlotHist(Stat.hist_interval, Stat.s_beta_hist, StatGraph[5], r);
    end;

{*********************************************************}
    procedure DisplayStat6;
        var
            format: DecForm;
            xmin, xmax, ymax: Real;
            str: DecStr;
            r: rect;
    begin
        ClipRect(StatWindow[6]^.portRect);
```

```
        format.style := fixedDecimal;
        SetRound(ToNearest);
        format.digits := 2 - Num2Integer(ln(Stat.lambda - Stat.mu) / ln(10.0));
        TextFont(Courier);
        TextSize(12);
        Num2Str(format, Stat.lambda - Stat.mu, str);
        str := Concat('Lambda-Mu=', str);
        Moveto(30, 15);
        DrawString(str);
        str := 'Standard deviation of Lambda-Mu';
        Moveto(15, 35);
        DrawString(str);
        str := '(Standard deviation)';
        Moveto(150, 331);
        DrawString(str);
        SetRect(r, 35, 50, 276, 301);
        MakeHistogram(Stat.s_alpha_table, Stat.nruns, Stat.s_alpha_hist, Stat.hist_interval,
StatGraph[6], xmin, xmax, ymax);
        SetupAxis(xmin, xmax, 0.0, ymax, StatGraph[6]);
        DisplayAxis(StatGraph[6], r);
        PlotHist(Stat.hist_interval, Stat.s_alpha_hist, StatGraph[6], r);
    end;

{*************************************************************}
    procedure DisplayStat7;
        var
            str: Str255;
            r: rect;
    begin
        ClipRect(StatWindow[7]^.portRect);
        TextFont(Courier);
        TextSize(12);
        str := '95% Confidence in Std Deviations (Nucho)';
        Moveto(10, 18);
        DrawString(str);
        str := '(Weeks)';
        Moveto(200, 311);
        DrawString(str);
        SetRect(r, 35, 30, 276, 281);
        SetupAxis(0.0, Stat.pred_interval, 0.0, 4.0, StatGraph[7]);
        PenPat(Gray);
        PlotHist(Stat.pred_interval, Stat.new_sigma, StatGraph[7], r);
        PenPat(Black);
        PlotHist(Stat.pred_interval, Stat.confidence95, StatGraph[7], r);
        ClipRect(StatWindow[7]^.portRect);
        DisplayAxis(StatGraph[7], r);
    end;

end.
```

A.4 Routines d'entrées / sorties

```
unit Tfile;
{ This unit contains the I/O routines }
interface
   uses
      SANE, Global;
   procedure TOPENFILE (var aRefNum, errCode: Integer);
   procedure TCLOSEFILE (aRefNum: Integer);
   procedure WRITE_TRAJECTORY (X: LongIntPtr; aRefNum: Integer);
   procedure STATFILE;
implementation
{*************************************************************}
{ This procedure is to ask the user if the trajectories are to be saved on disk }

   procedure TOPENFILE (var aRefNum, errCode: Integer);
      var
         p: Point;
         theReply: SFReply;
         filename: str255;
   begin
      errCode := 0;
      SetPt(p, 60, 30);
      SFPutFile(p, 'Save Trajectories as:', 'traject.out', nil, theReply);
      if (not theReply.good) then
         Exit(TOPENFILE);
      filename := copy(theReply.fName, 1, length(theReply.fName));
      errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
      if errCode = DupFNErr then
         begin
            errCode := FSDelete(filename, theReply.vRefNum);
            if errCode = FBsyErr then
               begin
                  errCode := StopAlert(512, nil);
                  Exit(TOPENFILE);
               end;
            errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
         end;
      if errCode = NoErr then
         errCode := FSOpen(filename, theReply.vRefNum, aRefNum);
   end;
```

```
{*************************************************************}
   procedure TCLOSEFILE (aRefNum: Integer);
      var
         errCode: Integer;
   begin
      errCode := FSClose(aRefNum);
      errCode := FlushVol(nil, 0);
   end;

{*************************************************************}
{ This routine write 1 trajectory on disk (ASCII format) }
   procedure WRITE_TRAJECTORY (X: LongIntPtr; aRefNum: Integer);
      var
         i, errCode: Integer;
         count: Longint;
         buffer: str255;
   begin
      for i := 1 to N_DAYS do
         begin
            NumToString(ReadLongInt(X, i), buffer);
            if i = N_DAYS then
               buffer := Concat(buffer, chr(13))
            else
               buffer := Concat(buffer, ',');
            count := Length(buffer);
            errCode := FSWrite(aRefnum, count, ptr(ord(@buffer) + 1));
         end;
   end;

{*************************************************************}
{ This routine is for saving some simulation statitics on disk  (ASCII format) }
   procedure STATFILE;
      var
         p: Point;
         theReply: SFReply;
         aRefNum, errCode, i: Integer;
         count: LongInt;
         str1, str2, str3, str4, str5, str6: DecStr;
         format: DecForm;
         filename, buffer: str255;
   begin
      if Stat.success = nil then
         Exit(STATFILE);
      errCode := 0;
      SetPt(p, 60, 30);
      SFPutFile(p, 'Save statistics as:', 'stat.out', nil, theReply);
      if (not theReply.good) then
         Exit(STATFILE);
```

```
        filename := copy(theReply.fName, 1, length(theReply.fName));
        errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
        if errCode = DupFNErr then
            begin
                errCode := FSDelete(filename, theReply.vRefNum);
                if errCode = FBsyErr then
                    begin
                        errCode := StopAlert(512, nil);
                        Exit(STATFILE);
                    end;
                errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
            end;
        if errCode = NoErr then
            begin
                errCode := FSOpen(filename, theReply.vRefNum, aRefNum);
                format.style := fixedDecimal;
                format.digits := 4;
                for i := 1 to Stat.pred_interval do
                    begin
                        Num2Str(format, ReadReal(Stat.success, i), str1);
                        Num2Str(format, ReadReal(Stat.vsuccess, i), str2);
                        Num2Str(format, ReadReal(Stat.success1, i), str3);
                        Num2Str(format, ReadReal(Stat.trajectory, i), str4);
                        Num2Str(format, ReadReal(Stat.confidence95, i), str5);
                        Num2Str(format, ReadReal(Stat.new_sigma, i), str6);
                        buffer := Concat(str1, ',', str2, ',', str3, chr(13));
                        count := Length(buffer);
                        errCode := FSWrite(aRefnum, count, ptr(ord(@buffer) + 1));
                    end;
                errCode := FSClose(aRefNum);
                errCode := FlushVol(nil, 0);
            end;
    end;

end.


unit IOfile;
interface
    uses
        SANE, Global, StatGraph;
    procedure WriteEstimators;
    procedure SaveSimulation;
    procedure LoadSimulation;
implementation
{************************************************************}
{ This routine is used to save the values of the estimators in an ASCII file }

    procedure WriteEstimators;
```

```
    var
        p: Point;
        i, aRefNum, errCode: Integer;
        count: Longint;
        theReply: SFReply;
        str1, str2: DecStr;
        filename, buffer: str255;
        format: DecForm;
begin
    if Stat.success = nil then
        Exit(WriteEstimators);
    SetPt(p, 60, 30);
    SFPutFile(p, 'Save Estimators as:', 'Estimators_data', nil, theReply);
    if (not theReply.good) then
        Exit(WriteEstimators);
    filename := copy(theReply.fName, 1, length(theReply.fName));
    errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
    if errCode = DupFNErr then
        begin
            errCode := FSDelete(filename, theReply.vRefNum);
            if errCode = FBsyErr then
                begin
                    errCode := StopAlert(512, nil);
                    Exit(WriteEstimators);
                end;
            errCode := Create(filename, theReply.vRefNum, '????', 'TEXT');
        end;
    if errCode = NoErr then
        begin
            errCode := FSOpen(filename, theReply.vRefNum, aRefNum);
            format.style := fixedDecimal;
            format.digits := 6;
            for i := 1 to N_RUNS do
                begin
                    Num2Str(format, ReadReal(stat.lambda_table, i), str1);
                    Num2Str(format, ReadReal(stat.mu_table, i), str2);
                    buffer := Concat(str1, ' ', str2, chr(13));
                    count := Length(buffer);
                    errCode := FSWrite(aRefnum, count, ptr(ord(@buffer) + 1));
                end;
            errCode := FSClose(aRefNum);
            errCode := FlushVol(nil, 0);
        end;
end;

{*************************************************************}
{ This routine is used to save the settings and results of the simulation as a binary}
{ file on disk so that it can be read latter by the program.}
```

```
procedure SaveSimulation;
    var
        p: Point;
        aRefNum, errCode: Integer;
        Size: LongInt;
        theReply: SFReply;
        filename: str255;
begin
    if Stat.success = nil then
        Exit(SaveSimulation);
    SetPt(p, 60, 30);
    SFPutFile(p, 'Save simulation as:', 'Sim.res', nil, theReply);
    if (not theReply.good) then
        Exit(SaveSimulation);
    filename := copy(theReply.fName, 1, length(theReply.fName));
    errCode := Create(filename, theReply.vRefNum, '????', 'KDLL');
    if errCode = DupFNErr then
        begin
            errCode := FSDelete(filename, theReply.vRefNum);
            if errCode = FBsyErr then
                begin
                    errCode := StopAlert(512, nil);
                    Exit(SaveSimulation);
                end;
            errCode := Create(filename, theReply.vRefNum, '????', 'KDLL');
        end;
    if errCode = NoErr then
        begin
            errCode := FSOpen(filename, theReply.vRefNum, aRefNum);
            Size := Sizeof(Real);
            errCode := FSWrite(aRefnum, Size, ptr(@X0));
            errCode := FSWrite(aRefnum, Size, ptr(@lambda));
            errCode := FSWrite(aRefnum, Size, ptr(@mu));
            Size := Sizeof(Integer);
            errCode := FSWrite(aRefnum, Size, ptr(@N_RUNS));
            errCode := FSWrite(aRefnum, Size, ptr(@N_WEEKS));
            errCode := FSWrite(aRefnum, Size, ptr(@WeekLength));
            errCode := FSWrite(aRefnum, Size, ptr(@Start_Estimation));
            errCode := FSWrite(aRefnum, Size, ptr(@End_Estimation));
            errCode := FSWrite(aRefnum, Size, ptr(@Start_Prediction));
            errCode := FSWrite(aRefnum, Size, ptr(@End_Prediction));

            Size := Sizeof(Real);
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.lambda));
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.mu));
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.sigma_lambda));
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.sigma_mu));
            Size := Sizeof(Integer);
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.nruns));
```

```
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.hist_interval));
            errCode := FSWrite(aRefnum, Size, ptr(@Stat.pred_interval));
            Size := Sizeof(Real) * stat.nruns;
            errCode := FSWrite(aRefnum, Size, ptr(Stat.lambda_table));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.mu_table));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.s_alpha_table));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.s_beta_table));
            Size := Sizeof(Real) * stat.pred_Interval;
            errCode := FSWrite(aRefnum, Size, ptr(Stat.success));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.vsuccess));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.success1));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.trajectory));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.vtrajectory));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.trajectory1));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.confidence95));
            errCode := FSWrite(aRefnum, Size, ptr(Stat.new_sigma));

            Size := 28000;
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.baseAddr));
            Size := Sizeof(Integer);
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.rowBytes));
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.bounds.top));
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.bounds.left));
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.bounds.right));
            errCode := FSWrite(aRefnum, Size, ptr(SimMap.bounds.bottom));

            errCode := FSClose(aRefNum);
            errCode := FlushVol(nil, 0);
        end;
    end;

{*********************************************************}
{ This routine is used to read the settings and results of a simulation from a file on disk }
{ that was previously created by this program.}

    procedure LoadSimulation;
        var
            p: Point;
            i, aRefNum, errCode: Integer;
            Size: Longint;
            theReply: SFReply;
            typeList: SFTypeList;
            findrInfo: FInfo;
    begin
        SetPt(p, 60, 30);
        typeList[0] := 'KDLL';
        SFGetFile(p, '', nil, 1, typeList, nil, theReply);
        if (not theReply.good) then
            Exit(LoadSimulation);
```

```
errCode := GetFInfo(theReply.fname, theReply.vRefNum, findrInfo);
if errCode = NoErr then
    begin
        if Stat.success <> nil then
            begin
                DisposPtr(Ptr(Stat.success));
                DisposPtr(Ptr(Stat.vsuccess));
                DisposPtr(Ptr(Stat.success1));
                DisposPtr(Ptr(Stat.trajectory));
                DisposPtr(Ptr(Stat.vtrajectory));
                DisposPtr(Ptr(Stat.trajectory1));
                DisposPtr(Ptr(Stat.confidence95));
                DisposPtr(Ptr(Stat.new_sigma));
                DisposPtr(Ptr(Stat.lambda_table));
                DisposPtr(Ptr(Stat.mu_table));
                DisposPtr(Ptr(Stat.s_beta_table));
                DisposPtr(Ptr(Stat.s_alpha_table));
            end;
        errCode := FSOpen(theReply.fname, theReply.vRefNum, aRefNum);
        Size := Sizeof(Real);
        errCode := FSRead(aRefnum, Size, ptr(@X0));
        errCode := FSRead(aRefnum, Size, ptr(@lambda));
        errCode := FSRead(aRefnum, Size, ptr(@mu));
        Size := Sizeof(Integer);
        errCode := FSRead(aRefnum, Size, ptr(@N_RUNS));
        errCode := FSRead(aRefnum, Size, ptr(@N_WEEKS));
        errCode := FSRead(aRefnum, Size, ptr(@WeekLength));
        errCode := FSRead(aRefnum, Size, ptr(@Start_Estimation));
        errCode := FSRead(aRefnum, Size, ptr(@End_Estimation));
        errCode := FSRead(aRefnum, Size, ptr(@Start_Prediction));
        errCode := FSRead(aRefnum, Size, ptr(@End_Prediction));
        N_DAYS := N_WEEKS * WeekLength;

        Size := Sizeof(Real);
        errCode := FSRead(aRefnum, Size, ptr(@Stat.lambda));
        errCode := FSRead(aRefnum, Size, ptr(@Stat.mu));
        errCode := FSRead(aRefnum, Size, ptr(@Stat.sigma_lambda));
        errCode := FSRead(aRefnum, Size, ptr(@Stat.sigma_mu));
        Size := Sizeof(Integer);
        errCode := FSRead(aRefnum, Size, ptr(@Stat.nruns));
        errCode := FSRead(aRefnum, Size, ptr(@Stat.hist_interval));
        errCode := FSRead(aRefnum, Size, ptr(@Stat.pred_interval));

        Size := Sizeof(Real) * stat.nruns;
        Stat.lambda_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
        Stat.mu_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
        Stat.s_beta_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
        Stat.s_alpha_table := RealPtr(NewPtr(Sizeof(Real) * Stat.nruns));
        errCode := FSRead(aRefnum, Size, ptr(Stat.lambda_table));
```

```
            errCode := FSRead(aRefnum, Size, ptr(Stat.mu_table));
            errCode := FSRead(aRefnum, Size, ptr(Stat.s_alpha_table));
            errCode := FSRead(aRefnum, Size, ptr(Stat.s_beta_table));

            Size := Sizeof(Real) * stat.pred_Interval;
            Stat.success := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.vsuccess := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.success1 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.trajectory := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.vtrajectory := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.trajectory1 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.confidence95 := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            Stat.new_sigma := RealPtr(NewPtr(Sizeof(Real) * stat.pred_interval));
            errCode := FSRead(aRefnum, Size, ptr(Stat.success));
            errCode := FSRead(aRefnum, Size, ptr(Stat.vsuccess));
            errCode := FSRead(aRefnum, Size, ptr(Stat.success1));
            errCode := FSRead(aRefnum, Size, ptr(Stat.trajectory));
            errCode := FSRead(aRefnum, Size, ptr(Stat.vtrajectory));
            errCode := FSRead(aRefnum, Size, ptr(Stat.trajectory1));
            errCode := FSRead(aRefnum, Size, ptr(Stat.confidence95));
            errCode := FSRead(aRefnum, Size, ptr(Stat.new_sigma));

            Size := 28000;
            errCode := FSRead(aRefnum, Size, ptr(SimMap.baseAddr));
            Size := Sizeof(Integer);
            errCode := FSRead(aRefnum, Size, ptr(SimMap.rowBytes));
            errCode := FSRead(aRefnum, Size, ptr(SimMap.bounds.top));
            errCode := FSRead(aRefnum, Size, ptr(SimMap.bounds.left));
            errCode := FSRead(aRefnum, Size, ptr(SimMap.bounds.right));
            errCode := FSRead(aRefnum, Size, ptr(SimMap.bounds.bottom));

            errCode := FSClose(aRefNum);
            errCode := FlushVol(nil, 0);

            SelectWindow(GraphWindow);
            ShowWindow(GraphWindow);
            CopyBits(SimMap, GraphWindow^.portBits, SimMap.bounds, GraphWindow^.portRect,
srcCopy, nil);
            for i := 1 to N_Graph do
               begin
                  EnableItem(theMenu[StatMenu], i);
                  if (WindowPeek(StatWindow[i])^.visible) then
                     ShowStatWindow(i);
               end;
         end;
   end;
end.
```

A.5 Routines des fenêtres de 'dialogue'

```
{ This section contains the implementation of the dialogs}
unit SimDialog;
interface
   uses
      SANE, Global;
   procedure DoSimDialog;
implementation
{*********************************************************}
{ This procedure opens a dialog that allow the user to specify the simulation's settings }

   procedure DoSimDialog;
      const
         OKBtn = 2;
         cancelBtn = 3;
         GraphCkBox = 4;
         LambdaText = 7;
         MuText = 9;
         StartvalueText = 11;
         DurationText = 13;
         WlengthText = 15;
         RunText = 17;
      var
         theDialog: DialogPtr;
         itemHit, theType, n: Integer;
         r: rect;
         GraphCheckBox: Handle;
         RunBox, LambdaBox, MuBox, StartvalueBox, DurationBox, WeeklengthBox: Handle;
         done, ok: Boolean;
         s: Str255;
         duration_value, weeklength_value, run_value, start_value: LongInt;
         lambda_value, mu_value: Real;
         format: DecForm;
   begin
      theDialog := GetNewDialog(192, nil, Pointer(-1));
      GetDItem(theDialog, GraphCkBox, theType, GraphCheckBox, r);
      GetDItem(theDialog, LambdaText, theType, LambdaBox, r);
      GetDItem(theDialog, MuText, theType, MuBox, r);
      GetDItem(theDialog, StartvalueText, theType, StartvalueBox, r);
      GetDItem(theDialog, DurationText, theType, DurationBox, r);
```

```
GetDItem(theDialog, WlengthText, theType, WeeklengthBox, r);
GetDItem(theDialog, RunText, theType, RunBox, r);
done := False;
ok := False;
if SimOutput then
   n := 1
else
   n := 0;
SetCtlValue(ControlHandle(GraphCheckBox), n);
format.style := fixedDecimal;
format.digits := 5;

lambda_value := lambda;
mu_value := mu;
start_value := X0;
duration_value := N_WEEKS;
weeklength_value := Weeklength;
run_value := N_RUNS;

Num2Str(format, lambda_value, DecStr(s));
SetIText(LambdaBox, s);
Num2Str(format, mu_value, DecStr(s));
SetIText(MuBox, s);
NumtoString(start_value, s);
SetIText(StartvalueBox, s);
NumtoString(duration_value, s);
SetIText(DurationBox, s);
NumtoString(weeklength_value, s);
SetIText(WeeklengthBox, s);
NumtoString(run_value, s);
SetIText(RunBox, s);

repeat
   ModalDialog(nil, itemHit);
   case itemHit of
      CancelBtn:
         done := True;
      OKBtn:
         begin
            done := True;
            ok := true;
         end;
      GraphCkBox:
         begin
            n := GetCtlValue(ControlHandle(GraphCheckBox));
            n := (n + 1) mod 2;
            SetCtlValue(ControlHandle(GraphCheckBox), n);
            SimOutput := not SimOutput;
         end;
```

```
        LambdaText:
            begin
                GetIText(LambdaBox, s);
                lambda_value := Str2Num(DecStr(s));
            end;
        MuText:
            begin
                GetIText(MuBox, s);
                mu_value := Str2Num(DecStr(s));
            end;
        StartvalueText:
            begin
                GetIText(StartvalueBox, s);
                StringToNum(s, start_value);
            end;
        DurationText:
            begin
                GetIText(DurationBox, s);
                StringToNum(s, duration_value);
            end;
        WlengthText:
            begin
                GetIText(WeeklengthBox, s);
                StringToNum(s, weeklength_value);
            end;
        RunText:
            begin
                GetIText(RunBox, s);
                StringToNum(s, run_value);
            end;
    end;
until done;
DisposDialog(theDialog);
if ok then
    begin
        mu := mu_value;
        lambda := lambda_value;
        X0 := Integer(start_value);
        N_RUNS := Integer(run_value);
        if N_RUNS < 1 then
            N_RUNS := 1;
        if N_RUNS > 8192 then
            N_RUNS := 8192;
        N_WEEKS := Integer(duration_value);
        WeekLength := Integer(weeklength_value);
        if N_WEEKS < 2 then
            N_WEEKS := 2;
        N_DAYS := N_WEEKS * Weeklength;
        if End_Prediction > N_WEEKS then
```

```
            End_Prediction := N_WEEKS;
          if End_Estimation > N_WEEKS then
            End_Estimation := N_WEEKS;
          if Start_Prediction > N_WEEKS then
            Start_Prediction := N_WEEKS;
          if Start_Estimation > N_WEEKS then
            Start_Estimation := N_WEEKS;
        end;
  end;
end.


unit EstDialog;
interface
  uses
    SANE, Global;
  procedure DoEstimationDialog;
implementation
{*************************************************************}
  procedure DoEstimationDialog;
    const
      OKBtn = 2;
      cancelBtn = 3;
      StartEstimationText = 6;
      EstimationLengthText = 8;
      StartPredictionText = 11;
      PredictionLengthText = 13;
    var
      theDialog: DialogPtr;
      itemHit, theType: Integer;
      StartEstimationBox, EstimationLengthBox: Handle;
      StartPredictionBox, PredictionLengthBox: Handle;
      done, ok: Boolean;
      r: rect;
      s: Str255;
      StartEstimation_value, Estimation_duration: LongInt;
      StartPrediction_value, Prediction_duration: LongInt;
  begin
    theDialog := GetNewDialog(200, nil, Pointer(-1));
    GetDItem(theDialog, StartEstimationText, theType, StartEstimationBox, r);
    GetDItem(theDialog, EstimationLengthText, theType, EstimationLengthBox, r);
    GetDItem(theDialog, StartPredictionText, theType, StartPredictionBox, r);
    GetDItem(theDialog, PredictionLengthText, theType, PredictionLengthBox, r);
    done := False;
    ok := False;
    StartEstimation_value := LongInt(Start_Estimation);
    Estimation_duration := LongInt(End_Estimation - Start_Estimation + 1);
    StartPrediction_value := LongInt(Start_Prediction);
    Prediction_duration := LongInt(End_Prediction - Start_Prediction + 1);
```

```
NumtoString(StartEstimation_value, s);
SetIText(StartEstimationBox, s);
NumtoString(Estimation_duration, s);
SetIText(EstimationLengthBox, s);
NumtoString(StartPrediction_value, s);
SetIText(StartPredictionBox, s);
NumtoString(Prediction_duration, s);
SetIText(PredictionLengthBox, s);

repeat
   ModalDialog(nil, itemHit);
   case itemHit of
      CancelBtn:
         done := True;
      OKBtn:
         begin
            done := True;
            ok := true;
         end;
      StartEstimationText:
         begin
            GetIText(StartEstimationBox, s);
            StringToNum(s, StartEstimation_value);
         end;
      EstimationLengthText:
         begin
            GetIText(EstimationLengthBox, s);
            StringToNum(s, Estimation_duration);
         end;
      StartPredictionText:
         begin
            GetIText(StartPredictionBox, s);
            StringToNum(s, StartPrediction_value);
         end;
      PredictionLengthText:
         begin
            GetIText(PredictionLengthBox, s);
            StringToNum(s, Prediction_duration);
         end;
   end;
until done;
DisposDialog(theDialog);
if ok then
   begin
      if Prediction_duration < 1 then
         Prediction_duration := 1;
      if Estimation_duration < 1 then
         Estimation_duration := 1;
```

```
            Start_Prediction := StartPrediction_value;
            if Start_Prediction < 2 then
               Start_Prediction := 2
            else if Start_Prediction > N_WEEKS then
               Start_Prediction := N_WEEKS;
            End_Prediction := Start_Prediction + Prediction_duration - 1;
            if End_Prediction > N_WEEKS then
               End_Prediction := N_WEEKS
            else if End_Prediction < 2 then
               End_Prediction := 2;
            Start_Estimation := StartEstimation_value;
            if Start_Estimation < 1 then
               Start_Estimation := 1
            else if Start_Estimation > N_WEEKS then
               Start_Estimation := N_WEEKS;
            End_Estimation := Start_Estimation + Estimation_duration - 1;
            if End_Estimation > N_WEEKS then
               End_Estimation := N_WEEKS
            else if End_Estimation < 1 then
               End_Estimation := 1;
         end;
   end;
end.


unit ConfirmDialog;
interface
   uses
      SANE, Global, Simulation;
   procedure DoConfirmDialog;
implementation
{*************************************************************}
{ This procedure opens a dialog for showing the current settings of the simulation}
{ and to request a confirmation before starting the simulation. }
   procedure DoConfirmDialog;
      const
         OKBtn = 2;
         cancelBtn = 3;
         LambdaText = 4;
         MuText = 5;
         StartText = 6;
         DurationText = 7;
         RunText = 8;
         EstimationText = 9;
         PredictionText = 10;
      var
         theDialog: DialogPtr;
         itemHit, theType, n: Integer;
         r: rect;
```

```
        GraphCheckBox: Handle;
        RunBox, LambdaBox, MuBox, StartBox, DurationBox, PredictionBox, EstimationBox: Handle;
        Done, Ok: Boolean;
        str1, str2: Str255;
        format: DecForm;
begin
    theDialog := GetNewDialog(208, nil, Pointer(-1));
    GetDItem(theDialog, LambdaText, theTvpe, LambdaBox, r);
    GetDItem(theDialog, MuText, theType, MuBox, r);
    GetDItem(theDialog, StartText, theType, StartBox, r);
    GetDItem(theDialog, DurationText, theType, DurationBox, r);
    GetDItem(theDialog, RunText, theType, RunBox, r);
    GetDItem(theDialog, EstimationText, theType, EstimationBox, r);
    GetDItem(theDialog, PredictionText, theType, PredictionBox, r);
    Done := False;
    Ok := False;
    format.style := fixedDecimal;
    format.digits := 5;

    Num2Str(format, lambda, DecStr(str1));
    str1 := Concat('Lambda = ', str1);
    SetIText(LambdaBox, str1);
    Num2Str(format, mu, DecStr(str1));
    str1 := Concat('Mu = ', str1);
    SetIText(MuBox, str1);
    NumtoString(X0, str1);
    str1 := Concat('Start value = ', str1);
    SetIText(StartBox, str1);
    NumtoString(N_WEEKS, str1);
    str1 := Concat('Number of weeks = ', str1);
    SetIText(DurationBox, str1);
    NumtoString(N_RUNS, str1);
    str1 := Concat('Number of  runs = ', str1);
    SetIText(RunBox, str1);
    NumtoString(Start_Estimation, str1);
    NumtoString(End_Estimation, str2);
    str1 := Concat('Estimation from week #', str1, ' to week #', str2);
    SetIText(EstimationBox, str1);
    NumtoString(Start_Prediction, str1);
    NumtoString(End_Prediction, str2);
    str1 := Concat('Prediction from week #', str1, ' to week #', str2);
    SetIText(PredictionBox, str1);

    repeat
        ModalDialog(nil, itemHit);
        case itemHit of
            CancelBtn:
                Done := True;
            OKBtn:
```

```
                Ok := true;
            end;
        until (Ok or Done);
        DisposDialog(theDialog);
        if Ok then
            begin
                KENDALL_PROCESS(X0, lambda, mu);
                for n := 1 to 3 do
                    EnableItem(theMenu[FileMenu], n);
            end;
    end;
end.
```

A.6 Routine de gestion de l'interface

```
unit ProcessEvent;
{ This unit contains window managements routine, such as responding to a mouse click,  a key press,}
{ an 'update' event in a window, or to the activation of a control. }

interface
    uses
        Global, SimGraph, StatGraph, MenuCommand;
    procedure DoMouseDown;
    procedure DoKeyDown;
    procedure DoUpdate;
    procedure DoActivate;
implementation
{*********************************************************}
    procedure DoMouseDown;
        var
            theWindow: WindowPtr;
            code: Integer;
            n: Integer;
            menuResult: LongInt;
    begin
        code := FindWindow(theEvent.where, theWindow);
        case code of
            inMenuBar:
                begin
                    menuResult := MenuSelect(theEvent.where);
                    DoMenuCommand(menuResult);
                    HiliteMenu(0);
                end;
            inContent:
                if theWindow <> FrontWindow then
                    SelectWindow(theWindow);
            inDrag:
                DragWindow(theWindow, theEvent.where, DragRect);
            inGoAway:
                if TrackGoAway(theWindow, theEvent.where) then
                    HideWindow(theWindow);
            inSysWindow:
                SystemClick(theEvent, theWindow);
        end;
```

```
    end;

{*************************************************************}
    procedure DoKeyDown;
        var
            c: Char;
    begin
        c := Chr(BitAnd(theEvent.message, charCodeMask));
        if BitAnd(theEvent.modifiers, cmdKey) <> 0 then
            begin
                DoMenuCommand(MenuKey(c));
                HiliteMenu(0);
            end
        else
            SysBeep(1);
    end;

{*************************************************************}
    procedure DoUpdate;
        var
            theWindow: WindowPtr;
            mask: RgnHandle;
            myRect: Rect;
            n: Integer;
    begin
        theWindow := WindowPtr(theEvent.message);
        if theWindow = GraphWindow then
            begin
                BeginUpdate(theWindow);
                mask := theWindow^.visRgn;
                CopyBits(SimMap, theWindow^.portBits, SimMap.bounds, theWindow^.portRect, srcCopy,
mask);
                EndUpdate(theWindow);
            end
        else if theWindow = StatWindow[1] then
            begin
                SetPort(theWindow);
                BeginUpdate(theWindow);
                DisplayStat1;
                EndUpdate(theWindow);
            end
        else if theWindow = StatWindow[2] then
            begin
                SetPort(theWindow);
                BeginUpdate(theWindow);
                DisplayStat2;
                EndUpdate(theWindow);
            end
        else if theWindow = StatWindow[3] then
```

```
        begin
            SetPort(theWindow);
            BeginUpdate(theWindow);
            DisplayStat3;
            EndUpdate(theWindow);
        end
    else if theWindow = StatWindow[4] then
        begin
            SetPort(theWindow);
            BeginUpdate(theWindow);
            DisplayStat4;
            EndUpdate(theWindow);
        end
    else if theWindow = StatWindow[5] then
        begin
            SetPort(theWindow);
            BeginUpdate(theWindow);
            DisplayStat5;
            EndUpdate(theWindow);
        end
    else if theWindow = StatWindow[6] then
        begin
            SetPort(theWindow);
            BeginUpdate(theWindow);
            DisplayStat6;
            EndUpdate(theWindow);
        end
    else if theWindow = StatWindow[7] then
        begin
            SetPort(theWindow);
            BeginUpdate(theWindow);
            DisplayStat7;
            EndUpdate(theWindow);
        end;
end;

{*************************************************************}
procedure DoActivate;
    var
        theWindow: WindowPtr;
        theControl: ControlHandle;
        n: Integer;
begin
    theWindow := WindowPtr(theEvent.message);
    SetPort(theWindow);
    theControl := windowPeek(theWindow)^.controlList;
    if Odd(theEvent.modifiers) then
        n := 0
    else
```

```
        n := 255;
      while theControl <> nil do
        begin
          HiliteControl(theControl, n);
          theControl := theControl^^.nextControl;
        end;
  end;
end.


unit MenuCommand;
interface
  uses
    SANE, Global, IOfile, StatGraph, ConfirmDialog, SimDialog, EstDialog, TFile;
  procedure DoMenuCommand (menuResult: LongInt);
implementation
{************************************************************}

  procedure DoAppleMenu (theItem: Integer);
    var
      name: str255;
  begin
    GetItem(theMenu[1], theItem, name);
    if OpenDeskAcc(name) > 0 then
      ;
  end;

{************************************************************}
{ This procedure is used to respond to a mouse click in one of the menus. }
  procedure DoMenuCommand (menuResult: LongInt);
    var
      aMenu, theItem: Integer;
  begin
    aMenu := HiWord(menuResult);
    theItem := LoWord(menuResult);
    case aMenu of
      DAmenu:
        DoAppleMenu(theItem);
      FileMenu:
        case theItem of
          1:
            WriteEstimators;
          2:
            STATFILE;
          3:
            SaveSimulation;
          4:
            LoadSimulation;
          6:
```

```
                quit := true;
            end;
        EditMenu:
            if SystemEdit(theItem - 1) then
                ;
        SimMenu:
            case theItem of
                1:
                    DoSimDialog;
                2:
                    DoEstimationDialog;
                3:
                    DoConfirmDialog;
            end;
        StatMenu:
            ShowStatWindow(theItem);
    end;
  end;
end.
```

A.7 Initialisation et programme principal

```
program Kendall_Interface;
   uses
      Global, ProcessEvent;
{*************************************************************}
{ This routine is used to initialize the interface and to give default values to variables of }
{ the simulation. }

   procedure Initialize;
      var
         r: Rect;
         aWindow: WindowPtr;
         n: integer;
         aControl: ControlHandle;
   begin
      MoreMasters;
      MoreMasters;
      MoreMasters;
      MoreMasters;
      InitGraf(@thePort);
      InitFonts;
      InitWindows;
      InitMenus;
      InitCursor;
      quit := False;
      SimOutput := True;

      GraphWindow := getNewWindow(128, nil, Pointer(-1));
      with SimMap do
         begin
            baseAddr := NewPtr(28000);
            rowBytes := 80;
            SetRect(bounds, 0, 0, 300, 350);
         end;
      SimGraph.xlabel := nil;
      SimGraph.ylabel := nil;

      for n := 1 to N_Graph do
         begin
            StatWindow[n] := getNewWindow(130 + n, nil, Pointer(-1));
```

```
        StatGraph[n].xlabel := nil;
        StatGraph[n].ylabel := nil;
      end;

  for n := 1 to N_MENU do
    theMenu[n] := GetMenu(n);
  AddResMenu(theMenu[1], 'DRVR');
  for n := 1 to N_MENU do
    InsertMenu(theMenu[n], 0);
  for n := 1 to 3 do
    DisableItem(theMenu[FileMenu], n);
  for n := 1 to 6 do
    DisableItem(theMenu[EditMenu], n);
  for n := 1 to N_Graph do
    DisableItem(theMenu[StatMenu], n);
  dragRect := screenbits.bounds;
  InsetRect(dragRect, 25, 25);
  DrawMenuBar;
  X0 := 100;
  lambda := 0.0032;
  mu := 0.0016;
  N_RUNS := 100;
  N_WEEKS := 52;
  Weeklength := 7;
  N_DAYS := N_WEEKS * Weeklength;
  Start_Estimation := 1;
  End_Estimation := 26;
  Start_Prediction := 27;
  End_Prediction := 52;
  PredictionTable := nil;

  Stat.success := nil;
  Stat.vsuccess := nil;
  Stat.success1 := nil;
  Stat.trajectory := nil;
  Stat.vtrajectory := nil;
  Stat.trajectory1 := nil;
  Stat.lambda_table := nil;
  Stat.mu_table := nil;
  Stat.s_beta_table := nil;
  Stat.s_alpha_table := nil;
  Stat.lambda_hist := nil;
  Stat.mu_hist := nil;
  Stat.s_beta_hist := nil;
  Stat.s_alpha_hist := nil;
  Stat.confidence95 := nil;
  Stat.new_sigma := nil;
  Stat.hist_interval := 20;
end;
```

```
{************************************************************}
{ Main driver module }
begin
    Initialize;
    repeat
        SystemTask;
        if GetNextEvent(everyEvent, theEvent) then
            case theEvent.what of
                mouseDown:
                    DoMouseDown;
                keyDown:
                    DoKeyDown;
                updateEvt:
                    DoUpdate;
                activateEvt:
                    DoActivate;
            end;
    until quit;
end.
```

Appendice B

Algorithme: calcul de r

Nous supposons comme étant donné les valeurs suivantes: α,β, X_0, t, t^*

Certaines constantes doivent être sélectionnées:

- L_1 et L_2 sont les limites de l'intégration sur les normales.

$$L_1 = 4\bar{\sigma}_\alpha$$

$$L_2 = 4\bar{\sigma}_\beta$$

est un choix adéquat.

- La valeur initiale pour r. Une valeur entre 0 et 1 est généralement choisie.

- ϵ est la tolérance sur la valeur de la somme. Une valeur de 0.001 a été utilisée.

- δ est une constante positive déterminant à quelle vitesse la valeur de r change d'une itération à l'autre. Les valeurs typiques utilisées vont de 0.05 à 1.

Étape 1: Initialisation

$$N = \frac{1}{\alpha\beta} X_0(e^{\alpha t^*} - 1)$$

$$\bar{\sigma}_\alpha = \frac{1}{\beta\sqrt{N}}$$

$$\bar{\sigma}_\beta = \frac{\beta}{\sqrt{N}}$$

Étape 2: Itération

Calculez la valeur de

$$S = \int_{-L_1}^{+L_1} N(\hat{\alpha}; \alpha, \sigma_\alpha^2) d\hat{\alpha} \int_{-L_2}^{+L_2} N(\hat{\beta}; \beta, \sigma_\beta^2) d\hat{\beta} \int_{\hat{m}-2\sigma_e}^{\hat{m}+2\sigma_e} N(x; m, v) dz$$

où $N(x; \mu, \sigma^2)$ est la densité normale avec moyenne μ et variance σ^2. Dans cette expression $m = X_0 e^{\alpha t}$ et $v = X_0 \frac{1}{\alpha\beta} e^{\alpha t}(e^{\alpha t} - 1)$. Nous utilisons aussi:

$$\hat{\alpha}_s = \hat{\alpha} + r\bar{\sigma}_{\hat{\alpha}}$$

$$\hat{\beta}_s = \hat{\beta} - r\bar{\sigma}_{\hat{\beta}}$$

$$\hat{m} = X_0 e^{\hat{\alpha} t}, \hat{m}_s = X_0 e^{\hat{\alpha}_s t}$$

$$\hat{\sigma}_s = \sqrt{X_0 \frac{1}{\hat{\alpha}_s \hat{\beta}_s} e^{\hat{\alpha}_s t}(e^{\hat{\alpha}_s t} - 1)}$$

$$\hat{\sigma}_e = \frac{\hat{m}_s - \hat{m} + 2\hat{\sigma}_s}{2}.$$

Étape 3: Test de la valeur de S

Si $|S - 0.9544| \leq \epsilon$ alors STOP.

sinon

$$r \leftarrow r + \delta(0.9544 - S)$$

et retour à l'étape 2.

Appendice C

Simulation, estimation of parameters and predictions
for a Kendall process

Thesis Long Summary

February 1994

1 Introduction

1.1 The Kendall Model

The Kendall birth and death process has been proposed by Nucho (1979) as an adequate description of the growth in point-to-point demand. In this model the transition rates are proportional to the current demand. If we denote by $X(t)$ the demand at time t then

$$dX(t) = dN(t) - dM(t)$$

where $N(t)$ counts the number of new connections by time t and $M(t)$ counts the number of disconnections by time t. The intensities of these point processes are $\lambda X(t)$ and $\mu X(t)$.

When $\lambda > \mu$ the process $X(t)$, starting at $n(0)$, is transient. Since there is a negligable probability of absorption at 0 we assume for simplicity that 0 is a reflecting point. The mean $m(t)$ and variance $v(t)$ increase according to the following formulae (Nucho 1979):

$$m(t) = n(0)e^{(\lambda-\mu)t}$$

$$v(t) = n(0)\frac{\lambda+\mu}{\lambda-\mu}e^{2t(\lambda-\mu)}[1 - e^{-(\lambda-\mu)t}].$$

If $\lambda > \mu$ then

$$\frac{X(t) - m(t)}{\sqrt{v(t)}} \rightarrow N(0,1).$$

Our objective is to estimate λ and μ from past history of the process during a time period of length t^*. Then, with the estimators $\hat{\lambda}$ and $\hat{\mu}$ we want to obtain a prediction interval t time units in the future.

1.2 The Estimators

In this section we estimate the parameters λ and μ. Suppose that during the period $[0, t^*]$ there were a total of N transactions and we denote by $\{\eta_k : k = 1, 2, \cdots, N\}$ the time between successive transactions. If $W_k = n$ then η_k is exponential with mean $1/[n(\lambda+\mu)]$. Let $Y_k := \exp(-W_k\eta_k)$ so for $0 \leq t \leq 1$,

$$P(Y_k \leq t | W_k = n) = P(\eta_k \leq -\frac{1}{n}\ln t | W_k = n) = t^{\lambda+\mu}.$$

We conclude that the transformed variables Y_k are i.i.d. and a simple calculation shows

$$\hat{\beta} := -\frac{(\sum_{k=1}^{N} \ln(Y_k)}{N} = \frac{\sum_{k=1}^{N} W_k \eta_k}{N}$$

is a maximum likelyhood estimator for $1/(\lambda+\mu)$. The estimator $\hat{\beta}$ is asymptotically normal since the variables $\{Y_k = W_k\eta_k\}$ are i.i.d. exponentials with mean $\beta := 1/(\lambda+\mu)$ so

$$\sqrt{N}\frac{\hat{\beta} - 1/(\lambda+\mu)}{\hat{\beta}} \rightarrow N(0,1).$$

An independent estimate may be made of the odds ratio $\lambda/(\lambda+\mu)$. Define $\{\xi_k : k = 1, \cdots, N\}$ to be $+1$ for connections and -1 for deconnections. The maximum likelihood estimator, $\hat{\rho}$ for $\lambda/(\lambda+\mu)$ is

$$\hat{\rho} := \sum_{k=1}^{N} \xi_k/(2N) + 1/2.$$

The estimator $\hat{\rho}$ is asymptotically normal and

$$\sqrt{N}\frac{\hat{\rho} - \lambda/(\lambda+\mu)}{\sqrt{\hat{\rho}(1-\hat{\rho})}} \Rightarrow N(0,1).$$

A little arithmetic shows $(2\hat{\rho}-1)/\hat{\beta}$ is a maximum likelihood estimator for $\alpha := \lambda - \mu$. It can be shown that:

$$\sqrt{N}\hat{\beta}[\hat{\alpha} - (\lambda-\mu)] \rightarrow N(0,1)$$

We also see that $\sum_{k=1}^{N}\{\xi_k = 1\}/\sum_{k=1}^{N} W_k\eta_k$ and $\sum_{k=1}^{N}\{\xi_k = -1\}/\sum_{k=1}^{N} W_k\eta_k$ are maximum likelyhood estimators for λ and μ.

2 Simulation of a Kendall Process

2.1 Generation a Kendall Process

The Kendall process can be simulated by superposing two non-homogeneous Poisson processes. If $X[i]$ is the demand at time t_i, and if U_1 and U_2 are

pseudo-uniform random numbers on $[0,1]$ then the cumulative distribution function of C_i (the number of connections at time t_i) and D_i (the number of disconnections at time t_i) are

$$F_1(C_i) = \sum_{j=0}^{C_i} e^{-\lambda X[i]} \frac{(\lambda X[i])^j}{j!}$$

$$F_2(D_i) = \sum_{j=0}^{D_i} e^{-\mu X[i]} \frac{(\mu X[i])^j}{j!}$$

and the demand at the time t_{i+1} is:

$$X[i+1] = X[i] + F_1^{-1}(U_1) - F_2^{-1}(U_2)$$

With given values for λ and μ, and an initial value for demand, we can simulate the evolution of that demand over a given period of time. We have developped a Pascal program with a friendly user-interface to generate histories. The user can control several parameters of the simulation such as λ, μ, X_0, the number of weeks for which to generate data, the period of estimation, the period of prediction and the number of histories to generate. A value for the demand is generated for each day but only one record per week is kept. This record contains the number of transactions in the week and the value of the demand on the last day. We use these records to calculate the estimators and to test our predictions. A complete listing of the program is given in Appendix A of the thesis.

2.2 Calculation of the Estimators

The maximum likelihood estimators are defined by:

$$\hat{\lambda} := \frac{\sum_{k=1}^{N} \{\xi_k = 1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

$$\hat{\mu} := \frac{\sum_{k=1}^{N} \{\xi_k = -1\}}{\sum_{k=1}^{N} W_k \eta_k}$$

The ξ_k sum is the total number of connections/disconnections generated during the estimation period. However we do not know the values of the η_k. We

only know the demand at fixed times and the number of transactions during the intervals. An approximation could be:

$$\sum_{i=1}^{L} X[i]\Delta t \approx \sum_{k=1}^{N} W_k \eta_k$$

where the $X[i]$'s are the demand at fixed times, with time interval Δt. This approximation is analogous to a technique of numerical integration of a function. (See fig. 2.2 of thesis) This analogy can inspire us to obtain a slightly better approximation (fig 2.3): Instead of taking $X[i]$ constant on $[t_{i-1}, t_i]$, we use an interpolation by an exponential function. This choice is natural since the expected value of the demand grows exponentially. The interpolation $\hat{X}_i$ can be written as:

$$\hat{X}_i = X[i]\frac{e^{h_i \Delta t} - 1}{h_i}$$

and h_i is determined by:

$$X[i+1] = X[i]e^{h_i \Delta t}$$

When h_i is small, the estimators can be calculated with:

$$\hat{\lambda} = \frac{\sum_{i=1}^{L} C_i}{\sum_{i=1}^{L} X[i]\Delta t(1 + \frac{h_i \Delta t}{2})}$$

$$\hat{\mu} = \frac{\sum_{i=1}^{L} D_i}{\sum_{i=1}^{L} X[i]\Delta t(1 + \frac{h_i \Delta t}{2})}$$

Using the histories generated in the simulation, we calculated the estimators using the method just presented. It turned out that these estimators tend to be normally distributed around their expected mean (see fig 2.4 and fig 2.5).

2.3 Success of Prediction

With the estimations for the parameters λ and μ, we calculated the 2σ prediction interval using Nucho's formulae. However these formulae are correct only if we use the exact value of the parameters, not their estimators. The consequence is that in general our confidence interval is not quite centered on the mean of the distribution (see fig 2.6). The result is that in general our

confidence interval captures less than 95% of the distribution. This is clearly demonstrated by the fact that the success ratio of our predictions, that is the fraction of observations which are actually inside our confidence interval, tend to decrease with each passing week. We noticed that the rate at which this ratio decreases depends primarily on the length of the time period we used to calculate the estimators $\hat{\lambda}$ and $\hat{\mu}$ (See figure 2.7, 2.9-2.11). To obtain a constant 95% success ratio we would need either an infinite estimation period or a wider confidence interval. The second solution is somewhat more realistic.

3 Finding a 95% Confidence Interval

3.1 A method to extend the confidence interval

The estimated mean using Nucho's formula is:

$$\hat{m}(t) = X_0 e^{(\hat{\lambda}-\hat{\mu})t}$$

We have a guaranty to have at least a 95% confidence interval with

$$\sigma_e(t) = \sigma(t) + \frac{|m(t) - \hat{m}(t)|}{2}$$

By replacing $\sigma(t)$ and $m(t)$ by some overestimate, it is possible to find an adequate value for $\sigma_e(t)$. Define

$$\hat{\alpha} := \hat{\lambda} - \hat{\mu}$$

$$\hat{\beta} := \frac{1}{\hat{\lambda} + \hat{\mu}}$$

Then we have:

$$\hat{m}(t) = X_0 e^{\hat{\alpha}t}$$

$$\hat{\sigma}(t) = \frac{X_0}{\hat{\alpha}\hat{\beta}} e^{\hat{\alpha}t}(e^{\hat{\alpha}t} - 1)$$

$$\hat{\sigma}_{\hat{\alpha}} = \frac{\hat{\beta}}{\sqrt{N}}$$

$$\hat{\sigma}_{\hat{\beta}} = \frac{1}{\sqrt{N}\hat{\beta}}$$

We can obtain an overestimate of $\hat{m}(t)$ by overestimating $\hat{\alpha}$.

$$\hat{\alpha}_s := \hat{\alpha} + r\hat{\sigma}_{\hat{\alpha}}$$

where r is some non-negative number. Then

$$\hat{m}_s(t) := X_0 e^{\hat{\alpha}_s t}$$

is an overestimate of $\hat{m}(t)$. Similarly we overestimate $\hat{\sigma}(t)$:

$$\hat{\beta}_s := \hat{\beta} - r\hat{\sigma}_{\hat{\beta}}$$

and we have:

$$\hat{\sigma}_s(t) := X_0 \frac{1}{\hat{\alpha}_s \hat{\beta}_s} e^{\hat{\alpha}_s t}(e^{\hat{\alpha}_s t} - 1)$$

The extended confidence interval is given by:

$$\hat{\sigma}_e(t) = \hat{\sigma}_s(t) + \frac{|\hat{m}_s(t) - \hat{m}(t)|}{2}$$

Note that if $\hat{\sigma}_{\hat{\alpha}} \to 0$ and $\hat{\sigma}_{\hat{\beta}} \to 0$ we have $\hat{m}_s(t) \to \hat{m}(t)$ and $\hat{\sigma}_s(t) \to \hat{\sigma}(t)$ and we get back Nucho's formula:

$$\hat{\sigma}_e(t) = \hat{\sigma}(t)$$

3.2 Determination of r

The problem is to determine the minimal r required to capture 95% of the distribution. On average, the fraction of the distribution captured is:

$$I = \int_{-\infty}^{+\infty} N(\hat{\alpha}; \alpha, \sigma_\alpha^2) \int_{-\infty}^{+\infty} N(\hat{\beta}; \beta, \sigma_\beta^2) \int_{\hat{m}-2\hat{\sigma}_e}^{\hat{m}+2\hat{\sigma}_e} N(x; m, v) dx d\hat{\beta} d\hat{\alpha}$$

where $N(x; \mu, \sigma^2)$ is the normal density with mean μ and variance σ^2. In this expression $m = X_0 e^{\alpha t}$ and $v = X_0 \frac{1}{\alpha\beta} e^{\alpha t}(e^{\alpha t} - 1)$. We also have

$$\hat{\sigma}_{\hat{\alpha}} = \frac{\hat{\beta}}{\sqrt{E(N)}}$$

$$\hat{\sigma}_{\hat{\beta}} = \frac{1}{\sqrt{E(N)}\hat{\beta}}$$

$$E(N) = \int_0^{t^*} (\lambda + \mu)E(X(t))dt = \frac{\lambda + \mu}{\lambda - \mu} X_0(e^{(\lambda-\mu)t^*} - 1)$$

where t^* is the length of the period on which the estimators are calculated. We then can determine the value of r by an iterative method with α, β, t, t^*, and X_0 given.(see Appendix B of thesis) We observed that the values of r tended to be the same for fixed values of t/t^*. And for fixed values of t/t^*, r showed only a weak dependence on X_0, α or β. Plotting r as a function of t/t^* we found that the function $\hat{r}(t) = ln(1 + \sqrt{t/t^*})$ was a good approximation for r (See figures 3.2-3.6).

3.3 Extension Algorithm

We now present the algorithm to extend the confidence interval:

$$\hat{\alpha} = \hat{\lambda} - \hat{\mu}$$

$$\hat{\beta} = \frac{1}{\hat{\lambda} + \hat{\mu}}$$

$$\hat{\alpha}_s(t) = \hat{\alpha} + ln(1 + \sqrt{t/t^*})\hat{\sigma}_{\hat{\alpha}}$$

$$\hat{\beta}_s(t) = \hat{\beta} - ln(1 + \sqrt{t/t^*})\hat{\sigma}_{\hat{\beta}}$$

$$\hat{m}(t) = X_0 e^{\hat{\alpha}t}$$

$$\hat{m}_s(t) = X_0 e^{t\hat{\alpha}_s(t)}$$

$$\hat{\sigma}_s(t) = X_0 \frac{1}{\hat{\alpha}_s(t)\hat{\beta}_s(t)} e^{t\hat{\alpha}_s(t)}(e^{t\hat{\alpha}_s(t)} - 1)$$

$$\hat{\sigma}_e(t) = \hat{\sigma}_s(t) + \frac{|\hat{m}_s(t) - \hat{m}(t)|}{2}$$

The interval $(\hat{m}(t) - 2\hat{\sigma}_e(t), \hat{m}(t) + 2\hat{\sigma}_e(t))$ capture 95% of the distribution on average. We validated this algorithm using our simulation. The proportion of observations that are within the confidence interval stays constant near 95% (See figures 3.7-3.9). Also the factor by which the interval is widened is directly related to the length of the period of estimation. By using a longer period of estimation we need less enlargement to get a 95% confidence interval (See figures 3.10-3.12).

4 Application of the Kendall Model

4.1 Method of analysis

This part of the work was done essentially on the SAS statistical analysis system in the CMS and Aix-Windows environments.

The database we worked on included end-to-end transactions from February 1992 to March 1993. Another database included the actuals in place for every end-to-end, as of September 15, 1992. The first thing we did was to sort the database to eliminate every end-to-end which had no corresponding actuals as of September 15, 1992. We had to eliminate a large number of end-to-ends for this reason. It is possible that many of these were simply badly identified since there was also a large number of end-to-ends with actuals but no corresponding transactions.

For each end-to-end at the client level, we reconstructed the history. We have noted that a large fraction of the end-to-end demands only had a few records. So for the analysis at this level we restricted ourselves to analyse only those end-to-ends which had at least 20 records of transactions. The method we have used is the following: We took the date of the first and the last transaction on record, and split the time interval in two equal part. We used the data from the first period to calculate the estimators and used them to make predictions of the demand of the second period. We then tested our predictions by comparing with the observations of the second period. The first results we obtained showed that our predictions gave good results for many of the end-to-ends but failed miserably some others. The global success ratio for every predictions was near 70%. We identified two main reasons for these failures: jumps and change in trend.

4.2 Jumps

We call a "jump" a transaction which is orders of magnitudes larger than the average transaction for the end-to-end in question. Figures 4.1 and 4.3 in the thesis show how jumps occuring in the estimation period can affect our predictions. We can set a criterion to detect and remove jumps from our calculations. We chose to suppress every transaction that was larger than the average transaction by at least two standard-deviations. The effect was radical in some cases. Compare figures 4.2,4.4 to figures 4.1,4.3. Globally the

success ratio raised to 77%. Figures 4.5 to 4.14 of the thesis illustrate a few typical examples of the prediction vs observation results at the client level. However this is of no use when jumps occur in the prediction period. We must point out that any model would fail in that case since there is simply no way to predict these events with the available information. To make further progress on this problem, we need to investigate if these jumps can be traced to some assignable causes. Then we could devise a method to make these jumps "predictable" by using the new information.

4.3 Change of trend

Figures 4.13 and 4.14 are two examples of change of trend. The Kendall model does not allow change of trend since it would imply that the parameters λ and μ are not constant. However the model can still be useful in a situation where λ and μ are relatively constant during a period of time but then abruptly change their values and stay relatively constant for another period of time. There are several cases where the end-to-end seems to behave that way. The process would be Kendall during the two periods, the problem would be to identify at which time the change occurred. This is the kind of problem that one could solve using some non-parametric statistical test.

4.4 Aggregation levels

We have grouped the end-to-end clients on a building-to-building, city-to-city and province-to-province basis. However the results were somewhat disappointing. The global ratio of success decreased noticeably at each level of aggregation (See figures 4.15-4.17). However we have reasons to suspects that deterministic factors are influencing our results.

When we looked at "intra" city-to-city demand, that is the aggregation of all end-to-end in the same city, we came across an interesting phenomenon. There was a clear change of trend at roughly the same date for many cities (figures 4.18-4.20). Also this phenomenon was almost absent if we looked at city-to-city demand between two different cities (figure 4.21)

Also we have grouped the end-to-end clients according to the type of circuits. Figures 4.22 to 4.26 are typical examples of patterns that the demand grouped by circuit would follow. In one case the number of actuals increased almost linearly. In other cases the number of actuals decreased continuously.

In many cases we observed abrupt changes of trend, often at the same time of the year. This may actually be the reflection of a policy of the company rather than the reflection of a real change of demand. This point is crucial since it would mean that the process is deterministic, not stochastic.

5 Conclusion

We presented the Kendall model as an adequate representation of growth of point-to-point demand. We have shown a method to estimate the parameters of the process. We developped a simulation to test our parameter estimates and our predictions of the demand. We constructed an algorithm to widen our confidence interval to guaranty we had a 95% confidence interval at all time.

We applied our model to a real-life situation. We found that despite jumps and changes of trend in the demand, we could obtain a reasonable 77% global success ratio for end-to-ends at the client level with 20 or more transactions. We found that the situation deteriorated when aggregating the end-to-end clients to higher levels. We suspect that deterministic factors may play a significant role at these levels.

Bibliographie

[1] Lehmann, E.L. (1951). Consistency and unbiasedness of certain nonparametric tests; Ann. Math. Statist., 22, 165-179.

[2] Billingsley, P. (1968). Convergence of Probability Measures. Wiley, 253 p.p.

[3] Nucho, R.N. (1981). Transient Behavior of the Kendall birth-death process; applications to capacity expansion for special services. The Bell System Technical Journal, Vol 60, No. 1,57-87.

[4] Chouinard, A., and McDonald, D. (1985). Limit theorems for a rank test measuring randomness; Technical Report No. 51, Laboratory for Research in Statistics and Probability, Carleton University, 34 pp.

[5] Beaucaire C.,Krten J., Tardiolli L.,McDonald D., (1990). Provisioning a network; Bell Northern Research internal report.

[6] McDonald, D. (1991). On the asymptotics of randomness statistics; The Canadian Journal of Statistics, Vol. 19, No. 2, 209-217.