Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Analyse de courbes et application à la rhinomanométrie

Citta, Allen

Award date:
1988

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 02. May. 2026

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 1987-1988

Analyse de courbes et application à la rhinomanométrie

CITTA Allen

Mémoire présenté en vue de l'obtention du grade de Licencié et Maitre en Informatique

Ce mémoire est divisé en deux grandes parties: une partie théorique et une partie application de la théorie.

La première partie présente tout d'abord une introduction à la notion de rhinomanométrie, suivie des propositions de standardisation faites lors de divers rencontres internationales.

Ensuite, les erreurs introduites par les appareils et par le patient,lors de la prise de mesure, sont traitées tout en indiquant leur impact sur les résultats grâce à une représentation graphique de ces derniers.

Enfin, pour conclure cette première partie, une analyse plus fine des données sera faite, suivie par une introduction à la notion de distance et son application à l'élimination des erreurs introduites par le patient.

La deuxième partie du mémoire traite du développement d'une application permettant pour un patient la saisie des données, le transfert de ces données sur un traceur, l'élimination de mauvaises courbes et la détermination d'une courbe moyenne modélisant les courbes obtenues.

This report consists of two parts : a theoretical part and an application of this theory.

The first part presents firstly an introduction to rhinomometry, followed by proposals for standardisation made during international meetings.

Then, errors introduced by the apparatus and by the patient, during mesures, are dealt with, while showing at the same time, their impact on the results thanks to graphical representations of the latter.

Finally, to conclude this first part, a more refined analysis of the datas will be made, followed by an introduction to the notion of distance and its application to the process of elimination of errors introduced by patients.

The second part of the report deals with the developping of a software allowing for a given patient, the taking of mesures, the transfer of the results on a plotter, the elimination of "bad" curves and the modelisation of the result by a mean curve.

Je remercie Madame Monique NOIRHOMME-FRAITURE, Professeur aux Facultés Notre Dame de la Paix à Namur, d'avoir assuré la direction de ce mémoire.

Pour m'avoir accueilli durant mon stage à la Clinique Universitaire de Mont-Godinne, que le Docteur Bertrand trouve ici l'expression de ma reconnaisance.

Je tiens à remercier également Mr Thierry LORANT, de chez Siemens, pour l'aide qu'il m'a apporté lors du développement du logiciel.

Que Monsieur Daniel DUVIVIER soit ici remercié du temps qu'il m'a consacré pour l'apprentissage de l'utilisation des appareils rhinomanométrique, pour la prise de mesure et pour les tests du logiciel.

Enfin, je remercie tous ceux qui m'ont conseillé et encouragé pendant la rédaction du mémoire.

PLAN DU MEMOIRE

****

PREMIERE PARTIE

0. INTRODUCTION

Avant-propos
Plan
Liste des figures

1. CHAPITRE 1 : INTRODUCTION A LA RHINOMANOMETRIE 1

1.1 Définition 1
1.2 Techniques 1
1.2.1 La Rhinomanométrie Passive Antérieure 1
1.2.2 La Rhinomanométrie Active 2
1.3 Les appareillages 3

2. CHAPITRE 2 : SAISIE DE DONNEES ET EXPRESSION DES RESULTATS 4

2.1 La saisie des données 4
2.2 Expression des résultats 4
2.2.1 Standardisation en rhinomanométrie 4
2.2.2 Expression graphique du débit en fonction du gradient de pression 5

3. CHAPITRE 3 : SOURCES D'ERREURS ET IMPACT SUR LES RESULTATS 7

3.1 Introduction 7
3.2 Erreurs dues au matériel utilisé 7
3.2.1 Le tube de FLEISH et capteurs différentiels de pression 7
3.2.2 Le masque 7
3.2.3 Occlusion de la narine controlatérale 8
3.2.4 Emploi de la canule 8

Plan du mémoire

3.3 Erreurs introduites par le patient 9
3.3.1 La nervosité et le gène 9
3.4 Impact des erreurs sur les résultats 9
3.4.2 Impact des erreurs dues aux appareils sur les résultats 9
3.4.1 Impact des erreurs dues aux patients sur les résultats 12

4. CHAPITRE 4: TECHNIQUES D'ELIMINATION DES ERREURS 15

4.1 Introduction 15
4.2 Analyse plus fine des données 16
4.2.1 Définition de la notion de courbe croissante et de courbe décroissante 16
4.2.2 Analyse des courbes obtenues lors de la prise de mesure 17
4.2.3 Premier traitement des données 19
4.3 la théorie 19
4.3.1 La notion de distance généralisée 19
4.3.2 La théorie du T^2 de Hotelling 21
4.4 Application au rejet des courbes 21
4.4.1 Première solution 21
4.4.1.1 Présentation 21
4.4.1.2 Inconvénients 24
4.4.2 Deuxième solution 25
4.4.2.1 Présentation 25
4.4.2.2 Inconvénients 26
4.5 Techniques pour améliorer les performances de la solution choisie 27

DEUXIEME PARTIE

5. CHAPITRE 5 : DEVELOPPEMENT DU LOGICIEL 28

1. Introduction 28
2. Schema conceptuel et description des fonctions 29
3. Conception et justification d'une 41

architecture logique
3.1 Hiérarchisation en niveaux de modules 41
3.2 Découpe en modules 41
3.2.1 Contenu des niveaux 6, 5 et 4 42
3.2.2 Graphe représentant l'architecture logique 45
3.2.3 Explicitation des relations dans le graphe 46
3.3 Justification de la stratégie de modularisation suivie 47
3.4 Spécification externe des modules des niveaux 6, 5 et 4 48
3.4.1 Specification externe des modules de niveau 6 48
3.4.2 Specification externe des modules de niveau 5 48
3.4.2 Specification externe des modules de niveau 4 58
4. Conception abstraite des algorithmes des modules du niveau 6 70
4.1 Conception abstraite des algorithmes de M1 70
4.1.1 Conception abstraite des algorithmes de gestion patient 70
4.1.2 Conception abstraite des algorithmes de saisie et traitement de données 70
4.1.3 Conception abstraite des algorithmes de transfert vers un traceur 71
4.1.4 Conception abstraite des algorithmes de visualisation courbes du patient courant 71
5. Conclusion 72
6. Bibliographie

ANNEXES

1. Annexe A
2. Listing des sources des programmes

3 Manuel de l'utilisateur

CHAPITRE 0

Introduction

****

0. Introduction

Le service O.R.L. de la Clinique Universitaire de Mont-Godinne possède un appareil rhinomanométrique permettant la prise de mesure et le transfert des données obtenues vers un traceur. L'informatisation de la rhinomanométrie a soulevé plusieurs problèmes: notamment le problème des erreurs, qui sont introduites lors de la prise de mesure, et le problème de stockage de ces données.

La première partie de cet ouvrage concerne toute la partie théorique du problème. On introduira la notion de rhinomanométrie, son objectif et les différentes techniques utilisées. Ensuite, on présentera la saisie des données et la présentation des résultats tout en se référant aux propositions de standardisation faites lors des rencontres internationales.

Le chapitre 3 de cette partie, traitera des erreurs. Pour chacune des sources d'erreurs, c'est à dire, les appareils et le patient, on explicitera la cause des erreurs, leur impact sur les résultats à l'aide de représentations graphiques.

Enfin, pour conclure cette première partie, on essayera de choisir une technique pour éliminer les mauvaises courbes. Pour ce faire, on fera une analyse plus fine des données, suivie d'une introduction à la notion de distance et son application à l'élimination des courbes. Deux solutions sont proposées ainsi que leurs avantages et leurs inconvénients, et les raisons du choix d'une solution.

La deuxième partie traite du développement d'une application permettant d'éliminer les mauvaises courbes. Le service O.R.L., ne possédant pas de service Informatique indépendant, avait à sa disposition très peu d'outils de développement et l'absence d'un gestionnaire d'écran et d'un gestionnaire du traceur n'a pas facilité la tache du développeur.

Pour clôturer le document, on indique les références bibliographiques qu'on a utilisées. On fournit entre-autres en annexe l'implémentation du logiciel développé, et le manuel utilisateur.

CHAPITRE 1

Introduction à la rhinomanométrie

****

1.1 Définition

La rhinomanométrie a pour but la mesure de la résistance des fosses nasales à l'écoulement de l'air. Cette résistance est une fonction complexe du débit, de la pression ainsi que du type d'écoulement.

L'écoulement est de 3 types :

a) laminaire
b) semi-turbulent
c) turbulent

1.2 Techniques

Pendant longtemps, l'emploi du miroir de GLATZEL a été la seule méthode clinique "objective" d'appréciation de la perméabilité des fosses nasales. Un miroir gradué en cercles concentriques étant placé sous les fosses nasales du patient, celui-ci est invité à expirer. L'observation du nombre de cercles concentriques atteints par la condensation de l'air expiratoire fournit une méthode grossière de l'obstruction nasale.

L'avènement de l'electronique a amené de nouvelles méthodes de mesures de résistance nasale. Les deux méthodes suivantes mesurent la relation existant entre le débit d'un courant d'air parcourant les fosses nasales et la variation de pression entre l'entrée et la sortie des fosses nasales.

1.2.1 La rhinomanométrie passive antérieure (R.A.P)

La rhinomanométrie passive mesure la variation de pression, ou gradient, entre l'entrée et la sortie des

fosses nasales pour un débit constant. Cette méthode est dite passive car elle utilise, pour provoquer le passage de l'air, une pompe aspirante ou foulante à débit constant.

Une fosse nasale est hermétiquement obturée par une canule reliée à un manomètre mesurant la pression (P_1) régnant dans le rhinopharynx, la narine controlatérale étant reliée à la pompe, le flux d'air parcourra donc cette fosse nasale, s'écoulant par la bouche. Un deuxième manomètre mesure la pression à l'entrée de cette fosse nasale (P_{nz}). Le passage de l'air à travers une structure tourmentée provoque une perte de charge; on aura donc $P_{nz} > P_1$, la différence entre ces deux pressions (δP) donne le gradient de pression pour un débit déterminé, pour une fosse nasale.

1.2.2 La rhinomanométrie active

La rhinomanométrie active mesure , comme la précédente, la relation entre le gradient de presion et le débit. Contrairement à la passive, dans la rhinomanométrie active, le débit varie constamment, le flux aérien étant ici assuré par la respiration du patient. Deux méthodes de rhinomanométrie active sont utilisées :

- la rhinomanométrie active antérieure (R.A.A) où le débit du courant aérien respiratoire est mesuré au niveau de la fosse nasale testée, tandis que le gradient de pression est mesuré entre la narine controlatérale et l'entrée de la fosse nasale testée,

- la rhinomanométrie active postérieure (R.A.P) où la mesure du gradient de pression se fait entre le rhinopharynx et la fosse nasale testée, la pression rhinopharyngée étant mesurée à l'aide d'un embout que l'on demande au patient de placer en bouche tout

en obturant artificiellement la narine controlatérale.

Le type de rhinomanométrie le plus fréquemment utilisé est la rhinomanométrie active antérieure.

1.3 Appareillages

La mesure du débit nasal se fait grâce à l'utilisation d'un pneumotachographe de FLEISCH relié à un électromanomètre différentiel. Le pneumotachographe est un tube divisé en fins canaux parallèles, réalisant une structure en nid d'abeilles, obligeant ainsi le flux aérien qui le traverse à adopter un régime d'écoulement laminaire; le débit est ainsi directement proportionnel à la pression $\delta P = KV$ où K est la résistance du pneumotachographe.

En mesurant les pressions à l'entrée et à la sortie du tube de FLEISCH, on obtient une valeur proportionnelle au débit. La mesure du gradient de pression se fait à l'aide d'un second électromanomètre. Les deux tensions électriques obtenues, après passage dans un amplificateur, peuvent être visualisées soit sur un oscilloscope, soit sur un écran, soit sur une table tracante donnant soit une mesure en fonction du temps δP et V, soit une courbe $V = f(\delta P)$.

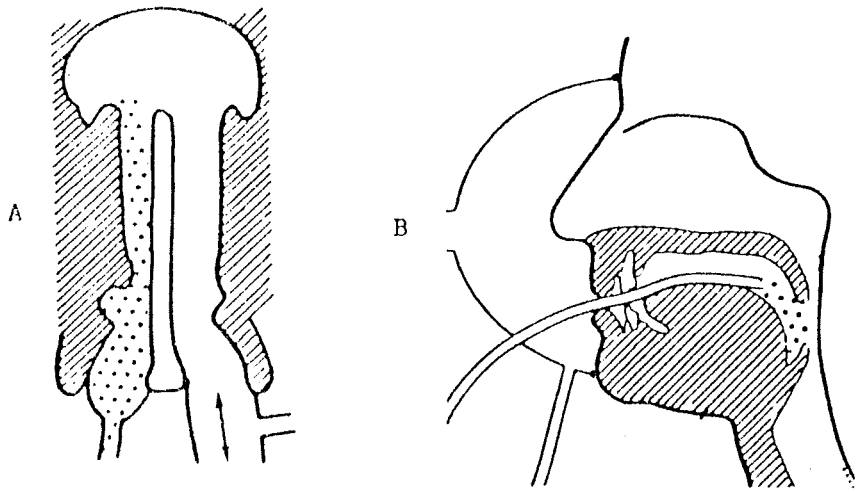


Fig 1: Méthode de mesure des pressions nasales postérieures (d'après Melon et Daele,1979)
A: par voie antérieure B: par voie postérieure

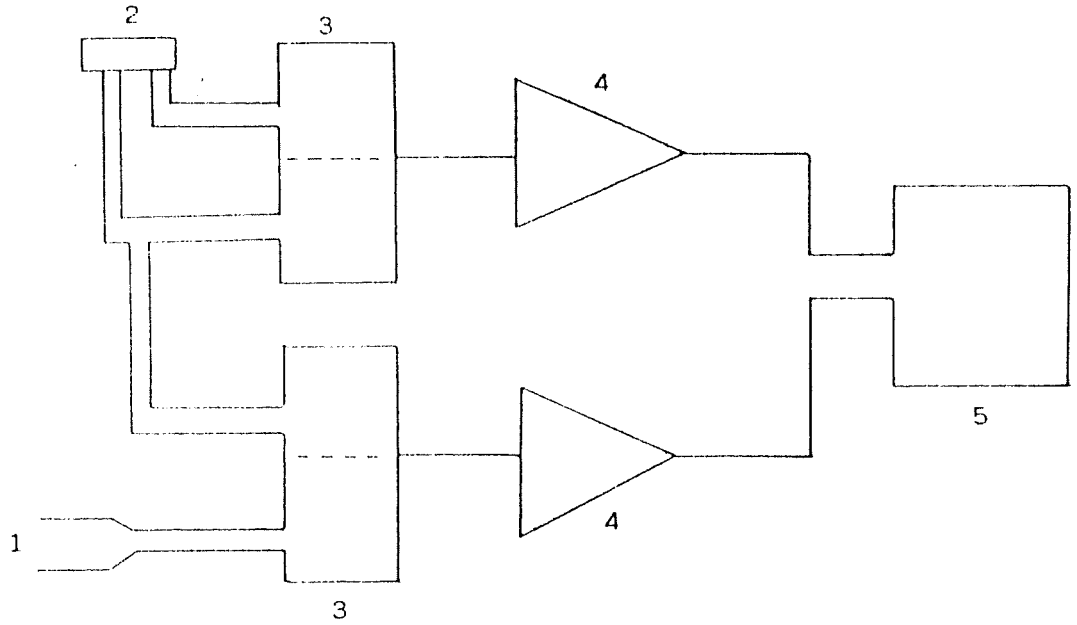


Fig 2: Appareillage utilisé en rhinomanométrie active
1 = prise de mesure, 2 = pneumotachographe, 3 = manomètre différentiels de pression, 4 = amplificateurs, 5 = table traçante ou oscilloscope

CHAPITRE 2

Saisie de données et expression des résultats

****

2.1 La saisie des données

Avant la saisie des données, le patient doit s'être reposé au moins 30 minutes. La saisie des données s'effectue avec le patient en position assise. La saisie se fera pendant une respiration normale du patient et plusieurs cycles respiratoires seront enregistrés.

Parfois plusieurs séries de mesures se font par patient. Ces mesures se font généralement dans des conditions différentes, par exemple après repos, après effort ...

2.2 Expression des résultats

2.2.1 Standardisation en rhinomanométrie

Lors d'une rencontre internationale des différents chercheurs en rhinomanométrie, certains standards ont été adoptés, notamment en ce qui concerne la représentation graphique des résultats des mesures (Clem84).

La représentation de type X-Y a été considérée comme le meilleur moyen d'enregistrement, car elle montre bien la relation entre le gradient de pression et le flux. Concernant la représentation graphique elle-même, les chercheurs ont recommandé la technique de l'image mirroir qui utilise les quatre cadrans du graphe. Cette technique permet de représenter les enregistrements des fosses nasales droites et gauches sur le même graphe en utilisant le 2e et le 4e cadrans pour la fosse nasale de gauche et le 1er et le 3e cadran pour celle de droite. Il a été aussi décidé que l'ordonnée représente le flux et l'abscisse le gradient de pression. L'unité adoptée pour le gradient de

pression (symbole δP) est le PASCAL, et celui pour le débit (symbole V) est cm^3/s.

Figure 1.
Mirror image technique using the four quadrants of the graph in active anterior rhinomanometry:
- inspiration: right of the flow-axis
- expiration: left of the flow-axis
- quadrant I-III: right nasal cavity
- quadrant IV-II: left nasal cavity
- resistance should be given at a fixed pressure of 150 Pascal or at radius 2 when using Broms' model

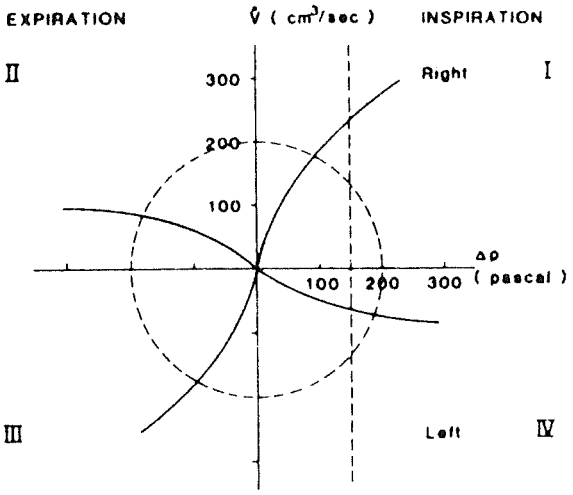


Fig 3:La technique de l'image mirroir utilisant les 4 cadrans du graphe en rhinomanométrie active antérieure :

- inspiration : à droite de l'axe de flux
- expiration : à gauche de l'axe de flux
- cadran 1 et 3 :fosse nasale droite
- cadran 2 et 4 :fosse nasale gauche

2.2.2 Expression graphique du débit en fonction du gradient de pression

L'inscription du débit en fonction de la pression donne une courbe en forme de sigmoïde. Cette courbe est d'autant plus inclinée sur l'axe du δP que l'obstruction nasale est importante (BERT85) (fig 4).

La sigmoïde peut être décomposée en trois segments, chacun correspondant à un type d'écoulement particulier :

la première portion montre une relation linéaire entre V et δP, ce qui correspond à un régime d'écoulement laminaire; $\delta P = KV$. La seconde portion signifie que l'écoulement devient semi-turbulent : la courbe obéit alors à la relation $\delta P = KV^n$, avec $1 < n < 2$. Enfin, le troisième segment signe un écoulement turbulent : $\delta P = KV^n$, avec n proche de 2 (fig 5)

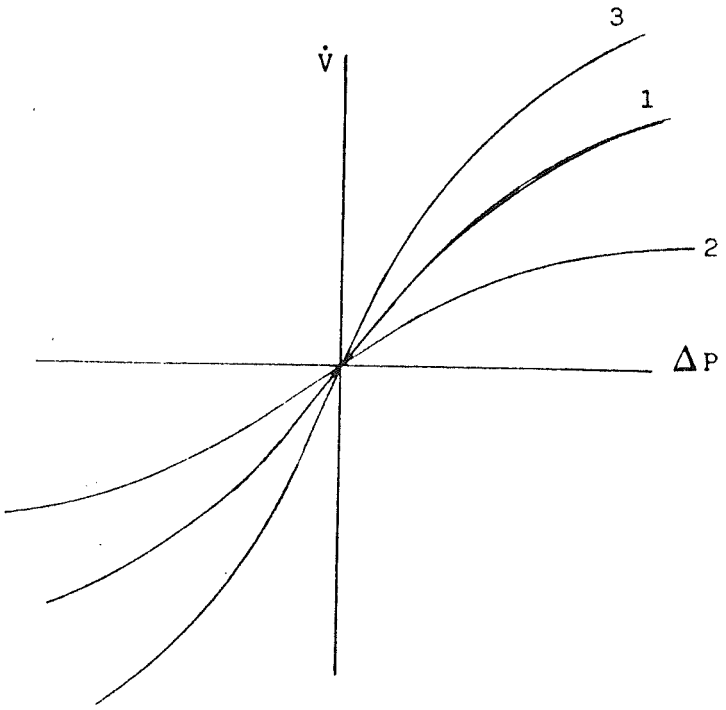


Fig 4 : Courbe débit-gradient de pression
1 : résistance nasale normale
2 : résistance nasale augmentée
3 : résistance nasale diminuée

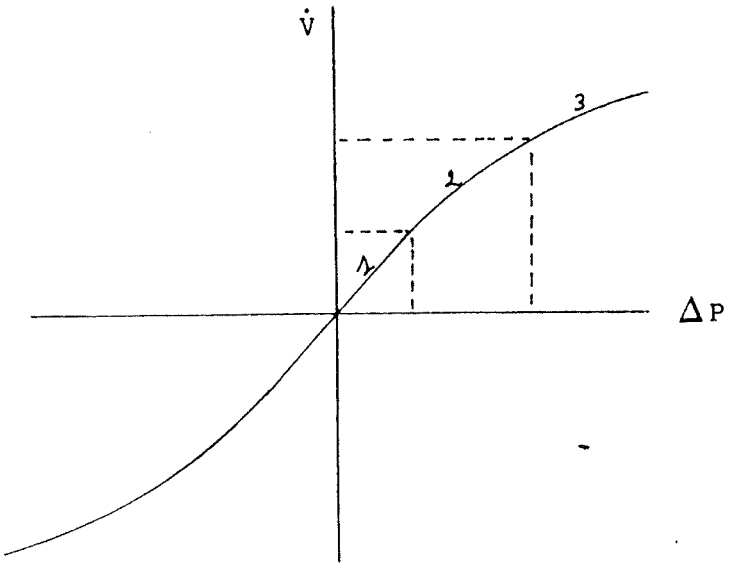


Fig 5 : Décomposition d'une courbe débit-gradient de pression en fonction du type d'écoulement
1 : écoulement laminaire
2 : écoulement semi-turbulent
3 ; écoulement turbulent

CHAPITRE 3

Sources d'erreurs et impact sur les résultats

****

3.1 Introduction

La prise de mesures ne s'effectue pas sans erreurs comme on peut s'en douter. Plusieurs facteurs introduisent des erreurs et donc modifient les résultats. Parmi ces facteurs, on notera les deux plus importantes :

a) le matériel utilisé
b) le patient

3.2 Erreurs dues au matériel utilisé

3.2.1 Le tube de FLEISCH et capteurs différentiel de pression

Tout instrument de mesure introduit une certaine erreur. Dans le cas qui nous préoccupe, on mesure deux choses qui sont fort affectées par les conditions atmosphériques*. Le tube de FLEISCH introduit une certaine erreur qui est doublée du fait qu'il existe une certaine inertie de la valve.

3.2.2 Le masque

En rhinomanométrie, on utilise plusieurs types de masques :

---

*Remarquons que dans la prise de mesure, on note le débit. Ce dernier est influencé par la température qui n'est pas la même à l'inspiration qu'à l'expiration. Le patient inspire l'air à la température ambiante et expire un air qui est à la température de son corps. De plus, l'air expiré a un taux d'humidité généralement supérieur à celui de l'air inspiré

- le masque d'anesthésie : recouvrant la pyramide nasale et la bouche, fermement appliqué sur la face par le patient. L'étanchéité est assurée par la pression exercée par le patient, ce qui n'est pas constant durant l'examen.

- le masque de plongée : fixé à l'aide d'une lanière élastique, l'étanchéité étant vérifiée au début de l'examen, on ne doit plus s'en occuper au cours de l'examen. L'emploi d'un masque ne recouvrant pas la bouche est préféré afin de pouvoir faire bouger l'embout buccal et obtenir un meilleur relâchement du voile du palais.

- le masque recouvrant entièrement le visage (type masque à gaz) : il a la plupart des avantages du masque de plongée avec l'inconvénient de ne pouvoir bouger le capteur de pression buccale. Le volume d'air du masque devra être aussi réduit que possible afin d'éviter un effet parasite d'inertie.

3.2.3 Occlusion de la narine controlatérale

L'occlusion de la narine controlatérale peut se réaliser par l'introduction d'un corps étranger (ex: un tampon d'ouate, un embout gonflable ...) dans la fosse nasale.

Toutes ces techniques se valent. On remarque cependant qu'ils peuvent entrainer une réaction vasomotrice de la muqueuse nasale conduisant à des résultats erronés.

3.2.4 Emploi de canules narinaires

La mesure du débit et du gradient de pression peut se faire directement à la sortie des fosses nasales, par l'introduction de canules narinaires. Cette méthode a l'avantage d'être très rapide et de pouvoir être réalisée par 99% des sujets.

L'introduction d'une canule provoque une déformation du vestibule nasal, ce qui peut poser des problèmes lorsque l'orifice narinaire joue un rôle prépondérant dans la résistance totale de la fosse nasale. De plus l'introduction profonde de la canule dans le vestibule nasal aurait tendance à augmenter la résistance nasale pour une narine large. Il faut également veiller à ce que le diamètre de la canule soit suffisant pour ne pas ajouter une résistance supplémentaire. Une surface d'ouverture de 50 mm^2, tant pour la canule que pour les tuyaux, est un minimum à respecter.

3.3 Erreurs introduites par le patient

Le patient est aussi à la base de certaines erreurs qui sont dues principalement à la nervosité et au gène que procurent les différents instruments de mesure.

3.3.1 La nervosité et la gène

Il est assez évident que devant les appareils qu'il a en face de lui, le patient ressente une certaine nervosité. De plus, le fait de devoir respirer avec des canules dans ses narines le gène énormément. Cet inconfort est d'autant plus prononcé chez les jeunes patients.

Ces deux agents, la gène et la nervosité, provoquent généralement un certain changement dans le rythme de respiration du patient.

3.4 Impact des erreurs sur les résultats

Comme on l'a fait remarqué précédemment, des erreurs sont introduites aussi bien par les appareils de mesures, que par les patients. Ces erreurs ont des effets importants sur les résultats, effets qui peuvent mieux être expliqués à l'aide de représentations graphiques des résultats.

3.4.1 Impact des erreurs dues aux appareils sur les résultats

Dans les différentes publications des chercheurs, on remarque une chose très importante: elles donnent toutes une courbe unique qui représente le cycle respiratoire, et cette courbe passe par l'origine (fig 6)

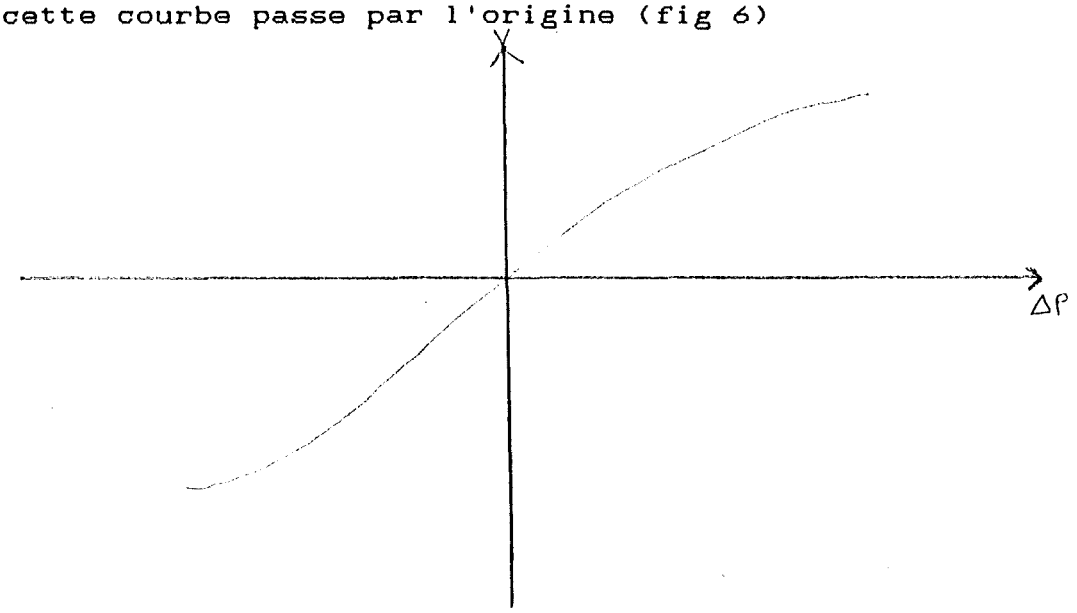


Fig 6

Dans les mesures faites sur des patients, on a pu constater que ce n'est pas le cas. La courbe représentant le cycle respiratoire ressemble à une courbe d'hystérésis (fig 7), où on voit nettement la différence entre les deux constituants du cycle respiratoire, c'est à dire l'inspiration et l'expiration. Cette courbe d'hystérésis ne passe pas nécessairement par l'origine.

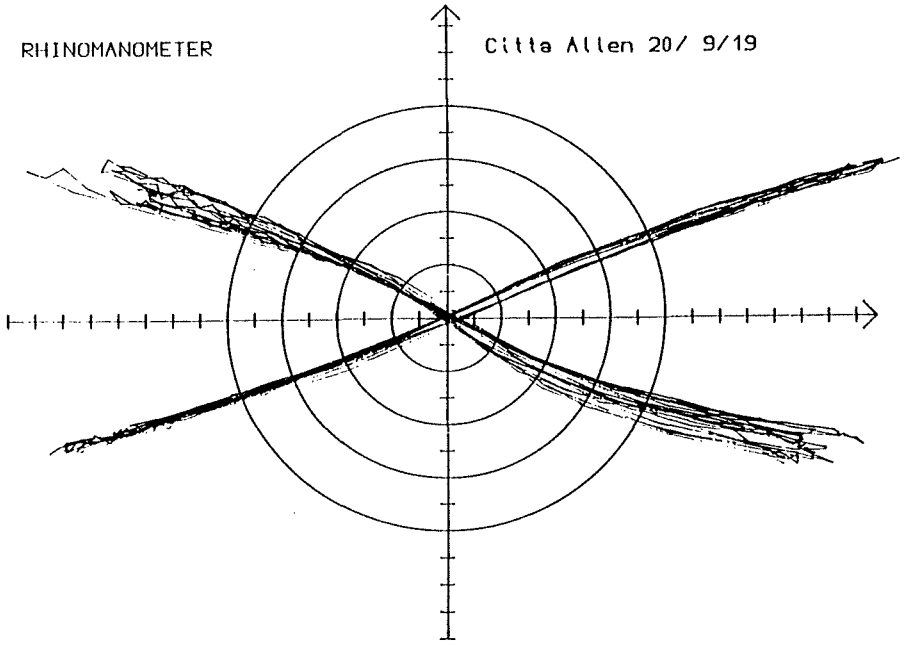


Fig 7

De cette constatation, on peut se poser certaines questions, notamment : est-ce que la forme de la courbe est due aux erreurs introduites par les appareils de mesures (et par l'influence des effets atmosphériques), ou est-elle due à un problème physiologique ou encore est-ce un mélange des deux?

Une réponse n'a pu être donnée qu'après avoir fait certaines observations sur une série de mesures : la forme de la courbe est un mélange des deux.

Dans un premier temps, comme on l'a déjà fait remarquer, l'utilisation d'un tube de FLEISCH introduit une certaine erreur qui est due à l'inertie de la valve. Les valeurs de δP et V ainsi obtenues sont donc entachées d'une certaine erreur qu'on peut calculer de deux manières :

- a) A partir des caractéristiques de la valve, calculer l'erreur introduites par cette dernière

- b) Avec un patient en "bonne santé", prendre une série d'observations, et voir l'intersection des courbes ainsi obtenues avec les axes.

Il arrive parfois qu'on obtienne une courbe d'hystérésis de type A (fig 8). Dans un premier temps, aucune explication plausible n'a pu être donnée pour expliquer ce genre de courbe, car les erreurs introduites par les instruments de mesures ne varient généralement pas autant. Mais, on a remarqué que le patient qui avait produit la courbe A, après avoir toussé, avait une courbe qui ressemble à B.

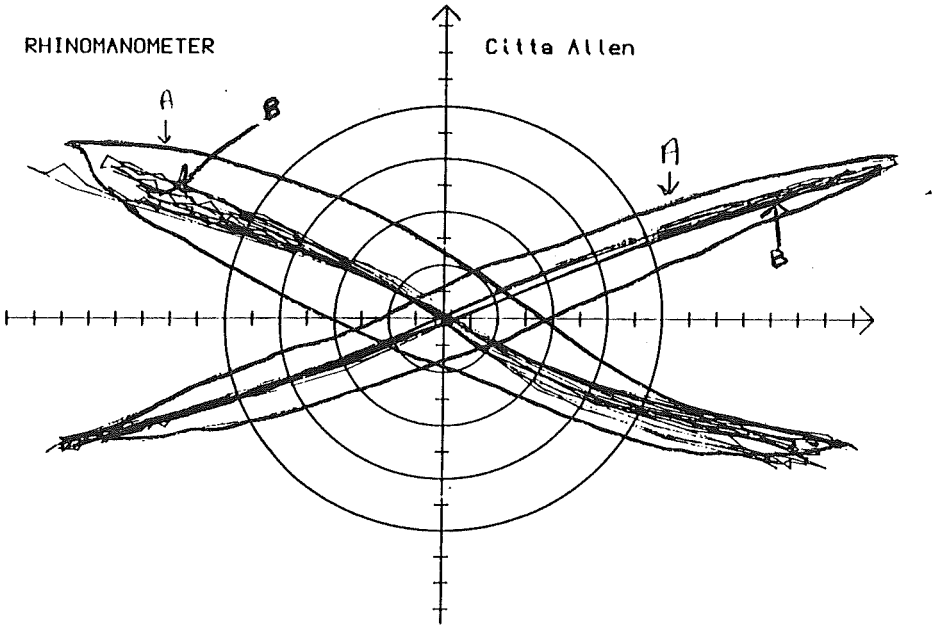


fig 8

De cette constatation, on est arrivé à la conclusion que le fait d'avoir toussé, a dégagé les bronches du patient et a ainsi diminué la résistance des bronches de ce dernier, d'où la courbe B.

3.4.2 Impact des erreurs dues aux patients sur les résultats

La gène et le nervosité provoque chez le patient une respiration irrégulière, voire accélérée. Ceci affecte beaucoup sur les mesures, et on note fréquemment la présence de courbes avec une forme bizarre A (fig 9) dans les graphiques faits à partir des mesures.

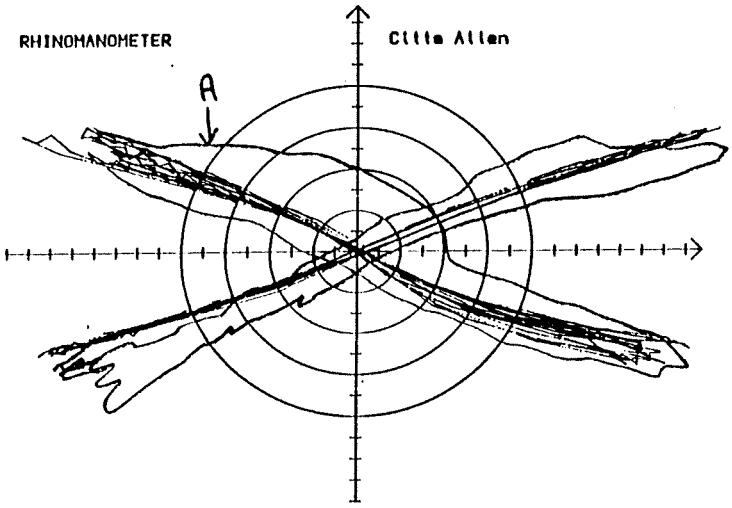


Fig 9

A priori, (c'est ce qui était pratiqué jusque maintenant), le fait de décider qu'une courbe soit mauvaise ou non se faisait de manière non scientifique. Celà se faisait à l'oeil en la comparant à la forme générale des autres courbes saisies sur le patient.

Ayant vu les sources d'erreurs dans la prise de mesure et leur impact sur les résultats, on essaiera de trouver des méthodes pour les éliminer.

CHAPITRE 4

Techniques d'élimination des erreurs

****

4.1 Introduction

Ayant pour objectif l'implémentation d'un outil clinique en rhinomanométrie, il serait intéressant de pouvoir, dans une certaine mesure, éliminer les erreurs introduites lors de la prise de mesure. Les erreurs dues au matériel ne seront pas traitées, et ceci pour les raisons suivantes :

- les caractéristiques du matériel permettant de calculer les erreurs introduites par ce dernier sont généralement difficiles à obtenir des fournisseurs
- le calcul des erreurs introduites par les conditions atmosphériques requiert la présence d'autres instruments de mesure et alourdit ainsi la procédure de prise de mesure
- quant aux autres erreurs dues aux matériels, elles sont difficiles à calculer

Ceci nous conduit donc à ne traiter que les erreurs dues au patient. Ce traitement peut se faire manuellement ou de façon automatique, et c'est cette dernière manière qu'on choisira. Pour ce faire, on fera une analyse plus fine des données, et on présentera un premier traitement puis on introduira la notion de distance généralisée, ensuite on présentera deux solutions pour éliminer les mauvaises courbes avec leur base théorique.

4.2 Analyse plus fine des données

4.2.1 Definition de la notion de courbe croissante et de courbe décroissante

Le service ORL étant aussi un service axé sur la recherche, il a été décidé de considérer la courbe d'un cycle respiratoire obtenue lors des mesures, comme deux courbes distinctes: une croissante indiquant l'inspiration et l'autre décroissante indiquant l'expiration. La raison étant la possibilité d'existence d'une double courbe contrairement à la courbe unique théorique. Dès lors, tout traitement informatique des données obtenues doit prendre en compte ces deux type de courbes.

Soit

- V_{max} : la plus grande valeur de V lors du cycle respiratoire complet courant.
- V_{min} : la plus petite valeur de V lors du cycle respiratoire complet courant.
- V_{max_pred} : la plus grande valeur de V lors du cycle respiratoire complet précédent.
- V_{min_pred} : la plus petite valeur de V lors du cycle respiratoire complet précédent.

Une courbe croissante (notée CC) est définie comme une courbe obtenue à partir des valeurs de la suite:

$(V_{min_pred}, V1, V2, \ldots, Vn, V_{max})$
où $Vi+1 \geq Vi \quad \forall\ 1 \leq i \leq n$
et $V_{min_pred} \leq V1$ et $Vn \leq V_{max}$

Une courbe décroissante (notée CD) est définie comme une courbe obtenue à partir des valeurs de la suite:

$(V_{max_pred}, V1, V2, \ldots, Vn, V_{min})$
où $Vi+1 \leq Vi \quad \forall\ 1 \leq i \leq n$
et $V_{min} \leq Vn$ et $V1 \leq V_{max_pred}$

On notera CX quand on parlera indifféremment de courbes CC ou CD.

La décision de l'appellation croissante ou décroissante d'une courbe n'est pas arbitraire mais due à la respiration du patient.

Comme, à priori, les courbes croissantes et décroissantes sont différentes et forment un hystérésis, et qu'on ne connait pas la raison exacte de ce phénomène, on traitera séparément les courbes CC et CD.

4.2.2 Analyse des courbes obtenues lors de la prise de mesure

Lors de la prise de mesure avec les appareillages décrits en 1.3, le patient respire avec les canules dans ses fosses nasales. Ceci implique donc une fluctuation continue des valeurs de δP et de V.

Lors de la prise de mesure, on prend des valeurs discrètes de δP et V, grâce à l'introduction d'un interface entre l'ordinateur et le pneumotacographe. Cette interface a pour objectif de prendre des échantillons de δP et V, chaque n milli-seconde où n est une valeur fixée avant la prise de mesure. De la sorte, on arrive à une discrétisation de δP et V. Dans la fig 10, chaque point représente une valeur de $(\delta P, V)$ ainsi obtenue.

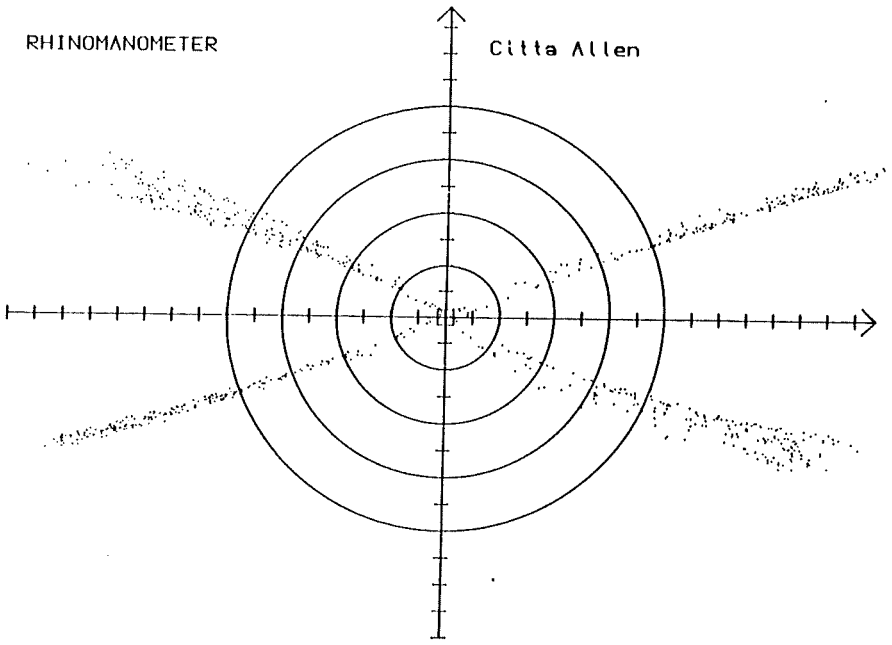


Fig 10

Entre deux échantillons consécutifs pris, les valeurs de δP et de V ne sont pas connues avec précision, et doivent être interpolées.

Une remarque importante à noter sur la fig 10 est que la distance entre deux points consécutifs n'est pas fixe. Ceci est dû au fait qu'on a un échantillonage temporel et non un échantillonage spatial.

Le premier segment de la courbe (2.2.2 et fig 5) représente un phénomène rapide , ce qui explique la relativement grande distance entre les points pris dans cette région. Le troisième segment représente lui un phénomène lent, ce qui explique le grand nombre de points les uns proches des autres à cet endroit et une distance assez petiteentre deux échantillons consécutifs. Le deuxième segment présente quant à elle une distance moyenne.

Pour les besoins du traitement des données obtenues lors de la prise de mesure, une petite application permettant la saisie des données sur un patient et la visualisation des courbes CX obtenues a été faite. Les

courbes tracées à l'écran sont obtenues en traçant une droite entre chaque deux points consécutifs obtenus lors de la prise de mesure. On a noté qu'en utilisant une interpolation linéaire, pour des valeurs de n (temps entre deux échantillonage) de l'ordre de 3 à 120 ms, on préservait la forme de la courbe, et de plus, on pouvait visualiser les "mauvaises" courbes.

4.2.3 Premier traitement des données

Pour qu'un quelconque traitement statistique puisse donc être appliqué à ces échantillons, il serait intéressant d'avoir pour une abscisse donnée δPi, une série d'ordonnées (Vij) correspondant aux valeurs de V pour les différentes courbes obtenues. Les valeurs de V pour un δPi donné seront calculées grâce une fonction d'interpolation linéaire. Pour celà, on choisira deux points consécutifs dans chaque courbe $(\delta Pk, Vk)$ et $(\delta Pk+1, Vk+1)$ tel que $\delta Pk < \delta P1 < \delta Pk+1$. (Remarquons qu'au cas où pour une courbe donnée, il existe un $(\delta Pk, Vk)$ telle que $\delta Pk = \delta Pi$, on n'interpole pas.)

Pour résumer cette étape de traitement, on peut dire ceci : on est passé d'une courbe continue à une suite de points $(\delta Pi, Vi)$ obtenue lors de la prise de mesure et enfin, à un nombre plus restreint de points après l'étape d'interpolation de valeurs de V pour un petit nombre de valeurs de δP, identiques d'une courbe à l'autre.

4.3 La notion de distance généralisée, la théorie et son application

4.3.1 La notion de distance généralisée

Pour la comparaison de moyennes, on définit une mesure globale des différences existant entre 2 vecteurs de moyennes.

Pour deux populations, de moyennes respectives

$$m_1^T = (m_{11}\ m_{12}\ \ldots\ m_{1p})$$

$$m_2^T = (m_{21}\ m_{22}\ \ldots\ m_{2p})$$

on peut employer

- a) $D1 = \sqrt{(\Sigma_i (m_{1i} - m_{2i})^2)}$ pour $i = 1 \ldots p$
  (la distance EUCLIDIENNE)

- b) $D2 = \frac{1}{\sqrt{p}} D1$

- c) $D3 = \sqrt{(\Sigma_i ((m_{1i} - m_{2i})/\sigma_i)^2)}$ pour $i = 1 \ldots p$
  (distance entre les points moyens obtenus après standardisation par les écarts types)

- d) $D4 = \frac{1}{\sqrt{p}} D3$

L'inconvénient de ces mesures est qu'elles ne dépendent aucunement des corrélations pouvant exister entre les variables (ici le nombre de lignes se trouvant dans m_1 ou m_2). D1 et D3 augmentent indéfiniment avec le nombre de variables tandis que D2 et D4, dans certain cas peuvent diminuer lorsqu'on augmente le nombre de variables, ce qui est, tout aussi illogique. D'où on utilise de préférence la distance généralisée de MAHALANOBIS.

Pour deux populations de mêmes variances et covariances Σ, la distance généralisée de MAHALANOBIS peut être définie comme suit :

$$D5 = \sqrt{((m_1 - m_2)' \Sigma^{-1} (m_1 - m_2))} \text{ ou}$$

$$D5 = \sqrt{(\delta' \Sigma^{-1} \delta)} \text{ où}$$

δ désigne le vecteur des différences des moyennes

$$\Sigma_{i,j} = \Sigma (m_{ik} * m_{kj}) - (\Sigma m_{ik})(\Sigma m_{jk})/(n - 1) \quad (n = 2)$$

4.3.2 La théorie[1] du T^2 de HOTELLING

Soit u un vecteur aléatoire à p composantes, de distribution multinormale de moyenne μ et de matrice de variance-covariance Σ ($N_p(\mu,\Sigma)$)

Soit D une matrice définie positive symmétrique de distribution de WISHART indépendante de u de degré de liberté f > p et de matrice associée Σ ($W_p(f,\Sigma)$)

Alors par définition, $u'(D/f)^{-1}u$ est une VA de T^2 de HOTELLING avec f degrés de libertés.

Dans le cas particulier où E(u) = 0, la distribution "nulle" du T^2 de HOTELLING ne dépend pas de Σ, en général inconnu. On peut montrer qu'alors

$\frac{(f - p + 1)T^2}{p * f}$ a une distribution F à p et (f-p+1) degrés de libertés

4.4 Application[2] au rejet des courbes

Deux solutions ont été proposées pour éliminer les mauvaises courbes. Les deux solutions ainsi que leur avantages et leur inconvénients seront présentées dans les paragraphes qui suivent.

4.4.1 Première solution

Soient n courbes de p points chacun. Chaque courbe i est définie par les p valeurs (x_{1i}, ..., x_{pi}) qui sont les p ordonnées correspondant à p abscisses identiques pour toutes les courbes[3].

---

[1]Kshirsagar p126, théorèmes 1 et 2

[2]Rappel : On essaie d'éliminer les mauvaises courbes dues à des erreurs introduites par le patient.

[3]En fait ceci revient à considérer les p ordonnées comme étant un point dans un espace à p dimensions

On souhaite définir une distance entre une de ces courbes et l'ensemble des n_1 autres où $n_1 = n - 1$ et $n1 > p$. De plus on voudrait avoir un test qui décide si cette distance est significativement différente de zéro.

Pour la suite on fera les hypothèses suivants :

- pour une abscisse donnée, la variance des points des différentes courbes est la même, que la courbe soit bonne ou non.

- la covariance entre deux points d'une même courbe est constante d'une courbe à l'autre, et que les courbes observées sont indépendantes l'une de l'autre.

- les observations $(x_{i1}, \ldots, x_{ip})$ sont extraites d'une population multinormale $Np(\mu,\Sigma)$ de vecteur moyenne μ et de matrice de variance-covariance Σ avec i = 1 .. n1

On considère

$x = (x_1 \ldots x_p)'$ le vecteur moyen des n - 1 courbes

y = la $n^{ième}$ courbe $(y_1 \ldots y_p)'$

X la matrice p * (n - 1) des n - 1 courbes formées chacune des p observations. Par construction, les colonnes de cette matrice sont statistiquement indépendantes.

$S_n = X(I - E/n_1)X' == XX' - XEX'/n_1$ la matrice des sommes des carrés et sommes de produits liée à X ($\Sigma = S_n/n_1$)

Alors $x - y$ est un vecteur multinormal de moyenne $\mu - \tau$ et de matrice de variance-covariance $(n_1 + 1)\Sigma/n_1$ où τ est la moyenne stochastique du vecteur y.[4]

S_{xx} a une distribution de Wishart de degré de liberté $n - 2$ et de matrice associée Σ. De plus, S_{xx} et x sont indépendantes, dès lors S_{xx} et $x - y$ sont indépendantes. On peut donc utiliser la définition du T^2 de HOTELLING en prenant pour u et D les variables :

$$u = \sqrt{((n_1/(n_1 + 1))(x - y))} \text{ et}$$

$$D = S_{xx}$$

On obtient

$T^2 = (x - y)'S_{xx}^{-1}(x - y)(n1(n1 - 1)/(n1 + 1))$ à $n - 2$ degrés de liberté.

Si les courbes xi et y sont issues d'une même population, alors

$$F = \frac{(n_1 - p)\ T^2}{p(n_1 - 1)}$$

a une distribution F de degrés de liberté p et $n1 - p$[5].

En conclusion, on peut prendre comme distance entre x et y la valeur de T^2. Pour autant que $n - 1 > P$, on testera l'hypothèse nulle

H_0 : $\mu = \tau$ en prenant comme région critique

$R_c = \{w : (n - p - 1)T^2/DIV > Q_F(1 - \alpha; p, n-p-1)\}$ où :

- Q_F est le quantile de la distribution F à p et n-p-1 degrés de liberté et

[4]Pour la demonstration, se référer à l'annexe A.

[5]Tshirsagar, p143

- DIV = $p(n_1 - 1)$.

Autrement dit, on rejettera une courbe chaque fois que $\frac{(n - p - 1)T^2}{p(n_1 - 1)}$ est trop grand par rapport au quantile de F

4.4.2 Inconvénients

Cette méthode comporte plusieurs inconvénients majeurs, dus pour la plupart aux caractéristiques des courbes.

Pour chaque patient, au maximum 10 courbes, de p points chacune, sont prises. La matrice S est donc une matrice p * p avec $p \leq 8$. Le premier inconvénient est qu'il faut inverser une telle matrice 10 fois. Ceci est un prix très cher à payer, surtout quand on désire un traitement en temps réel, ce qui est le cas.

Un deuxième inconvénient, est que pour le vecteur x, on fait la moyenne des n1 courbes , qui comporte des bonnes courbes et des mauvaises courbes. Cette moyenne sera d'autant plus éloignée de la moyenne des bonnes courbes si les mauvaise courbes présentes se trouvent d'un seul côté(fig 11). Le rejet de toutes les courbes souhaitées (les mauvaises courbes) n'est donc pas garanti.

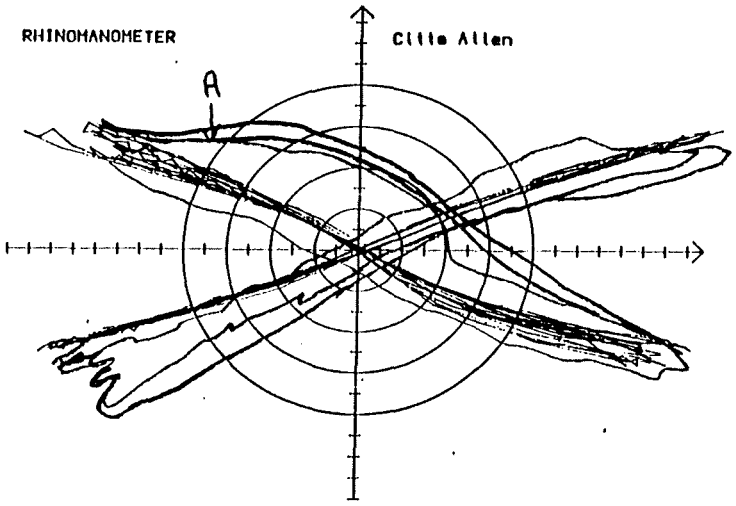


Fig 11

4.4.2 Deuxième solution

4.4.2.1 Présentation de la solution

On se propose de procéder par une agglomération hiérarchique.

Soient

- n le nombre total de courbes prises
- n1 le nombre de courbes proche l'une de l'autre
- p le nombre de points pris sur une courbe

L'algorithme général de cette méthode est la suivante :

1 : Rechercher les 2 courbes les plus proches en utilisant la distance EUCLIDIENNE.

2 : Faire la moyenne de ces 2 courbes

3 : Calculer la distance des autres courbes par rapport à la courbe moyenne obtenue (en utilisant cette fois le distance de MAHALANOBIS) et prendre celle qui est

la plus proche de cette courbe moyenne. Aller à 2 si le nombre de courbes agglomérées est inférieur à un nombre arbitraire, par example 7 si on a 5 abscisses, car le T^2 de HOTELLING et le test F de SNEDECOR nécessite la condition $n > p + 1$.

4 : Tester de façon statistique les courbes restantes dès que $n1 > p$ à l'aide du test de F comme expliqué dans le première méthode.

4.4.2.2 Avantages et Inconvénients

Un inconvénient de cette solution, est qu'il existe une possibilité que deux mauvaises courbes soient les plus proches au stade 1. Cette possibilité est assez infime en général, mais néanmoins, on doit être conscient de son existence.

Généralement, dix courbes sont prises sur chaque patient. Etant donné que $n - 1 > p$, et que $n = 10$, on a donc $p \leq 8$. Pour que le test T^2 de Hotelling soit d'une quelconque utilité, on prend $p = 5$ et on commence à tester la présence de mauvaises courbes quand on a déjà aggloméré 7 courbes. Un deuxième inconvénient de cette méthode provient de là, car on fait l'hypothèse que dans les 7 premières courbes agglomérées, il n'y a pas de mauvaises courbes.

Le troisième inconvénient est le calcul de S (estimateur de Σ) et de son inverse. Ce calcul se fait chaque fois sur les n1 courbes déjà agglomérées, ce qui est assez lourd.

L'avantage de cette solution est qu'elle nous assure de ne rejeter que les mauvaises courbes, pourvu qu'il n'y en ait pas de trop.

4.4.3 Techniques pour améliorer les performances de la solution choisie

Pour pallier au troisième inconvénient (4.4.2.3) de la deuxième solution, on pourrait se demander pourquoi ne pas estimer Σ une fois pour toute sur l'ensemble des n courbes, et ainsi on n'aurait qu'à inverser la matrice S une seule fois. Cette solution n'est pas correct car elle ne garantit plus l'indépendance entre n et D comme le requiert le T^2 de HOTELLING.

Une deuxième solution est de considérer le cas particulier où les covariances sont nulles ou sont ignorées.

Soit

- x la moyenne calculée sur m courbes
- y la courbe à tester
- $ms^2_k = \sum_{j=1}^{m} X^2_{k,j} - (\sum X_{k,j})^2/m$, $k^{ème}$ élément diagonal

de la matrice S_m estimée sur m courbes

alors on a la formule simplifiée :

$$D6^2 = (y-x)^T S^{-1}_m (y-x) = \Sigma (y_k - x_k)^2 / ms^2_k$$

$$= (y-x)^T \begin{vmatrix} 1/ms^2_1 & & & \\ & 1/ms^2_2 & & \\ & & 1/ms^2_3 & \\ & & & 1/ms^2_p \end{vmatrix} (y-x)$$

Le test de rejet devient $R > Q_F(1-\alpha; p; m-p)$

où $R = \Sigma ((y_k - x_k)^2 m(m-p))/(ms^2_k(m+1)p)$.

DEUXIEME PARTIE
DEVELOPPEMENT DU LOGICIEL

1. INTRODUCTION

INTRODUCTION

****

1. Introduction

Cette deuxième partie du mémoire traite du développement du logiciel qui implémente les principes fondamentaux introduits dans la première partie.

La méthodologie appliqué est la suivante :

- déterminer les besoins de l'utilisateur, et faire les spécifications fonctionnelles
- dans un deuxième temps, concevoir et justifier l'architecture logique proposée. Pour ce faire, on donnera une première hiérarchisation en niveaux de modules, suivie de la découpe en modules, tout en justifiant la stratégie de modularisation choisie
- pour chacun des modules, donner la spécification externe des modules de traitements et des modules de données
- pour terminer on donnera les algorithmes abstraits des modules fonctionnels.

Des procedures de test n'ont pas été conçues car :

- les modules de niveau 6 sont d'une simplicité qui nécessite nullement le besoin d'être testé pour vérifier leur validité et
- les modules de niveau cinq et quatre font appel à des algorithmes déjà testés.

2. SCHEMA CONCEPTUEL ET DESCRIPTION DES FONCTIONS

SCHEMA CONCEPTUEL

Définition des entités et des associations

****

ENTITE : **PATIENT**

DEFINITION :

Toute personne subissant (ou ayant subie) une prise de mesure dans le service ORL de l'hôpital.

IDENTIFIANTS :

id1 : **nom_pat** : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

ENTITE : **FIC_DON**

DEFINITION :

Un ensemble de données obtenues lors de la prise de mesure sur un patient donné dans le service ORL.

IDENTIFIANTS :

id1 : **nom_fic_don** : le nom de l'ensemble de données

ATTRIBUTS :

Une suite de données composées de couple (x,y) où

- x : la pression
- y : le débit

CONTRAINTES :

la suite (x,y) doit être dans le même ordre que lors de la prise de mesure.

ASSOCIATION : **CORRESPOND**

DEFINITION :

Associe un patient à ses fichiers de données obtenues lors des prises de mesure.

ATTRIBUTS :

date : la de la prise de mesure
type : les conditions de prise de mesure

DESCRIPTION DES FONCTIONS

****

FONCTION : **IMPRESSION**

OBJECTIFS :

Pour un patient donné, transférer les données obtenues lors de la prise de mesure sur un traceur.

MESSAGES D'ENTREES :

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : la tracée des données
sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant

E1 : sortie1 est produite

E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X

FONCTION : **IMPRESSION_COURANT**

OBJECTIFS :

Pour un patient donné, transférer les données obtenues lors de la dernière date de prise de mesure sur un traceur.

MESSAGES D'ENTREES :

nom_pat : le nom du patient

pre_pat : le prénom du patient

date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : la tracée des données

sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X

FONCTION : **VISUALISATION**

OBJECTIFS :

Permet, pour un patient donné, de visualiser à l'écran, toutes les courbes obtenues pour ce dernier. Toutes les courbes obtenues à partir des différentes prises de mesure seront visualisées.

MESSAGES D'ENTREES

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : la visualisation des différentes courbes à l'écran
sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X

FONCTION : **VISUALISATION_COURANT**

OBJECTIFS :

Permet, pour un patient donné, de visualiser à l'écran, les courbes obtenues pour ce dernier à la dernière date de prise de mesure.

MESSAGES D'ENTREES

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : la visualisation des différentes courbes à l'écran
sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X

FONCTION : **INTRODUCTION_PATIENT**

OBJECTIFS :

Permet l'introduction d'information concernant un patient et ce dernier devient le patient courant.

MESSAGES D'ENTREES

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : le patient est admis et devient le patient courant
sortie2 : le patient devient le patient courant

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : le patient est admis et devient le patient courant
sortie2 : le patient devient le patient courant

EFFET SUR LE S.I. :

Les changements possibles sur le S.I. dûs à cette fonction, sont les E(i) suivants :

E3 : une occurence de "patient" est crée.

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X
E3		X

FONCTION : **PRISE_DE_MESURE**

OBJECTIFS :

Permet la prise de mesure de données sur un patient.

MESSAGES D'ENTREES :

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : une visualisation à l'écran des données obtenues sous forme de courbes
sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

Les changements possibles sur le S.I. dûs à cette fonction, sont les E(i) suivants :

E3 : une occurence de "**correspond**" est crée.
E4 : une occurence de "**fic_don**" est crée.

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1		X
E2	X	
E3	X	
E4	X	

FONCTION : **TRAITEMENT_DES_DONNEES**

OBJECTIFS :

Permet pour un patient donné, d'éliminer les "mauvaises" courbes obtenues lors de la prise de mesure et éventuellement de modéliser les courbes restantes.

MESSAGES D'ENTREES

nom_pat : le nom du patient
pre_pat : le prénom du patient
date_nais : la date de naissance du patient

CONTRAINTES :

Le patient identifié par (**nom_pat**, **pre_pat**, **date_nais**) correspond bien à un patient existant.

MESSAGES DE SORTIES :

sortie1 : la tracée des courbes obtenues après élimination des mauvaises courbes
sortie2 : le patient est inconnu

EFFET SUR LE S.I. :

REGLES DE TRAITEMENT :

Denotons par C(i) et E(i) les conditions et effets suivants :

C1 : l'identifiant introduit correspond à un patient existant
E1 : sortie1 est produite
E2 : sortie2 est produite

Les règles R(i) à respecter sont illutrées par la table suivante :

	R1	R2
C1	V	F
E1	X	
E2		X

3. CONCEPTION ET JUSTIFICATION D'UNE ARCHITECTURE LOGIQUE

CONCEPTION ET JUSTIFICATION D'UNE ARCHITECTURE LOGIQUE

****

3.1 Hiérarchisation en niveaux de modules (Premier squelette d'architecture)

DECOUPE EN NIVEAUX :

NIVEAU 1 : Modules de l'Operating System (OS)
NIVEAU 2 : Modules Utilitaires
NIVEAU 3 : Modules de Contrôle
NIVEAU 4 : Modules de gestion des Entrées-Sorties (E/S)
NIVEAU 5 : Noyau Fonctionnel (Tri, Extraction, Modules de Gestion d'Ecrans et du Traceur, Modules Mathématiques
NIVEAU 6 : Modules Fonctionnels

Remarque : Nous ne serons concerné que par les modules des niveaux 4, 5, et 6, les modules 1, 2, et 3 étant pré-définis par le concepteur du système sur lequel tournera l'application.

3.2 Découpe en modules

Ici, on se propose d'expliciter le contenu des niveaux 4, 5 et 6, puis de construire le graphe représentant l'architecture logique complète tout en explicitant les relations dans le graphe. Enfin, on donnera la spécification externe de chaque module.

3.2.1 Contenu des niveaux 6, 5 et 4

NIVEAU 6 : Modules Fonctionnels

NOM	FONCTIONS PRINCIPALES
M1	Introduction_patient
	Prise_de_mesure
	Traitement_des_donnees
	Visualisation
	Visualisation_courant
	Impression
	Impression_courant

NIVEAU 5 : Noyau Fonctionnel

MODULE MATHEMATIQUE

```
interpol(tab,n)
inverse_mat(n)
fill_cvar_h(table,nx,n,matvar,h)
fill_cvar(table,nx,n,matvar)
agglomération_hierarchique(distance,hierarchie,ptr,tab
  y)
```

MODULE ECRAN

NIVEAU 4 : Modules de Gestion des Entrées-Sorties

ACCES_RHINO :

```
open_rhino()
close_rhino()
start_rhino(n)
read_rhino(x,y)
```

ACCES_TRACEUR :

P_OPEN_PLOTTER
P_CLOSE_PLOTTER
P_RESET(n)
P_SCALE(sx,sy)
P_PEN_SEL(n)
P_PEN_UP
P_PEN_DOWN
P_COORD(x,y)
P_PLOT
P_LINE(x1,y1,x2,y2)
P_FONT(n)
P_CHAR(str)
P_CIRCLE(r,angle1,angle2)

ACCES_ECRAN :

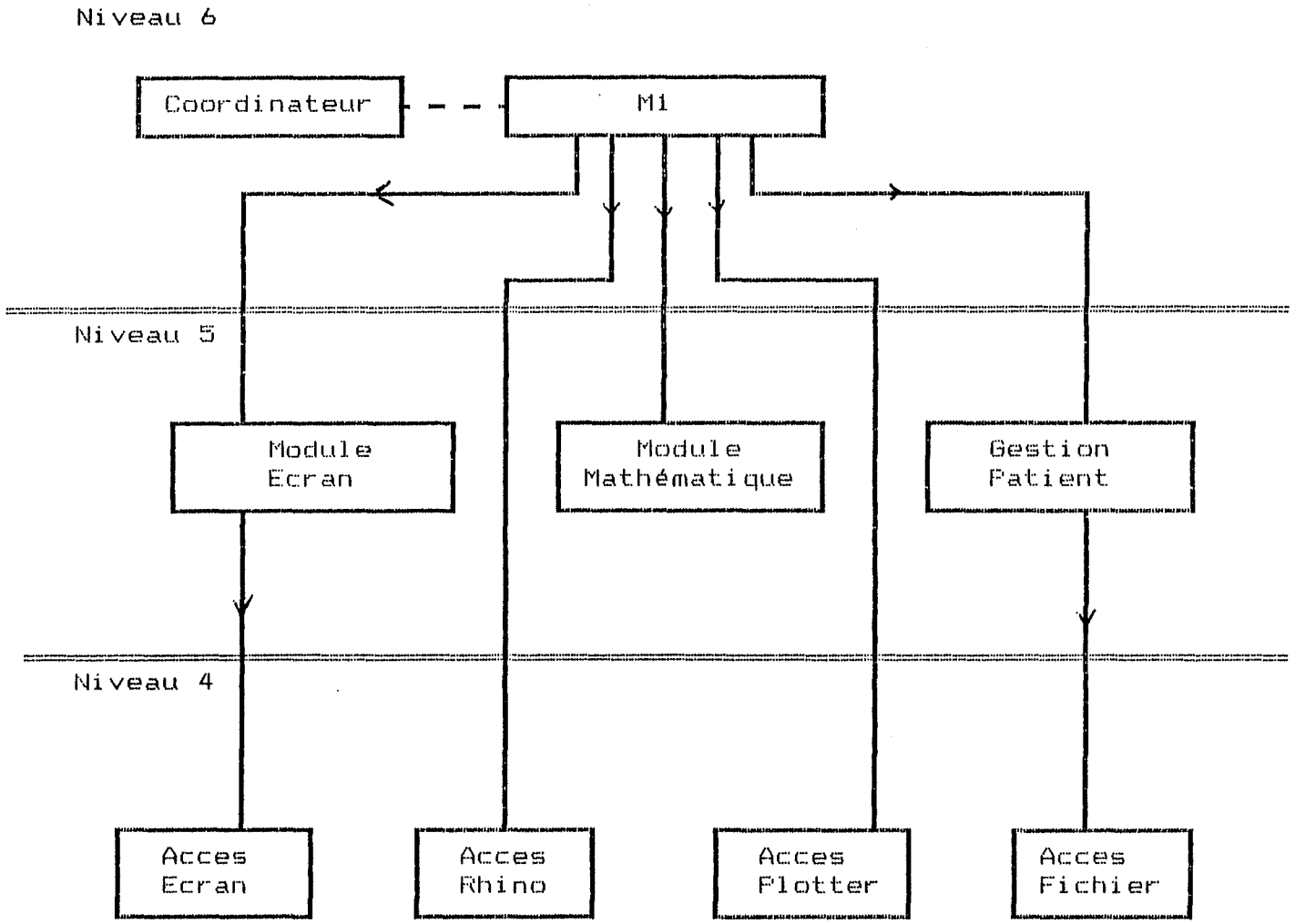
G_RESET
G_GRON
G_GROFF
G_CLRWIN
G_TSTSCRN
G_CURSON
G_CURSOFF
G_CURSHOME
G_CURS1POS(y,x)
G_CURS2POS(y,x)
G_CUR1SADV(dy,dx)
G_CURS2ADV(dy,dx)
G_DEFTYPE(a)
G_DEFCOLOR
G_DEFBLIN(a,b)
G_COLWIND(a)
G_COLORS(a,b)
G_COLFOREG(a,b)
G_COLBACK(a,b)
G_DRAW(a,y,x)
G_DRAWVSP(a)

```
G_DRAWVECT(a,y,x)
G_LINE(a,y1,x1,y2,x2)
G_DRAWRECT(a,y1,x1,y2,x2)
G_DRAWCIRC(a,y1,x1,y2,x2)
G_FILLRECT(a,y1,x1,y2,x2)
G_ZOOM(a)
G_DRAWCAR(a,b,c)
G_GREAT(a)
G_INIT54
```

ACCES_FICHIER

```
recherche_patient(ident)
ajout_patient(patient)
maj_patient(patient)
list_patient()
ajout_correspond()
```

3.2.2 Graphe représentant l'Architecture Logique



3.2.3 Explicitation des relations dans le graphe

---------	APPEL	Le coordinateur lance (exécute) le scénario implémenté grâce à ses ordres d'appel
----->----	UTILISE	Définition : A utilise B SSI le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B.

3.3 Justification de la stratégie de modularisation suivie

En ce qui concerne les modules fonctionnels, les fonctions utilisant les mêmes ensemble de données ont été regroupées dans un même module. Ce regroupement a aussi inclu la fonction "introduction_patient" car dans l'application qui nous préoccupe, la notion de patient est une notion importante. De plus, un module "gestion patient" n'a pas été fait car l'application a été pensée comme étant un sous-ensemble d'une plus grande application qui aurait un module "gestion patient".

Dans le but d'offrir un interface utilisateur convivial, le module de gestion ECRAN (niveau 5) a été conçu. Ce module regroupera les fonctions d'affichage écran et de saisie écran. Un module mathématique se chargera de toutes les fonctions ayant trait à des calculs statistiques.

Les modules de niveau le plus bas de la découpe choisie ont été classés dans le niveau 4. Un gestionnaire d'écran et un gestionnaire du traceur étant absent, le développement des deux modules **accès_écran** et **accès_plotter** s'est avéré nécessaire. On retrouvera aussi dans ce niveau les modules **accès_rhino**, et **accès_fichier**. Ces modules servent d'interface aux modules des niveaux supérieurs. De plus, ils permettent de cacher des informations ayant trait au matériel. Ceci permet donc une modification rapide de ces modules lors d'un éventuel transfert sur un matériel différent.

La stratégie poursuivie permet ainsi

- de LOCALISER des données nécessaires à l'exécution d'un module particulier
- de CACHER les données non nécessaires à la vue de ce module
- de CENTRALISER les modifications à l'intérieur des modules

- de SIMPLIFIER le niveau 6 en reportant les problèmes plus loin au niveau 5.

3.4 Spécification externe des modules des niveaux 6, 5 et 4

3.4.1 Spécification externe des modules de niveau 6

OBJ : Objectifs
ARGS : Arguments
PROP : Propriétés
RES : Résultats

M1 Module de gestion de données des patients

OBJ : Pour un patient donné, permettre la prise de mesure et le traitement des données tout en permettant la possibilité d'un éventuel transfert de ces données vers un traceur et de plus ce module doit permettre la visualisation à l'écran des données obtenues lors des prises de mesures sur ce dernier.

ARGS : ---

PROP : cfr "contraintes" de Description des fonctions

RES : cfr "Msg de sortie et Effet sur le S.I" de Description des fonctions

3.4.2 Spécification externe des modules du niveau 5

1 Définition des structures de données

Toute les fonctions des modules de niveau 5 et 6 requièrent, la définition des structures de données qui suivent. Ces dernières doivent être définies dans un fichier entête "**files.h**", qui doit être inclu au début de chacun des modules de ce niveau.

```
#define TEMP1       "/usr/tmp/RHINO_1"
#define TEMP2       "/usr/tmp/RHINO_2"
#define TEMP1DX     "/usr/tmp/RHINO_1.IDX"
#define TEMP2DX     "/usr/tmp/RHINO_2.IDX"
#define CHOIX1DX    "/usr/tmp/CHOIX_1.IDX"
#define CHOIX2DX    "/usr/tmp/CHOIX_2.IDX"
#define SOMM1SUM    "/usr/tmp/SOMME_1.sum"
#define SOMM2SUM    "/usr/tmp/SOMME_2.sum"

#define CHAR          0
#define INT           1
#define ALPHABET      2
#define ALPHANUM      3
#define ERROR         -1
#define ABORT         12
#define OK            13
#define TRUE          1
#define FALSE         0
#define MAX_INTX      100
#define NBRE_COURBES  20
#define POINT_CURVES  100
#define HIGH_VALUE    9999
#define MAX_DIM       50
#define INTERPOL      3
#define SUPERIEUR     1
#define INFERIEUR     -1
#define EGAL          0
#define BOOLEAN       int
#define SIZE_REC      (2 * sizeof(short))
#define EPSILON       0.0001
```

DEFINITION DE LA STRUCTURE DE DONNEES RHINO

```
typedef struct _rhino
{
  short y;
```

- **y** représente la valeur de V obtenue lors de la prise de mesure

short x;
- **x** représente la valeur de δP obtenue lors de la prise de mesure

} s_rhino;

DEFINITION DE LA STRUCTURE DE TYPE S_LINEXX

```
typedef struct _line
{
```

int c;
- **c** = indice, dans le tableau, à partir duquel commence les courbes de type CC

int d;
- **d** = indice, dans le tableau, à partir duquel commence les courbes de type CD

} s_line;

```
typedef struct _lines
{
  int indice;
  - indice = indice à partir duquel commence les
    courbes de type CC dans le tableau d'index

  int nbre;
  - nbre = nombre de courbes de type CC dans le
    tableau d'index

  int nbrex;

} s_lines;
```

DEFINITION DE TYPE POUR LA MATRICE DES X

```
typedef struct _x
{
  short   x;
  - la valeur de δP

  int     nbre;
  long    sx;

} s_x;


typedef struct _tablex
{
  s_x    tabx[2][NBRE_COURBES + 1];
  -  i = 0 tabx[i][j] représente pour une courbe donnée
    CC l'ensemble des valeurs de δP de cette dernière.
  -  i = 1 tabx[i][j] représente pour une courbe donnée
    CD l'ensemble des valeurs de δP de cette dernière.

  int    first_column;
```

```
  int    last_column;

} s_tablex;
```

DEFINITION DE TYPE POUR LA MATRICE DES Y

```
typedef struct _ltable
{
  int   first_column;
```

- **first_column** = indice indiquant la première colonne à partie duquel se trouve la première valeur de V dans **column[]**

```
  int  last_column;
```

- **last_column** = indice indiquant la dernière colonne où se trouve la dernière valeur de y dans **column[]**

```
  short column[NBRE_COURBES + 1];
```

- tableau contenant les valeurs des ordonnées **Vij** pour une courbe i

```
} s_ltable;


typedef struct _table
{
  s_ltable line[NBRE_COURBES + 1];
```

- **line[j]** représente les valeurs de **Vij** pour une courbe i

```
} s_table;
```

DEFINITION DE TYPE POUR LA MATRICE DE VARIANCE/COVARIANCE ET DE R

```
typedef struct  _covarl
```

```
{
  double c[NBRE_COURBES + 1];
  - d[j] représente pour une courbe de type CC, les
    variances et covariances de celle-ci par rapport à
    la courbe j

  double d[NBRE_COURBES + 1];
  - d[j] représente pour une courbe de type CD, les
    variances et covariances de celle-ci par rapport à
    la courbe j

} s_covar1;


typedef struct _covar
{
  s_covar1 vc[NBRE_COURBES];
  - vc[i] représente pour la courbe i les variances et
    covariances de celle-ci par rapport aux autres
    courbes

} s_covar;
```

DEFINITION DE TYPE DE LA MATRICE DES DISTANCES

```
typedef struct _distance
{
  double d2[NBRE_COURBES + 1][NBRE_COURBES + 1];
  - chaque élément d2[i,j] de la matrice représente la
    distance entre la courbe i et la courbe j, les 2
    courbes étant bien sur du même type
     si i > j d2[i,j] de la matrice représente la
    distance entre la courbe i et la courbe j, (
    courbes de type CD)
     si i < j d2[i,j] de la matrice représente la
```

```
    distance  entre la courbe i et la courbe j, ( courbes
      de type CD)

  } s_distance;
```

DEFINITION DES STRUCTURES DE FICHIERS

```
  typedef struct _filename
  {
    short  max_x;
    - la valeur maximale de δP pour l'ensemble des
      courbes CX

    short  min_x;
    - la valeur minimale de δP pour l'ensemble des
      courbes CX

    short  max_y;
    - la valeur maximale de V pour l'ensemble des
      courbes CX

    short  min_y;
    - la valeur minimale de V pour l'ensemble des
      courbes CX

    short  min_x1_x2;
    short  max_x1_x2;

    char y[12];
    - nom du fichier des données

  } s_filename;


  typedef struct _f_pointer
  {
    long  offset;
```

```
        - déplacement par rapport au début du fichier de
          données de la nouvelle courbe CX

        char  type;
        - indique le type de la courbe
          'C' pour une courbe de type CC
          'D' pour une courbe de type CD

        int   nombre;
        - nombre de couples (δP,V) composant cette courbe

        int   no_courbe;
        - le numéro de la courbe

        short max_x;
        -  la valeur de δP pour laquelle on a la valeur
          maximale de V pour la courbe

        short min_x;
        -  la valeur de δP pour laquelle on a la valeur
          minimale de V pour la courbe

        short max_y;
        - la valeur maximale de V pour la courbe

        short min_y;
        - la valeur minimale de V pour la courbe

      } s_f_pointer;
```

DEFINITION DE TYPE POUR LA STRUCTURE HIERARCHIE

```
      typedef struct _num_distance
      {
        int     numero;
        - numéro de la courbe x qu'on vient d'aggglomérer
```

```
    double  distance;
    - la distance de la courbe numéro par rapport à la
      moyenne des autres courbes déjà agglomérées avant
      elle

} s_num_distance;

typedef struct _hierarchie
{
    s_num_distance  c[NBRE_COURBES + 1];
    - représente l'ordre dans laquelle les courbes de
      type CC ont été agglomérés

    s_num_distance  d[NBRE_COURBES + 1];
    - représente l'ordre dans laquelle les courbes de
      type CD ont été agglomérés

} s_hierarchie;
```

DEFINITION DE STRUCTURE POUR UNE VARIABLE DE TYPE INCONNU

```
typedef struct _variable
{
    int   type;
    - type de la variable à traiter

    union
    {
        int   ival;
        - définit une variable de type entière

        char  cval;
        - définit une variable de type caractère

        char  *pval;
        - définit une variable de type string
```

```
    ) uval;

) s_variable;
```

2 Spécification externe des modules du niveau 5

MODULE MATHEMATIQUE

interpol(tab,n) :

OBJ :

calcule la valeur de y, obtenu lors d'une interpolation avec une fonction d'ordre **n**-1, pour une valeur donnée de x. L''ensemble des couples (x,y) de départ se trouve rangé, par ordre croissant de x, dans les éléments d'indice ≥ 1 de **tab**.

inverse_mat(n) :

OBJ :

inverse une matrice carré d'ordre **n**

PROP :

- la matrice à inverser doit être d'un ordre maximal de NBRE_COURBES
- la matrice à inverser doit être d'abord transféré dans une matrice appelée mat1, qui doit donc être définie en externe
- la matrice résultante se trouve dans une matrice mat2 d'un ordre maximal de NBRE_COURBES

fill_cvar_h(table,nx,n,matvar,h) :

OBJ :

calcule la matrice de variance/covariance pour les vecteurs de **table**, chacun de **nx** composants, dont l'indice se trouve dans **h** (c'est à dire les courbes déjà agglomérées). Le résultat se trouvera dans **matvar**.

fill_cvar(table,nx,n,matvar) :

OBJ :

calcule la matrice de variance/covariance des **n** premiers vecteurs, de **nx** composants chacun, de **table**. Le résultat se trouvera dans **matvar**.

agglomération_hierarchique(distance,hierarchie,ptr,taby) :

OBJ :

fait l'agglomération hiérarchique des courbes CC et CD, avec comme matrice de distances de départ (distance EUCLIDIENNE) la matrice pointée par **distance**, le résultat se trouvera dans la structure pointée par **hierarchie**. **taby** est un pointeur vers une structure de type s_table qui contient l'ensemble des valeurs de y pour chacune des courbe. **ptr** est un tableau de type s_lines. Chacun de ses éléments déterminera le nombre d'abscisses prises pour les courbes et le nombre de courbes.

MODULE ECRAN

Ce module regroupe les fonctions suivantes :

- Affichage d'un écran
- Saisie des informations associées à cet écran.

Ce module étant dépendant des outils utilisés lors du développement du logiciel, on ne donnera pas plus de détail sur ses spécifications. Il est important, tout de même, de noter que toutes les variables contenant des informations provenant des écrans de saisie doivent être déclarées globalement.

2.2.3 Spécification externe des modules du niveau 4

NIVEAU 4 : Modules de Gestion des Entrées-Sorties

ACCES_RHINO :

Toutes les procédures permettant l'accès au device rhino font les hypothèses suivantes :

- il existe une entrée "**/dev/rhino**" dans le système associé au device rhino
- la séquence du début d'échantillonage est "**st--\r**" où '--' représente le nombre de milli-secondes entre chaque échantillonage
- la séquence de fin d'échantillonage est "**ha\r**"
- la séquence de prise de d'echantillon est "**\r**"

open_rhino() : ouvre le device rhino
close_rhino() : ferme le device rhino
start_rhino(n) : initialise le device rhino pour prendre des échantillons chaque **n** ms
read_rhino(x,y) : les entiers pointés par **x** et **y** sont garnis des valeurs obtenues lors de la prise de mesure

ACCES_TRACEUR :

Toutes les procédures permettant l'accès au device plotter font les hypothèses suivantes :

- il existe une entrée "**/dev/ife1600**" associé au device plotter dans le système

Les primitives suivantes ont été construites à partir des spécifications du traceur et ont été implémentées sous forme de macro dans le fichier entête "plotter.h"

P_OPEN_PLOTTER : ouvre le device plotter
P_CLOSE_PLOTTER : ferme le device plotter
P_RESET(n) : réinitialise le traceur selon la valeur de n :

n = 0 : le traceur n'écrit plus rien, la plume est mise au coin supérieure droite et aucun paramètre n'est réinitialisé

n = 1 : le traceur n'écrit plus rien, la plume est mise au coin supérieure droite et le traceur est mise en attente

n = 2 : les paramètres prennent leur valeur initiales

P_SCALE(sx,sy) : l'échelle des x est fixée à 1 : **sx**, et l'échelle des y est fixée à 1 : **sy**

P_PEN_SEL(n) : choisi la plume numéro **n** où **n** est compris entre 0 et 7 inclus

P_PEN_UP : déplacement vertical vers le haut de la plume du traceur

P_PEN_DOWN : déplacement vertical vers le bas de la plume du traceur

P_COORD(x,y) : stocke les coordonnées (x,y) sans déplacement de la plume. Les coordonnées (x,y) deviennent la position courante

P_PLOT : déplace la plume et la positionne à la position courante

P_LINE(x1,y1,x2,y2) : trace une ligne de (**x1**,**y1**) à (**x2**,**y2**) sur le traceur

P_FONT(n) : détermine la fonte utilisé selon la valeur de **n** (0 <= **n** <= 5). Pour la fonte standard ascii **n** = 0

P_CHAR(str) : imprime la chaine de caractère **str** à la position courante

P_CIRCLE(r,angle1,angle2) : trace un cercle ou un arc centré à la position courante de rayon **r**, avec comme angle de départ par rapport à l'horizontal **angle1**, et angle finale **angle2**

ACCES_ECRAN :

Les primitives suivantes ont été construites à partir des spécifications du terminal sur lequel on travaille et ont été implémentées sous forme de macro dans le fichier entête "**9754.h**"

```
/* |480                                    */
/* |  Y                                    */
/* |0,0                    X               */
/* +------------------------------ 640     */
```

G_RESET :
OBJ : fait un reset du terminal

G_GRON :
OBJ : met le terminal en mode graphique

G_GROFF :
OBJ : met le terminal en mode normal

G_CLRWIN :
OBJ : Fait un "clear" de la fenêtre courante

G_CURSON :
OBJ : Rend le curseur visible

G_CURSOFF :
OBJ : Cache le curseur

G_CURSHOME :
OBJ : positionne le curseur au coin inférieur gauche

G_CURS1POS(y,x) :
OBJ : positionne le curseur 1 au coordonné (y,x)

G_CURS2POS(y,x) :
OBJ : positionne le curseur 2 au coordonné (y,x)

G_CUR1SADV(dy,dx) :
OBJ : déplacement relatif du curseur 1 de **dy** et **dx**

G_CURS2ADV(dy,dx) :
OBJ : déplacement relatif du curseur 2 de **dy** et **dx**

G_WINDHOME :
OBJ : la position de la fenêtre est (0,0)

G_DEFTYPE(a) :
OBJ : défini la forme de la ligne pour les vecteurs et les rectangles
Les options standard sont et 1
11111111111111111111111.. 0
11111111111111111111111.. 1
11001100110011001100110.. 2
11110000111100001111000.. 3
11100100111001001110010.. 4
11111110001110001111111.. 5

```
11111111101010101111111.. 6
11111100111111001111100.. 7
11111100110011001111100.. 8
```

G_DEFCOLOR(a,r,g,b) :

OBJ : défini une couleur

- **a** 0-7 numéro de la couleur normale
  10-17 numéro de la couleur clignotante
- **r** 0-7 quantité de rouge
- **g** 0-7 quantité de vert
- **b** 0-3 quantité de bleu

G_DEFBLIN(a,b) :

OBJ : défini la couleur qui doit clignoter

- **a** 0-8
  0-7 couleur
  8 pause
- **b** 0-999 temps entre chaque clignotement
  unité : 1/16 ms

G_COLWIND(a) :

OBJ : défini la couleur d'une fenêtre
Cette commande fontionne en mode graphique et en mode alphanumérique

a = 0 fills the window with main color
1 delete the main color into window
2 delete all the screen at actual level
3 delete all the G_COLORS in all level

G_COLORS(a,b) :

OBJ : Défini la couleur principal et la couleur secondaire

- **a** = numéro de la couleur principale
- **b** = numéro de la couleur secondaire

G_COLFOREG(a,b) :

OBJ : défini le "foreground color" des caractères en mode graphique

a = attribut
- 0 = normale
- 1 = haute intensité
- 4 = souligné
- 7 = video inversé

b = 0-7 couleur principale
10-17 pour les couleurs 8 -15

G_COLBACK(a,b)

OBJ : défini le "background color" des caractères en mode graphique

a = attribut
- 0 = normale
- 1 = haute intensité
- 4 = souligné
- 7 = video inversé

b = 0-7 couleur principale
10-17 pour les couleurs 8 -15

G_DRAW(a,y,x)

OBJ : trace un point à la position courante

a = 2 trace un point au curseur 1
3 efface un point au curseur 1
4 inverse un point au curseur 1
5 trace un point au curseur 2
6 efface un point au curseur 2
7 inverse un point au curseur 2

G_DRAWVSP(a) :

OBJ : défini la position de départ pour le traçé des droites

a = 0 la position courante du curseur est la position de départ

1 la position de fin de droite est la position de début de la prochaine droite à tracer
3 tracer un vecteur avec le curseur 1

G_DRAWVECT(a,y,x) :
OBJ : trace des droites
a = 0 la position courante du curseur 1 est la position de départ
3 tracer un vecteur avec le curseur 1

G_LINE(a,y1,x1,y2,x2)
OBJ : trace une droite de (**y1,x1**) à (**y2,x2**)
a = 3 tracer un vecteur avec le curseur 1

G_DRAWRECT(a,y1,x1,y2,x2)
OBJ : dessine un rectangle
a = 0 dessine le contour d'un rectangle
1 efface le contour d'un rectangle mais pas le remplisage
4 rempli avec --------
5 rempli avec des points
6 rempli avec des lignes verticales
7 horizontales
8 des "uphash"
9 des "downhash"

G_DRAWCIRC(a,y1,x1,y2,x2) :
OBJ : dessine un cercle de centre (**x1,y1**), de rayon **y2**
a = 0 dessine un circle
1 efface un circle

G_FILLRECT(a,y1,x1,y2,x2) :
OBJ : rempli un rectangle ayant comme coin supérieur gauche les coordonnées (**y1,x1**) et coin inférieur droite les coordonnées (**y2,x2**)
a = 0 rempli le rectangle avec un G_PAINT plein

G_C_RED :
OBJ : défini la couleur rouge

G_C_GREEN :
OBJ : défini la couleur verte

G_C_YELLOW :
OBJ : défini la couleur jaune

G_C_BLUE :
OBJ : défini la couleur bleue

G_C_MAGENTA :
OBJ : défini la couleur magenta

G_C_CYAN :
OBJ : défini la couleur cyan

G_C_WHITE :
OBJ : défini la couleur blanche

G_INIT54 :
OBJ : initialise le terminal 54

ACCES_FICHIER

Toutes les procédures permettant l'accès aux fichiers font les hypothèses suivantes :

- le fichier patient se trouve dans le repertoire /usr/gast/rhino/jojo et est défini dans le macro PATIENT

Les definitions de structures de données suivantes se trouve dans le fichier "struct.h" :

```
#define  CONNECTIVITE 10
```

- le nombre maximum d'occurences de CORRESPOND qui peut exister pour un patient donné

DEFINITION DE TYPE POUR LES ELEMENTS DU SCHEMA CONCEPTUEL

```
typedef struct _identification
{
  char nom[35];
  - nom du patient

  char prenom[35];
  - prenom du patient

  char date_nais[12];
  - date de naissance du patient

} s_identification;

typedef struct _correspond
{
  char date[12];
  - date de prise de  mesure

  char type;
  - type de mesure

  int    num;
  -   numero identifiant le fichier de données du
    patient pour la date et de type type

} s_correspond;

typedef struct _patient
{
  int nombre_mesure;
```

- nombre de mesures faites pour le patient et auxquelles on peut accéder

s_identification ident;
- identification du patient

s_correspond correspond[CONNECTIVITE];
- les occurences de l'association CORRESPOND du patient

} s_patient;

recherche_patient(ident)
OBJ : recherche le patient identifié par **ident**, et renvoie 0 si patient existe et 1 sinon

ajout_patient(patient)
OBJ : ajoute une occurence de **patient** en fin de fichier PATIENT

maj_patient(patient)
OBJ : recopie dans le fichier PATIENT les nouvelles données de patient

list_patient()
OBJ : pour chaque patient, le nom, prénom et date de naissance de ce dernier est affiché

ajout_correspond()
OBJ : ajoute une occurence de CORRESPOND au patient courant (sans écriture dans le fichier PATIENT) tout en maintenant la coherence des informations, c'est à dire le nombre maximum d'occurence de CORRESPOND = CONNECTIVITE

4. ALGORITHMES ABSTRAITS DES MODULES DU NIVEAU 6

Conception abstraite des algorithmes des modules du niveau 6

****

4.Conception abstraite des algorithmes des modules du niveau 6

4.1 Conception abstraite des algorithmes de M1

```
lire CHOIX

tant que (CHOIX <> 9)
  Si CHOIX = 1 gestion patient
  Si  CHOIX =  2 saisie  et  traitement des  données  du
               patient courant
  Si  CHOIX =  3 transfert  sur un traceur des données du
               patient  courant
  Si  CHOIX =  4 visualisation  des  données du patient
               courant
fintant
```

4.1.1 Conception abstraite des algorithmes de **gestion patient**

```
lire COORDONNES-PATIENT
Si (patient n'existe pas) creer_patient
patient devient patient courant
```

4.1.2 Conception abstraite des algorithmes de **saisie et traitement** de données

```
Donner un nom unique au fichier de données(NOMFIC)
Créer  une nouvelle  entrée CORRESPOND  pour le patient
  courant
Lire DATE, TYPE

Tant que (pas fin saisie)
  saisir-données
```

```
  créer fichier de données
fintant

recherche courbes CC et CD de NOMFIC
éliminer les mauvaises courbes de NOMFIC
```

4.1.3 Conception abstraite des algorithmes de **transfert vers un traceur**

```
lire CHOIX
SI CHOIX = 1
(  Utiliser fichier correspondant à la dernière date de
  prise de mesure)
  Tant que (pas fin fichier)
    lire données
    transférer vers le traceur
  fintant
Si CHOIX = 2
  ∀ les entrées de CORRESPOND du patient courant
    Tant que (pas fin fichier)
      lire données
      transférer vers le traceur
    fintant
```

4.1.4 Conception abstraite des algorithmes de **visualisation courbes du patient courant**

```
lire CHOIX
SI CHOIX = 1
  (  Utiliser fichier  correspondant à la dernière date
    de prise de mesure)
  Tant que (pas fin fichier)
  lire données
  visualiser les données à l'écran
fintant
Si CHOIX = 2
  ∀ les entrées de CORRESPOND du patient courant
```

```
Tant que (pas fin fichier)
  lire données
  visualiser les données à l'écran
fintant
```

5. Conclusion

Le logiciel dévéloppé traite surtout les parties saisie et traitement des données,tout en permettant la visualisation des résultats, néanmoins certains problèmes importants, méritant toute l'attention des personnes travaillant sur ce sujet, de la rhinomanométrie n'ont pas été abordés par faute de temps.

Deux de ces problèmes attirent surtout l'attention:

- Comme on l'a fait remarquer en 4.2.2, l'échantillonage effectué est fait de façon périodique. Il serait intéressant de pouvoir mieux analyser le segment linéaire de la courbe en utilisant un échantillonage spatial, c'est à dire, en prenant des échantillons de telle sorte que la distance entre chacun des échantillons soit constante.
- Jusque maintenant, on stocke toutes les données obtenues lors de la prise de mesure. Ceci prend beaucoup d'espace disque. Certains chercheurs, notamment Broms [BROM80], ont essayé de modéliser la courbe rhinomanométrique par des modèles mathématiques mais le logiciel développé ne prend pas en charge cette partie modélisation.

Les deux problèmes mentionnés font l'objet de recherche, et peuvent être intégrés dans le logiciel dévéloppé.

6. BIBLIOGRAPHIE

[BERT85] : B. BERTRAND, D. DUVIVIER, Rhinomanométrie, Clinique Universitaire U.C.L de Mont-Godinne, Belgique, 1985

[BROM82] : P. BROMS, B. JOHNSON and C.J. LAMM, RHINOMANOMETRY, a system for numerical description of nasal airway resistance, Sweden,1982

[CLEM80] : P.A.R. CLEMENT and J.J. DALE, Some statistical data about anterior rhinomanometry, Brussels, Belgium, 1980

[CLEM84] : P.A.R. CLEMENT, Committee report on standardisation of rhinomanometry, Brussels, Belgium, 1984

[COLE80] : Philip COLE, Respiratory rhinomanometry, a review of recent trends, Toronto, Canada, 1980

[EICH85] : J. EICHER and H. LENZ, Comparison of different coefficients and units in rhinomanometry, West-Germany,1985

[KERN81] : E.B. KERN, Committee report on standardisation of rhinomanometry, Rochester, U.S.A., 1981

[TSIR] : KSHIRSAGAR, Multivariate Analysis, Statistics: textbooks and monographe, vol.2

ANNEXES

ANNEXE A :

Si

$x = (x_1 \ldots x_p)'$ le vecteur moyen des n - 1 courbes

y = la $n^{ième}$ courbe $(y_1 \ldots y_p)'$

X la matrice $p * (n - 1)$ des n - 1 courbes formées chacune des p observations. Par construction, les colonnes de cette matrice sont statistiquement indépendantes.

$S_x = X(I - E/n_1)X' == XX' - XEX'/n_1$ la matrice des sommes des carrés et sommes de produits liée à X ($\Sigma = S_x/n_1$)

Alors

$x - y$ est un vecteur multinormal de moyenne $\mu - \tau$ et de matrice de variance-covariance $(n_1 + 1)\Sigma/n_1$ où τ est la moyenne stochastique du vecteur y.

Demonstration :

Toute combinaison linéaire de vecteurs multinormaux est multinormal[1]

De plus

$$E(x^m - y) = E(x^m) - E(y) = \mu - \tau$$

$$var(x^m_k - y_k) = \sigma^2_k / n1 + \sigma^2_k = (n1 + 1)\sigma^2_k/n1$$

$$cov(x^m_k - y_k, x^m_j - y_j) = cov(x^m_k, x^m_j) - cov(y_k, x^m_j) - cov(y_j, x^m_k) + cov(y_k, y_j)$$

Or

$cov(y_k, x^m_j) = cov(y_j, x^m_k) = 0$ car les courbes sont indépendantes les unes des autres.

$$cov(y_k, x^m_j) = \sigma^k_j$$

$$cov(x^m_k, x^m_j) = \sigma^k_j/n1$$

dès lor

[1] Ksirsagar, chap 2, Théorème 2

$$\mathrm{cov}(x^{m}_{ik} - y_{ik}, x^{m}_{jl} - y_{jl}) = \sigma^{kl}_{jl}/n1 + \sigma^{kl}_{jl} = (n1 + 1)\sigma^{kl}_{jl}/n1$$

LISTING DES SOURCES DES PROGRAMMES

MODULES FONTIONNELS

```
#define RHINO "/dev/rhino"
#define FNX(x) (((x)>>px) + 320)
#define FNY(y) (((y)>>py) + 240)

#define COMMENT(Z)  { G_CURSIPOS(1,30);printf("%-25.25s",Z);G_CURSIPOS(FNY(y),FNX(x));}
#include <stdio.h>
#include "9754.h"
#include "err_msg.h"
#include "struct.h"
#include <signal.h>
#define TEMP1 "/usr/tmp/RHINO_1"
#define TEMP2 "/usr/tmp/RHINO_2"
#define TEMPO "/usr/tmp/RHINO"
#define LOOP(x) {int i = x ; while (i > 0)  --i ;}
#define MAX(x) (30 * 1000 / x )
char  f1[60], f2[60];
/* ------------------------------------------------------------------------ */
/* state = 1 : calibrate                                                    */
/* state = 2 : draw in yellow 1 part                                        */
/* state = 3 : draw real and store part 1                                   */
/* state = 4 : stop before inversing                                        */
/* state = 5 : draw in yellow 2 part                                        */
/* state = 6 : draw real and store part 2                                   */
/* state = 7 : stop for observing                                           */
/* state = 8 : model                                                        */
/* ------------------------------------------------------------------------ */

FILE *tfopen();
char *rhino;
if_del(sig)
int sig;
{
        close_rhino();
        cbk_nch(-1);
        exit(sig - 1);
}

/*
    rep    : nbre de ms entre chaque scanning
    alpha  : % de l'interval de confiance
*/

trt_pat(rep,alpha,patient)
int alpha, rep;
s_identification *patient;
{
        FILE *fp, *rdi;
        int nbms;
        int count;
        int loop ;
        short x,y;
        int c,co;
        int state ;
        int px,py;
        px = 2;
        py = 3 ;            /* pragmatic !!!! */
        loop = 1000 ; /* atoi(getenv("LOOP"));  */
        if (loop == 0 )
                {
                 printf("ENVIRONNEMENT : LOOP \n");
```

```
            exit(1);
            }
        rhino = RHINO;
        signal(SIGINT,if_del);
        open_rhino();
        nbms = rep;

        state = 1 ;
        cbk_nch(0) ;

        while (state >= 1 )
        {

                G_CURS1POS(FNY(0),FNX(0));
                G_DRAWVSP(0); /* restart position */
                G_DRAWVSP(1); /* end  = start position */
                /* read data and position the cursor */
                G_COLORS(G_C_GREEN,G_C_GREEN);
                message("Calibrer       o",G_C_GREEN);
                while ((c = getchar()) != 'o' && c != 'c') printf("\007");
                if (c == 'o' )
                        {
                         state = 0 ;
                         continue;
                        }
                x = y = 0 ;
                start_rhino(nbms);
                message("Voir         o",G_C_GREEN);
                while (state == 1 )     /* for calibrate */
                {
                        read_rhino(&y,&x);
                        G_CURS1POS(FNY(y),FNX(x));
                        if (rdchk(0))
                        {
                                if (( c = getchar()) == 'o' ) state = 1;
                                if ( c == 'v' ) state = 2 ;
                        }
                }
                if (state != 2 ) continue;

                /* *************    DEBUT 1  ****************** */
                /* read data and draw it in yellow */
                G_COLORS(G_C_WHITE,G_C_WHITE);
                message("Go           o",G_C_WHITE);
                while (state == 2 )
                {
                        read_rhino(&y,&x);
                        G_DRAWVECT(3,FNY(y),FNX(x));
                        if (rdchk(0))
                        {
                                if (( c = getchar()) == 'o' ) state = 1;
                                if ( c == 'n' ) state = 3 ;
                        }
                }
                if (state != 3) continue;
                /* read data and draw it in red and store it */
                G_COLORS(G_C_GREEN,G_C_GREEN);
                count = 0;

                fp = tfopen(TEMP1,"w");
```

```
        message("Stop 1 fois  o",G_C_GREEN);
        while (state == 3 && count < MAX(nbms) )
        {
                read_rhino(&y,&x);
                G_DRAWVECT(3,FNY(y),FNX(x));
                fwrite(&y,sizeof(short),1,fp);
                fwrite(&x,sizeof(short),1,fp);
                if (rdchk(0))
                {
                        if (( c = getchar()) == 'o' ) state = 1;
                        if ( c == 's' ) state = 4 ;/* for inversion */
                }
                count++;
        }
        if (count == MAX(nbms)) state = 4;
        stop_rhino();
        fclose(fp);
        if (state != 4) continue;
        /* *************      FIN1   ***************** */
        message("Inverse      o",G_C_GREEN);
        while (( c = getchar()) != 'o' && c != 'i') printf("\07"); /* BELL */
        if (c != 'i')
        {
                state = 1;
                continue;
        }
        state = 5 ;
        /* *************    DEBUT 2  ***************** */
        /* read data and draw it in yellow */
        start_rhino(nbms);
        G_COLORS(G_C_WHITE,G_C_WHITE);
        message("Go           o",G_C_WHITE);
        while (state == 5 )
        {
                read_rhino(&y,&x);
                G_DRAWVECT(3,FNY(-y),FNX(x));
                if (rdchk(0))
                {
                        if (( c = getchar()) == 'o' ) state = 1;
                        if ( c == 'g' ) state = 6 ;
                }
        }
        if (state != 6) continue;
        /* read data and draw it in red and store it */
        G_COLORS(G_C_BLUE,G_C_BLUE);
        count = 0;
        fp = tfopen(TEMP2,"w");
        message("Stop 2 fois  o",G_C_BLUE);
        while (state == 6 && count < MAX(nbms) )
        {
                read_rhino(&y,&x);
                G_DRAWVECT(3,FNY(-y),FNX(x));
                fwrite(&y,sizeof(short),1,fp);
                fwrite(&x,sizeof(short),1,fp);
                if (rdchk(0))
                {
                        if (( c = getchar()) == 'o' ) state = 1;
                        if ( c == 's' ) state = 7 ;
                }
                count++;
```

```
        }
        if (count == MAX(nbms)) state = 7;
        stop_rhino();
        fclose(fp);
        if (state != 7) continue;
        /* **************      FIN2   ****************** */
        /* Stop for observing and storing part of data */
        message("Partie (y/n)?    o",G_C_BLUE);
        while (( c = getchar()) != 'o' && c != 'n' && c != 'y') printf("\07"); /* BELL */
        if (c == 'y')
        {
                FILE *fo;
                int st = 0;
                clr_all2();
                G_CURSIPOS(FNY(0),FNX(0));
                G_DRAWVSP(0); /* restart position */
                G_DRAWVSP(1); /* end  = start position */
                message("Go draw curves ",G_C_BLUE);
                while (( c = getchar()) != 'g') printf("\07"); /* BELL */
                G_COLORS(G_C_YELLOW,G_C_YELLOW);

                fp = tfopen(TEMP1,"r");
                fo = tfopen(TEMPO,"w");

                while (fread(&y,sizeof(y),1,fp) == 1)
                {
                        fread(&x,sizeof(x),1,fp);
                        G_DRAWVECT(3,FNY(y),FNX(x));
                        if (st)
                        {
                                fwrite(&y,sizeof(y),1,fo);
                                fwrite(&x,sizeof(x),1,fo);
                        }
                        if (rdchk(0))
                        {
                                if ((c = getchar()) == 'g')
                                {
                                        st = 1;
                                        message("Stop",G_C_BLUE);
                                        G_COLORS(G_C_RED,G_C_RED);
                                }
                                else
                                {
                                        if (c == 's') break;
                                }
                        }
                        LOOP(loop);
                }
                fclose(fo);
                fclose(fp);
                mv(TEMPO,TEMP1);
        }
        jojo();



        /* ********* MODELISATION ********** */
        message("Model        o",G_C_BLUE);
        while (( c = getchar()) != 'o' && c != 'm') printf("\07"); /* BELL */
        if (c != 'm')
```

```
                {
                        state = 1;
                        continue;
                }
                /* model of curves */

                ecran(IQ,rep);
                G_CURSIPOS(20,20);
                model(f1,f2,px,py);
                state = 0 ;
        }
        close_rhino();
        cbk_nch(-1);
}

jojo()
{
   FILE *fdat;
   char tab[5];
   int  n;

   fdat  = fopen("dat.dat","r");
   fscanf(fdat,"%d",&n);

   sprintf(f1,"/usr/gast/rhino/jojo/jojo%d",n);
   strcat(f1,"1");
   fclose(fdat);
   fdat = fopen(f1,"w");
   mv(TEMP1,f1);
   fclose(fdat);

   sprintf(f2,"/usr/gast/rhino/jojo/jojo%d",n);
   strcat(f2,"2");
   fdat = fopen(f2,"w");
   mv(TEMP2,f2);
   fclose(fdat);

   fdat  = fopen("dat.dat","w");
   fprintf(fdat,"%d",++n);
   fclose(fdat);
}
```

```
#include "9754.h"
#include <signal.h>
#include <stdio.h>
#include "files.h"
#include "err_msg.h"
#define FNX(x) (((x)>>px) + 320)
#define FNY(y) (((y)>>py) + 240)
#define COMMENT(Z) { G_CURSIPOS(20,20);printf("%s",Z);G_CURSIPOS(FNY(srhino.y), FNX(srhino.x));}


static FILE         *f_index, *f_input, *f_output;
static s_table      *stable, taby;
static s_f_pointer  tab_index[MAX_DIM];
static s_covar      varcov, r;
static s_lines      ptr[2];
static s_line       last_line, pas;
static s_hierarchie hierarchie;
static s_distance   *dist, distance;
static s_tablex     tx;

int         numcmp();

choix(fi, fo, n)
char  *fi, *fo;
int   n;
{
   int          status, nbre_x, nbre_enreg, ptr_index[2 * (NBRE_COURBES + 1)];
   int          *last, *first;
   s_filename   filename;
   short        max_xneg, min_xpos;

   if ((f_index = fopen(fi,"r")) != NULL)
   {
      status = read_filename(&filename, f_index);
      f_input = fopen(filename.y,"r");
      if (status != ERROR)
      {
         if ((f_output = fopen(fo,"w")) != NULL)
         {
            stable = &taby;
            dist   = &distance;
            nbre_enreg  = fill_index(f_index, tab_index);
            fill_limits(tab_index, nbre_enreg, &min_xpos, &max_xneg, ptr, ptr_index);
            det_x(max_xneg, min_xpos, ptr, &pas);
            fill_x(&tx, ptr, max_xneg, min_xpos, &pas);
            scr_tx(&tx, ptr[0].nbrex, ptr[1].nbrex);
            write_filename(&filename, f_output);
            complete_y(stable, tab_index, nbre_enreg, ptr, f_input);
            scr_taby(&taby,ptr);
            fill_cvar(&taby, &varcov, &r, ptr);
            scr_varcov(&varcov, ptr[0].nbrex, ptr[1].nbrex);
            scr_r(&r, ptr[0].nbrex, ptr[1].nbrex);
            calcul_distance(&taby, &distance, ptr, &varcov);
            scr_distance(&distance, ptr[0].nbre, ptr[1].nbre);
            agglomeration_hierarchique(&distance, &hierarchie, ptr, &taby);
            scr_hierarchie(&hierarchie, ptr[0].nbre, ptr[1].nbre);
            fclose(f_index);
            fclose(f_input);
            fclose(f_output);
         }
```

```
         else
         {
            printf("%s %s\n",ERR_MSG1,fo);
         }
      }
      else
      {
         printf("%s %s\n",ERR_MSG1,fi);
      }
   }
}

fill_index(f,tab_index)
FILE        *f;
s_f_pointer tab_index[];
{
   int i;

   i = 0;

   while ((read_index(&tab_index[i],f) != ERROR)  && (i <  (2 * NBRE_COURBES + 1)))
   {
    scr_index(&tab_index[i]);
    i++;
   }

   return(--i);
}

fill_limits(tab_index,nbre_enreg,min_xpos,max_xneg,ptr,ptr_index)
s_f_pointer tab_index[];
s_lines     ptr[];
int         nbre_enreg,ptr_index[];
short       *min_xpos, *max_xneg;
{
   int indice_c, indice_d, j;

   *min_xpos = tab_index[2].max_x;
   *max_xneg = tab_index[2].min_x;

   for (j = 2; j < nbre_enreg; j++)
   {
      if (*min_xpos > tab_index[j].max_x) *min_xpos = tab_index[j].max_x;
      if (*max_xneg < tab_index[j].min_x) *max_xneg = tab_index[j].min_x;
   }

   printf("min_xpos = %5d  max_xneg = %5d\n",*min_xpos,*max_xneg);

   indice_c = 0;
   indice_d = NBRE_COURBES;
   ptr[0].nbre = ptr[1].nbre = 0;

   for (j = 2; j < nbre_enreg; j++)
   {
      if (tab_index[j].type == 'C')
      {
         ptr_index[indice_c++] = j;
         ptr[0].nbre++;
      }
      else
```

```
      {
         ptr_index[indice_d++] = i;
         ptr[1].nbre++;
      }
   }

   ptr[0].indice = --indice_c;
   ptr[1].indice = --indice_d;
}

/*
end_c()
{

   fclose(f_index);
   fclose(f_input);
   fclose(f_output);
}
*/

det_x(max_xneg,min_xpos,ptr,pas)
short   max_xneg, min_xpos;
s_lines ptr[];
s_line  *pas;
{
   int  ptr0_nbre, ptr1_nbre;

   if ((ptr[0].nbre > 2) && (ptr[0].nbre < 5)) ptr[0].nbrex = ptr[0].nbre - 1;
   if ((ptr[0].nbre >= 5) && (ptr[0].nbre <= 7)) ptr[0].nbrex = 4;
   if (ptr[0].nbre > 7) ptr[0].nbrex = 6;

   if ((ptr[1].nbre > 2) && (ptr[1].nbre < 5)) ptr[1].nbrex = ptr[1].nbre - 1;
   if ((ptr[1].nbre >= 5) && (ptr[1].nbre <= 7)) ptr[1].nbrex = 4;
   if (ptr[1].nbre > 7) ptr[1].nbrex = 6;

   pas->c = ((min_xpos - max_xneg) / (ptr[0].nbrex - 1)) + 1;
   pas->d = ((min_xpos - max_xneg) / (ptr[1].nbrex - 1)) + 1;

   printf("pas->c = %5d  pas->d = %5d\n",pas->c,pas->d);
   printf("ptr[0].nbrex = %d  ptr[1].nbrex = %d\n",ptr[0].nbrex, ptr[1].nbrex);

}

fill_x(tx,ptr,max_xneg,min_xpos,pas)
s_tablex *tx;
s_line   *pas;
s_lines  ptr[];
short    max_xneg, min_xpos;
{
   int i, p_0_nx, p_1_nx;

   p_0_nx = ptr[0].nbrex - 1;
   p_1_nx = ptr[1].nbrex - 1;
   tx->tabx[0][0].x = tx->tabx[1][0].x = max_xneg;
   tx->tabx[0][p_0_nx].x = tx->tabx[1][p_1_nx].x = min_xpos;

   for (i = 1; i< p_0_nx; i++)
   {
      tx->tabx[0][i].x = tx->tabx[0][i-1].x + pas->c;
   }
```

```
   for (i = 1; i < p_l_nx; i++)
   {
      tx->tabx[1][i].x = tx->tabx[1][i-1].x + pas->d;
   }
}

complete_y(stable,tab_index,nbre_enreg,ptr,f_input)
s_table     *stable;
s_f_pointer tab_index[];
s_lines     ptr[];
int         nbre_enreg;
FILE        *f_input;
{
   int    ic, id, i, j;

   ic = 0;
   id = NBRE_COURBES;

   for (i = 2; i < nbre_enreg; i++)
   {
      if (tab_index[i].type == 'C')
      {

         fill_y(stable,ic,0,&tab_index[i],f_input,ptr[0].nbre - 1);

         /*
         scr_taby(stable,ptr);
         */

         ic++;
      }
      else
      {

         fill_y(stable,id,1,&tab_index[i],f_input,ptr[1].nbre - 1);

         /*
         scr_taby(stable,ptr);
         */


         id++;
      }
   }

}

interpol(tab,n)
s_rhino tab[];
int     n;
{
   double x, y, dx, dy;

   switch (n)
   {
      case 2 :
             x = (double)(tab[2].x - tab[1].x);
             y = (double)(tab[2].y - tab[1].y);
             dx= (double)(tab[0].x - tab[1].x);
```

```
                tab[0].y = (int)(((y * dx) / x)) + tab[i].y;
                return(tab[0].y);
        default :
                return(ERROR);
    }

}

fill_y(y,i,indice_1,pointer,f_input,n)
s_table     *y;
s_f_pointer *pointer;
int         i, indice_1, n;
FILE        *f_input;
{
    s_rhino srhino,sort_table[POINT_CURVES],table_interpol[INTERPOL];
    int     nombre, num_cmp(), swap_srhino();
    int     is, ix, ii, temp, i_interpol, nbre_erreurs;
    BOOLEAN b_premier, b_search, b_interpol;

    ii = 0;
    nbre_erreurs = 5;
/* scr_index(pointer);  */
    while ((fseek(f_input,pointer->offset,0) == ERROR) && (nbre_erreurs-- >= 0)) ;
    nombre = pointer->nombre;
    /* chargement de la table de tri */

    /*
    printf("offset = %5d nombre = %5d file_pointer = %ld\n",pointer->offset, pointer->nombre, f_input);
    scr_test_file(f_input);
    */

    while ((read_data(&srhino,f_input) != ERROR) && (nombre-- >= 0))
    {
        /*
        scr_data(&srhino);
        */

        sort_table[ii].x = srhino.x;
        sort_table[ii].y = srhino.y;
        ii++;
    }

    /*
    printf("  ENTREE : sort()\n");
    */

    sort(sort_table,ii,numcmp,swap_srhino);

    /*
    scr_sort_table(sort_table,ii);
    */

    /*
    printf("  SORTIE : sort()\n\n");
    */

    is = 0;
    ix = 0;
    i_interpol = 1;
    b_search = TRUE;
```

```
   b_premier = TRUE;
   b_interpol = FALSE;

   while ((is < ii) && !b_interpol)
   {
      temp = numcmp(&tx.tabx[indice_l][ix].x,&sort_table[is].x);
      /*
      printf("1");
      */

      switch (temp)
      {
         case INFERIEUR :
                         /*
                         printf("2");
                         */
                         if (i_interpol == 2)
                         {
                            table_interpol[i_interpol].x = sort_table[is].x;
                            table_interpol[i_interpol].y = sort_table[is].y;
                            y->line[i].column[ix] = interpol(table_interpol,2);
                            b_interpol = (ix == n) ? TRUE : FALSE;
                            tx.tabx[indice_l][ix].nbre++;
                            tx.tabx[indice_l][ix].sx = tx.tabx[indice_l][ix].sx + y->line[i].column[ix];
                            ix++;
                            i_interpol = 1;
                         }

                         if (b_search)
                            ix++;
                         else
                         {
                            i_interpol = 2;
                            table_interpol[0].x = tx.tabx[indice_l][ix].x;
                         }

                         break;
         case EGAL      :
                         /*
                         printf("3");
                         */
                         if (b_premier)
                         {
                            y->line[i].first_column = ix;
                            b_premier = FALSE;
                         }
                         b_interpol = (ix == n) ? TRUE : FALSE;
                         y->line[i].column[ix] = sort_table[is].y;
                         tx.tabx[indice_l][ix].nbre++;
                         tx.tabx[indice_l][ix].sx =tx.tabx[indice_l][ix].sx + y->line[i].column[ix];
                         ix++;
                         b_search = FALSE;
                         table_interpol[i_interpol].x = sort_table[is].x;
                         table_interpol[i_interpol].y = sort_table[is].y;
                         table_interpol[0].x          = tx.tabx[indice_l][ix].x;
                         i_interpol++;
                         break;
         case SUPERIEUR :
                         /*
                         printf("4");
```

```
                        */
                        if (b_premier)
                        {
                           y->line[i].first_column = ix;
                           b_premier = FALSE;
                        }

                        b_search = b_interpol = FALSE;
                        i_interpol = 1;
                        table_interpol[i_interpol].x = sort_table[is].x;
                        table_interpol[i_interpol].y = sort_table[is].y;
                        table_interpol[0].x          = tx.tabx[indice_l][ix].x;
                        is++;
                        i_interpol++;
                        break;
      }
   }
   y->line[i].last_column = ix;
   /*
   printf("5\n");
   */
}

calcul_distance(stable,distance,ptr,varcov)
s_table    *stable;
s_distance *distance;
s_lines    ptr[];
s_covar    *varcov;
{
   int     i,j;
   BOOLEAN err_inv;
   s_covar varcov_inv;

   /* calcul de la matrice inverse */

   if (inverse(varcov,&varcov_inv,ptr) == ERROR) err_inv = TRUE ; else err_inv = FALSE;

   /* mise a zero de la diagonale */

   for (i = 0; i <= NBRE_COURBES; i++) distance->d2[i][i] = 0;

   /*
   scr_distance(distance, ptr[0].nbre, ptr[1].nbre);
   */

   /* calcul de la distance D */

   for (i = NBRE_COURBES; i <= ptr[1].indice; i++) fill_distance(stable,distance,i,'D',ptr,&varcov_inv,err_inv,varcov);

   /*
   scr_distance(distance, ptr[0].nbre, ptr[1].nbre);
   */

   /* calcul de la distance C */

   for (i = 0; i <= ptr[0].indice; i++) fill_distance(stable,distance,i,'C',ptr,&varcov_inv,err_inv,varcov);

   /*
   scr_distance(distance, ptr[0].nbre, ptr[1].nbre);
   */
```

```
)

fill_distance(stable, distance, i, type, ptr, varcov_inv, err_inv, varcov)
s_table     *stable;
s_distance  *distance;
s_covar     *varcov_inv, *varcov;
int         i;
BOOLEAN     err_inv;
char        type;
s_lines     ptr[];
{
   int    ligne, colonne, j, k, l, last, n;
   double var, dist, vect_colonne[NBRE_COURBES + 1], vect_result[NBRE_COURBES + 1];

   n    = (type == 'C') ? ptr[0].nbre   : ptr[i].nbre;
   last = (type == 'C') ? ptr[0].indice : ptr[i].indice;

   for (j = i; j <= last; j++)
   {
      ligne = (type == 'C') ? j : j - NBRE_COURBES;

      for (k = j + 1; k <= last; k++)
      {
         colonne = (type == 'C') ? k : k - NBRE_COURBES;

         for (l = 0; l < n; l++)
         {
            dist           = stable ->line[j].column[l] - stable->line[k].column[l];
            vect_result[l] = vect_colonne[l] = dist;
         }

         if (!err_inv) mul_mat_vect(type,varcov_inv,vect_colonne,ptr,vect_result);

         for (l = 0; l < n; l++)
         {
            /*
            printf("y[%3d][%3d] = %5d ",j,l,stable ->line[j].column[l]);
            printf("y[%3d][%3d] = %5d ",k,l,stable ->line[k].column[l]);
            printf("dist = %f\n",dist);
            */

            var = (type == 'C') ? varcov->vc[l].c[l] : varcov->vc[l].d[l];

            if (err_inv)
               distance->d2[ligne][colonne] += (vect_colonne[l] * vect_result[l]) / var;
            else
               distance->d2[ligne][colonne] += (vect_colonne[l] * vect_result[l]);

            distance->d2[colonne][ligne] = (type =='D') ? distance->d2[ligne][colonne] : distance->d2[colonne][ligne];
         }

         /*
         if (type == 'C')
            printf("distance[%3d][%3d] = %f\n",ligne,colonne,distance->d2[ligne][colonne]);
         else
            printf("distance[%3d][%3d] = %f\n",colonne,ligne,distance->d2[colonne][ligne]);
         */
      }
   }
}
```

```
#include <stdio.h>
#include "9754.h"
double a[10][10];
double result[10];

model(f1,f2,px,py)
char  *f1, *f2;
int   px, py;
{
   char c, nomf1[60], nomf2[60];
   int  t;

   strcpy(nomf1,f1);
   strcat(nomf1,".idx");
   strcpy(nomf2,f2);
   strcat(nomf2,".idx");

   indexe(f1,nomf1);
   indexe(f2,nomf2);

   strcpy(nomf1,f1);
   strcat(nomf1,".idx");
   strcpy(nomf2,f1);
   strcat(nomf2,".chx");
   choix(nomf1,nomf2,2);

   strcpy(nomf1,f2);
   strcat(nomf1,".idx");
   strcpy(nomf2,f2);
   strcat(nomf2,".chx");
   choix(nomf1,nomf2,2);
/*
   ecran(10,"jojo");
   visual('i',2,3,6,nomf1);
   visual('i',2,3,5,nomf2);
*/

   while ((c = getchar()) != 's') ;
   G_CLRWIN;
   G_COLORS(G_C_WHITE,G_C_WHITE);
   G_GROFF;
   cls();
   cursor(10,30);
}
```

```
#include "9754.h"
#include <stdio.h>
#include "files.h"
#include "err_msg2.h"
#define FNX(x) (((x)>>px) + 320)
#define FNY(y) (((y)>>py) + 240)
#define COMMENT(Z) { G_CURSIPOS(20,20);printf("%s",Z);G_CURSIPOS(FNY(srhino.y), FNX(srhino.x));}
#define GAUCHE  1
#define DROITE  -1


FILE *f_index, *f_input;

init_v(fi)
char *fi;
{
 s_filename      DFILE;
 if ((f_index = fopen(fi, "r")) != 0)
    {
     read_filename(&DFILE,f_index);
     if ((f_input = fopen(DFILE.y, "r")) == 0)
        {
         printf("%s %s \n",ERR_MSG1,DFILE.y);
         return(ERROR);
        }
    }
 else
    {
     printf("%s %s \n",ERR_MSG1,fi);
     return(ERROR);
    }
}

end_v()
{
 fclose(f_input);
 fclose(f_index);
}

visual(type,px,py,color,f)
int px, py, color;
char *f, type;
{
 s_f_pointer   IFILE;
 s_rhino       srhino;
 int i = 3,x;
 int state = OK;
 int y = color;
 int multfact;
 char c;
 multfact = (index(f,'1') != NULL) ? GAUCHE : DROITE;
 if ((x = init_v(f)) != ERROR) ;
    {
     G_COLORS(color,color);
     while (read_index(&IFILE,f_index) != ERROR)
           {
            fseek(f_input, IFILE.offset, 0);
            switch (type)
                  {
                   case 'O' :
```

```
                        while ((read_data(&srhino,f_input) != ERROR) && (--IFILE.nombre >= 1))
                              {
                               G_DRAW(2, FNY(multfact * srhino.y), FNX(srhino.x));
                              }
                        break;
                   case '1' :
                        read_data(&srhino,f_input);
                        G_CURSIPOS(FNY(multfact * srhino.y), FNX(srhino.x));
                        while ((read_data(&srhino,f_input) != ERROR ) && (--IFILE.nombre >= 1))
                              {
                               G_DRAWVECT(3, FNY(multfact * srhino.y), FNX(srhino.x));
                              }
                       break;
                 default : printf("%s \n",ERR_MSG5);
                }
          }
    }
 end_v();
 G_COLORS(G_C_GREEN,G_C_GREEN);
}
```

```
#include "ploter.h"
#include "err_msg2.h"
#include "files.h"
#include <stdio.h>
#define TIRET 5
#define MAX_X 640
#define MAX_Y 480
#define MID_X ( MAX_X / 2)
#define MID_Y ( MAX_Y / 2)
#define DETAIL  1
#define P_FNX(x) (((x) >> px) + 320)
#define P_FNY(y) (((y) >> px) + 240)



static FILE *fd_plot;

c_plot(facdiv,filname)
int facdiv;
char *filname;
{
 int i;

 P_OPEN_PLOTTER;
 P_RESET(2); /* reset ploter */
 P_SCALE(4,4);
 P_PEN_SEL(1);
/* ------------------------------------------------------------------------ */
/* grid                                                                     */
/* ------------------------------------------------------------------------ */
 /* +++ */ P_PEN_SEL(1) ;/* set principal color to green */
 P_LINE(MID_Y,0,MID_Y,MAX_X - TIRET); /* horizontal axes    **/
/* ------------------------------------------------------------------------ */
/* right arrow                                                              */
/* ------------------------------------------------------------------------ */
 P_LINE(MID_Y - 2 * TIRET,MAX_X -TIRET - 2 * TIRET,MID_Y,MAX_X - TIRET);      /* up part */
 P_LINE(MID_Y + 2 * TIRET,MAX_X - TIRET - 2 * TIRET,MID_Y,MAX_X -TIRET);      /* down part */
#if DETAIL
 for (i = 0; i< MAX_Y;i +=20)   /* horizontal tiret */
        {
         P_LINE(i,MID_X - TIRET,i,MID_X + TIRET);       /* horizontal */
        }
#endif
/* ------------------------------------------------------------------------ */
/* vertical     axes                                                        */
/* ------------------------------------------------------------------------ */
 P_LINE( 0,MID_X,MAX_Y - TIRET,MID_X);
/* ------------------------------------------------------------------------ */
/* UP  ARROW                                                                */
/* ------------------------------------------------------------------------ */
 P_LINE(MAX_Y - TIRET - 2 * TIRET,MID_X - 2 *  TIRET,MAX_Y - TIRET ,MID_X );
/* ------------------------------------------------------------------------ */
/* left part                                                                */
/* ------------------------------------------------------------------------ */
 P_LINE(MAX_Y -TIRET - 2 * TIRET ,MID_X + 2 * TIRET,MAX_Y - TIRET ,MID_X );      /* right part */
#if    DETAIL
 for ( i = 0 ; i< MAX_X ; i += 20)    /* vertical tiret */
        {
         P_LINE(MID_Y - TIRET,i,MID_Y + TIRET,i);
        }
```

```
#endif
/* ------------------------------------------------------------------------ */
/* circle                                                                   */
/* ------------------------------------------------------------------------ */
#if    DETAIL
 P_PEN_SEL(1);  /* select pen for circle */
 P_COORD(MID_X,MID_Y);
 P_PEN_UP;
 P_PLOT;
 for ( i = 40 ; i<200; i +=40)  /* circle */
       {
        P_CIRCLE(i,0,360);
       }
#endif
 P_PEN_UP;
 P_COORD(2,440);
 P_PLOT;
 P_DIR_SIZE(10,0,10); /* dim of lettre */
 P_FONT(0);
 P_CHAR(" RHINOMANOMETER ");
 P_COORD(MID_X + 30,440);
 P_PLOT;
 print("B%s\n",filname);
 draw_curves("/usr/gast/rhino/jojo/jojo2",0.10,0.10,2);
 printf("\nzorro");

 P_CLOSE_PLOTTER;
}


draw_curves(name,xscale,yscale,line_type)
char      *name;
int       xscale, yscale, line_type;
{
   int   i, mult, px, py;
   char  filnam[61];
   FILE  *fd;
   s_rhino srhino;



   i = 0;
   mult = 1;
   P_RESET(2);
   P_SCALE(0.1,0.02);
   P_PEN_SEL(2);
   P_PEN_UP;
   P_ORIG(MID_X*4,MID_Y*4 );
   P_COORD(MID_X*4,MID_Y*4 );
   P_PLOT; P_PT_MARK(0);
   P_TYPE_LIG(line_type,1);
   while (i++ < 2)
   {
      sprintf(filnam,"%s%d",name,i);
      if ((fd = fopen(filnam,"r")) == 0 )
      {
         printf("%s %s \n",ERR_MSG1,filnam);
         return(ERROR);
      }
      read_data(&srhino,fd);
```

```
      P_COORD((int)(srhino.x+ MID_X*4),(int)(mult * srhino.y * 0.4+ MID_Y*4));
      P_PLOT;
      P_PEN_DOWN;
      while (read_data(&srhino,fd) != ERROR)
      {
         P_COORD((int)(srhino.x+MID_X*4),(int)(mult * srhino.y * 0.4+MID_Y*4));
         P_PLOT;
      }
      P_PEN_UP;
      mult = -1;
      fclose(fd);
   }
}
```

NOYAU FONCTIONNEL

```
#include <time.h>
#include <stdio.h>
#include "9754.h"

long        time(), clock;
char        *asctime(), *data;
struct  tm *gmtime(), *timei;

imp_date()
{
   char  c[12];

   clock = time(0);
   timei = gmtime(&clock);
   sprintf(c,"%2d/%2d/19%2d ",timei->tm_mday, timei->tm_mon + 1, timei->tm_year);

   G_GREAT(0);
   G_DRAWCAR(0,0,0);
   G_CURSIPOS(460,540); printf("%s",c);
}
```

```
#include "files.h"
#include <stdio.h>
#include <math.h>
#include "err_msg.h"
#define  POSITIF  0
#define  NEGATIF  1
#define  DEPART   2

static FILE       *f_ii, *f_oi;
static int        nbre_plateau,mult;
short      push(), pop();
static s_filename m;

static t_min_max(s,f)
s_rhino  *s,*f;
{
 short  x1_x2;
 if (s->x < m.min_x)
    m.min_x = s->x;
 else if (s->x > m.max_x)
    m.max_x = s->x;
 if (s->y < m.min_y)
    m.min_y = s->y;
 else if (s->y > m.max_y)
    m.max_y = s->y;
 if ((x1_x2 = abs(s->x - f->x)) > m.max_x1_x2)
    m.max_x1_x2 = x1_x2;
 else  if (x1_x2 <  m.min_x1_x2)
    m.min_x1_x2 = x1_x2;
}

static init_min_max(f,rhino)                    /* initialise les min/max et x1_x2 */
FILE        *f;
s_rhino     *rhino;
{
 if (read_data(rhino,f) != ERROR)
    {
     m.min_x = rhino->x;
     m.max_x = rhino->x;
     m.min_y = rhino->y;
     m.max_y = rhino->y;
     fseek(f,01,0);
    }
}

static create_index(indexes)
s_f_pointer  indexes[];
{
 s_rhino    srhino,x1_x2;
 s_somme    s1, s2;
 int        i, cpt, type;
 short      y_min, y_max, x_min, x_max;

 indexes[0].offset = 01;
 y_min = y_max = 0;
 m.min_x = m.max_x = m.min_y = m.max_y = 0;
 m.min_x1_x2 = m.max_x1_x2 = 0;
 type = DEPART;
 i = 0;
 indexes[i].min_x = indexes[i].max_x = indexes[i].min_y = indexes[i].max_y = 0;
```

```
cpt = 0;
zero_somme(&s1);
zero_somme(&s2);
init_min_max(f_ii,&x1_x2);
while ((read_data(&srhino,f_ii) != ERROR) && (i < (2 * NBRE_COURBES + 1)))
      {
       t_min_max(&srhino,&x1_x2);
       if (srhino.y >= 0)
          {
           if (type != POSITIF)
              {
               i++;
               indexes[i].min_x = indexes[i].max_x = indexes[i].min_y = indexes[i].max_y = 0;
               y_min=0;
               indexes[i].type = 'D';
               if (type != DEPART)
                  {
                   write_sum(&s1,f_oi);
                   write_sum(&s2,f_oi);
                  }
               type = POSITIF;
               zero_somme(&s1);
               zero_somme(&s2);
              }
           calcul_somme(srhino.x * mult, srhino.y, &s2);
           if (srhino.y > y_max)
              {
               clear(); push(srhino.x); nbre_plateau = 1;
               indexes[i].offset = ftell(f_ii) - SIZE_REC;
               indexes[i-1].max_y = indexes[i].max_y = y_max = srhino.y;
               indexes[i-1].min_x = indexes[i].min_x = x_min = srhino.x;
               add_somme(&s1, &s2);
               zero_somme(&s2);
              }
           else if (srhino.y == y_max)
              {
               if ((indexes[i].offset - ftell(f_ii)) == 2 * SIZE_REC)
                  {
                   nbre_plateau++;
                   push(srhino.x);
                  }
              }
          }
      else
          {
           if (type != NEGATIF)
              {
               i++;
               indexes[i].min_x = indexes[i].max_x = indexes[i].min_y = indexes[i].max_y = 0;
               y_max=0;
               indexes[i].type = 'C';
               if (type != DEPART)
                  {
                   write_sum(&s1,f_oi);
                   write_sum(&s2,f_oi);
                  }
               type = NEGATIF;
               zero_somme(&s1);
               zero_somme(&s2);
              }
```

```
            calcul_somme(-srhino.x * mult, -srhino.y, &s2);
            if (srhino.y < y_min)
               {
                clear(); push(srhino.x); nbre_plateau = 1;
                indexes[i].offset = ftell(f_ii) - SIZE_REC;
                indexes[i-1].min_y = indexes[i].min_y = y_min = srhino.y;
                indexes[i-1].max_x = indexes[i].max_x = x_max = srhino.x;
                add_somme(&s1, &s2);
                zero_somme(&s2);
               }
            else if (srhino.y == y_min)
               {
                if ((indexes[i].offset - ftell(f_ii)) == 2 * SIZE_REC)
                   {
                    nbre_plateau++;
                    push(srhino.x);
                   }
               }
             }
          x1_x2.x = srhino.x;
          x1_x2.y = srhino.y;
        }
 return(i);
}

static complete_name(n)
s_filename   *n;
{
 n->min_x = m.min_x;
 n->max_x = m.max_x;
 n->min_y = m.min_y;
 n->max_y = m.max_y;
 n->max_x1_x2 = m.max_x1_x2;
 n->min_x1_x2 = m.min_x1_x2;
}

static complete_index(indexes,n)
s_f_pointer   indexes[];
int           n;
{
 int i;
 long l;
 indexes[0].type = (indexes[1].type == 'C') ? 'D' : 'C';
 for (i = 0; i <= n ; i++)
     {
      indexes[i].no_courbe = i;
      indexes[i].nombre    = (int)(((indexes[i+1].offset - indexes[i].offset)/SIZE_REC));
     }
 l = fseek(f_ii,0l,2);
 indexes[n].nombre    = (int)((ftell(f_ii) - indexes[n - 1].offset)/SIZE_REC + 1);
}

indexe(filename_i,filename_o)
char  *filename_i, *filename_o;
{
 s_f_pointer   tab_ind[MAX_DIM];
 s_filename    name;
 char          *f_somme;
 int           n,i;
```

```
 if (index(filename_i, '1') != NULL)
    {
     f_somme = "SOMME1.SUM";
     mult    = 1;
    }
 else
    {
     f_somme = "SOMME2.SUM";
     mult    = -1;
    }

 if ((f_ii = fopen(filename_i, "r")) == 0)
    {
     printf("%s %s\n",ERR_MSG1,filename_i);
    }
 else
    {
     if ((f_oi = fopen(f_somme,"w")) == 0)
        {
         printf("%s %s\n",ERR_MSG1,f_somme);
        }
     else
        {
         n = create_index(tab_ind);
         fclose(f_oi);
         if ((f_oi = fopen(filename_o, "w")) == 0)
            {
             printf("%s %s \n",ERR_MSG1, filename_o);
            }
         complete_name(&name);
         for (i = 0; i < 60; i++) name.y[i] = ' ';
         strcpy(name.y, filename_i);
         write_filename(&name,f_oi);
         complete_index(tab_ind,n);
         write_tab_index(tab_ind,n,f_oi);
         fclose(f_ii);
         fclose(f_oi);
        }
    }
}
```

```
#define TEMP1       "/usr/tmp/RHINO_1"
#define TEMP2       "/usr/tmp/RHINO_2"
#define TEMP1DX     "/usr/tmp/RHINO_1.IDX"
#define TEMP2DX     "/usr/tmp/RHINO_2.IDX"
#define CHOIX1DX    "/usr/tmp/CHOIX_1.IDX"
#define CHOIX2DX    "/usr/tmp/CHOIX_2.IDX"
#define SOMM1SUM    "/usr/tmp/SOMME_1.sum"
#define SOMM2SUM    "/usr/tmp/SOMME_2.sum"

#define CHAR          0
#define INT           1
#define ALPHABET      2
#define ALPHANUM      3
#define ERROR        -1
#define ABORT         12
#define OK            13
#define TRUE          1
#define FALSE         0
#define MAX_INTX      100
#define NBRE_COURBES  20
#define POINT_CURVES  100
#define HIGH_VALUE    9999
#define MAX_DIM       50
#define INTERPOL      3
#define SUPERIEUR     1
#define INFERIEUR     -1
#define EGAL          0
#define BOOLEAN       int
#define SIZE_REC      (2 * sizeof(short))
#define EPSILON       0.0001

/*
typedef struct _tablebis
       {
        s_ltable line[6];
       } s_tablebis;
*/

typedef struct _vecty
       {
        double vligne[1][NBRE_COURBES];
        double vcolonne[NBRE_COURBES][1];
       } s_vecty;

typedef struct _vect
       {
        s_vecty vy[NBRE_COURBES];
       } s_vect;

typedef struct _vectbis
       {
        s_vecty vy[6];
       } s_vectbis;

/* definition de la structure de donnees rhino */
typedef struct _rhino
       {
        short y;
        short x;
       } s_rhino;
```

```
/* structure de type S_LINExx */
typedef struct _line
       {
        int c;
        int d;
       } s_line;

typedef struct _lines
       {
        int indice;
        int nbre;
        int nbrex;
       } s_lines;



/* definition de type pour la matrice des x */
typedef struct _x
       {
        short   x;
        int     nbre;
        long    sx;
       } s_x;

typedef struct _tablex
       {
        s_x     tabx[2][NBRE_COURBES + 1];
        int     first_column;
        int     last_column;
       } s_tablex;

/* definition de type pout la matrice des y */
typedef struct _ltable
       {
        int   first_column;
        int   last_column;
        short column[NBRE_COURBES + 1];
       } s_ltable;

typedef struct _table
       {
        s_ltable line[NBRE_COURBES + 1];
       } s_table;

/* definition de type pour la matrice de variance/covariance et de r */
typedef struct  _covarl
       {
        double c[NBRE_COURBES + 1];
        double d[NBRE_COURBES + 1];
       } s_covarl;

typedef struct _covar
       {
        s_covarl vc[NBRE_COURBES];
       } s_covar;

/* definition de type de la matrice des distances */
typedef struct _distance
       {
```

```
        double d2[NBRE_COURBES + 1][NBRE_COURBES + 1];
       } s_distance;

/* definition des structures de fichiers */
typedef struct _filename
       {
        short  max_x;
        short  min_x;
        short  max_y;
        short  min_y;
        short  min_x1_x2;
        short  max_x1_x2;
        char y[12];
       } s_filename;

typedef struct _f_pointer
       {
        long  offset;
        char  type;
        int   nombre;
        int   no_courbe;
        short max_x;
        short min_x;
        short max_y;
        short min_y;
       } s_f_pointer;

typedef struct _somme
       {
        double      x;
        double     x2;
        double    lnx;
        double    lny;
        double   xlny;
        double   lnx2;
        double lnxlny;
        int       cpt;
       } s_somme;

/* definition de type pour la structure hierarchie */
typedef struct _num_distance
       {
        int     numero;
        double  distance;
       } s_num_distance;

typedef struct _hierarchie
       {
        s_num_distance  c[NBRE_COURBES + 1];
        s_num_distance  d[NBRE_COURBES + 1];
       } s_hierarchie;

typedef struct _variable
       {
        int   type;
        union
        {
         int   ival;
         char  cval;
         char  *pval;
```

```
        } uval;
       } s_variable;
```

```
#include <stdio.h>
#include "files.h"


read_sum(s,f_input)
s_somme  *s;
FILE     *f_input;
{
 if (fread((char *)s, sizeof(struct _somme), 1, f_input) == 0)
    return(ERROR);
}

read_data(srhino,f_input)
s_rhino  *srhino;
FILE     *f_input;
{
 if (fread((char *)srhino, sizeof(struct _rhino), 1, f_input) == 0)
    return(ERROR);
}

read_index(IFILE,f_index)
s_f_pointer *IFILE;
FILE        *f_index;
{
 if (fread((char *)IFILE, sizeof(struct _f_pointer), 1, f_index) == 0)
     return(ERROR);
}

read_filename(f,f_input)
s_filename   *f;
FILE         *f_input;
{
 if (fread((char *)f, sizeof(struct _filename), 1, f_input) == 0)
    return(ERROR);
}

write_tab_index(indexes,n,f_output)
s_f_pointer  indexes[];
int          n;
FILE         *f_output;
{
 fwrite(indexes, sizeof(struct _f_pointer), n + 1, f_output);
}

write_index(IFILE,f_index)
s_f_pointer *IFILE;
FILE        *f_index;
{
 fwrite(IFILE, sizeof(struct _f_pointer), 1, f_index);
}

write_sum(s,f_sum)
s_somme     *s;
FILE        *f_sum;
{
 fwrite(s, sizeof(struct _somme), 1, f_sum);
}

write_filename(f,f_output)
s_filename    *f;
```

```
FILE          *f_output;
{
 fwrite(f, sizeof(struct _filename), 1, f_output);
}
```

```
#define ERR_MSG1  "ERROR : impossible d'ouvrir le fichier "
#define ERR_MSG2  "ERROR : la pile est remplie "
#define ERR_MSG3  "ERROR : la pile est vide "
#define ERR_MSG4  "ERROR : dimension de la matrice trop grande "
#define ERR_MSG5  "ERROR : mauvais parametre type de visual() "
#define ERR_MSG6  "ERROR : pivot dans la resolution par Gauss = 0 "
#define ERR_MSG7  "ERROR : Entrer 1, 2, 3, 4 ou 9 selon votre choix "
#define ERR_MSG10 "ERROR : det(A) = 0, matrice est non-inversible "

#define MSG0      "Nombre de ms entre chaque 'scanning'"
#define MSG1      "Continuer     "
#define MSG2      "Voir         o"
#define MSG3      "Commencer    o"
#define MSG4      "Arret 1 fois o"
#define MSG5      "Inverser     o"
#define MSG6      "Arret 2 fois o"
#define MSG7      "Modeliser    o"

#define MSG_ECR0    " Votre choix : "
#define MSG_ECR10   " 1.  RHINOMANOMETRIE"
#define MSG_ECR11   " 1. Saisie des coordonnees d'un patient"
#define MSG_ECR12   " 2. Saisie et traitement des donnees"
#define MSG_ECR13   " 3. Visualisation des courbes d'un patient"
#define MSG_ECR14   " 4. Transfert sur plotter"
#define MSG_ECR110  " 1.1  SAISIE DES COORDONNEES D'UN PATIENT"
#define MSG_ECR111  " IDENTIFICATION PATIENT"
#define MSG_ECR112  " NOM               : "
#define MSG_ECR113  " PRENOM            : "
#define MSG_ECR114  " Date de naissance :   /  /     "
#define MSG_ECR120  " 1.2 SAISIE ET TRAITEMENT DES DONNEES"
#define MSG_ECR121  " SAISIE DES PARAMETRES"
#define MSG_ECR122  " Nombre de ms entre chaque scanning : "
#define MSG_ECR123  " Interval de confiance              : "
#define MSG_ECR130  " 1.3 VISUALISATION DES COURBES D'UN PATIENT"
#define MSG_ECR140  " 1.4 TRANSFERT SUR PLOTTER"
#define MSG_ECR215  " Veuillez, s'il vous plait verifier si vos donnees sont exactes"
#define MSG_ECR9    " 9. Retour au menu precedent"
```

```
#include <stdio.h>
#include "err_msg.h"
#include "9754.h"

trt_error(i,cond)
int  i, cond;
{
   G_CURSIPOS(1,30);
   if (cond == 0)
   {
      G_COLFOREG(0,G_C_RED);
      G_DEFBLIN(G_C_RED,30);
   }
   else
   {
      G_DEFBLIN(8,10);
      G_COLFOREG(0,G_C_WHITE);
   }

   switch(i)
   {
      case  1  : printf("%s",ERR_MSG7);
                 break;
   }
}

message(msg,col)
char  *msg;
int   col;
{
   int i, j;

   G_COLORS(G_C_GREEN,G_C_GREEN);
   clr_err2();
   G_CURSIPOS(1,30);
   printf("%s",msg);
   G_COLORS(col,col);
}
```

```
#include "files.h"

sort(sort_table, n, numcmp, swap)
int         n;
s_rhino     sort_table[];
int         (*numcmp)(), (*swap)();
{
 int       gap, i, j;

 for (gap = n / 2; gap > 0; gap /= 2)
     for (i = gap; i < n; i++)
         for (j = i - gap; j >= 0; j -= gap)
             {
              if ((*numcmp)(&sort_table[j].x, &sort_table[j + gap].x) <= 0)  break;
              (*swap)(&sort_table[j], &sort_table[j + gap]);
             }
}

numcmp(x,y)
short     *x, *y;
{
 if (*x < *y)
     return(INFERIEUR);
 else if (*x > *y)
     return(SUPERIEUR);
 else
     return(EGAL);
}

swap_srhino(x,y)
s_rhino    *x, *y;
{
 s_rhino  temp;
 temp =  *x;
 *x   = *y;
 *y   =  temp;
}
```

GESTION PATIENT

```
#define  CONNECTIVITE 10

typedef struct _identification
{
   char nom[35], prenom[35], date_nais[12];
} s_identification;

typedef struct _correspond
{
   char   date[12], type;
   int    num;
} s_correspond;

typedef struct _patient
{
   int               nombre_mesure;
   s_identification  ident;
   s_correspond      correspond[CONNECTIVITE];
} s_patient;
```

```
#include <stdio.h>
#include "struct.h"
#define PATIENT    "/usr/gast/rhino/jojo/test."

static FILE  *fp_read, *fp_append, *fp_write, *fp_list, *fp_verify;
static s_patient        patient_courant;
static s_correspond     scorrespond;
static long             ptr;


main()
{
   s_patient        patient;
   s_identification  ident;
   int              i;
   char             c;

   garnir_liste();
   while (1)
   {
      saisie_patient(&ident);
      if (!recherche_patient(&ident))
      {
         printf("Ce patient n'existe pas et va etre cree\n");
         copy_patient(&(patient.ident),&ident);
         ajout_patient(&patient);
      }
      patient_en_cours(&patient);

      i = 0;
      while (!i)
      {
         printf("\n Voulez vous faire des saisies pour ce patient (o/n) : ");
         while ((c = getchar()) != 'n')
         {
            saisie_correspond();
            ajout_correspond();
            maj_patient(&patient_courant);
         }
         i = 1;
      }
      list_patient();
   }
}

patient_en_cours(patient)
s_patient *patient;
{
   int i;

   copy_patient(&(patient_courant.ident),&(patient->ident));
   patient_courant.nombre_mesure = patient->nombre_mesure;
   for (i = 0; i < CONNECTIVITE; i++)
   {
      copy_correspond(&(patient_courant.correspond[i]),&(patient->correspond[i]));
   }
   printf("\n");
}

copy_patient(ident_pat1,ident_pat2)
```

```
s_identification  *ident_pat1, *ident_pat2;
{
   strcpy(ident_pat1->nom,ident_pat2->nom);
   strcpy(ident_pat1->prenom,ident_pat2->prenom);
   strcpy(ident_pat1->date_nais,ident_pat2->date_nais);
}

creation_patient(n,patient)
s_patient *patient;
int   n;
{
   int i;

   fp_write = fopen(PATIENT,"w");
   i = 0;
   while (i++ < n)
   {
      saisie_patient(&(patient->ident));
      write_patient(fp_write,patient);
   }
   fclose(fp_write);
}

write_patient(fp,patient)
FILE   *fp;
s_patient *patient;
{
    if (fwrite(patient,sizeof(s_patient),1,fp) != 1) printf("\neureka");
}

saisie_patient(ident)
s_identification *ident;
{
   int i;

   printf("\n\nNom    : ");
   scanf("%s",ident->nom);
   printf("Prenom : ");
   scanf("%s",ident->prenom);
   printf("Date de naissance (JJ/MM/AA) : ");
   scanf("%s",ident->date_nais);
}

read_patient(fp,patient)
FILE      *fp;
s_patient *patient;
{
   if ((!feof(fp)) && (!ferror(fp)))
       fread(patient,sizeof(s_patient),1,fp);
}

recherche_patient(ident)
s_identification  *ident;
{
   int       found;
   s_patient ident1;

   found = 0;
   fp_read   = fopen(PATIENT,"r");
   while ((!found) && (!feof(fp_read)))
```

```
   {
      read_patient(fp_read,&ident1);
      if (strcmp(ident1.ident.nom,ident->nom) == 0)
      {
         if (strcmp(ident1.ident.prenom,ident->prenom) == 0)
         {
            if (strcmp(ident1.ident.date_nais,ident->date_nais) == 0)
            {
               found = 1;
               ptr = ftell(fp_read) - (long)sizeof(s_patient);
               return(found);
            }
         }
      }
   }
   fclose(fp_read);
   return(found);
}

ajout_patient(patient)
s_patient   *patient;
{
      fp_append = fopen(PATIENT,"a");

      write_patient(fp_append,patient);
      ptr = ftell(fp_append) - (long)sizeof(s_patient);
      fclose(fp_append);
      printf("fin ajout patient\n");
}

maj_patient(patient)
s_patient   *patient;
{
      fp_append = fopen(PATIENT,"r+");

      fseek(fp_append,ptr,0);
      write_patient(fp_append,patient);
      fclose(fp_append);
}

list_patient()
{
   s_patient   patient;

   fp_list   = fopen(PATIENT,"r");
   while ((!feof(fp_list)) && (!ferror(fp_list)))
   {
      read_patient(fp_list,&patient);
      printf("%s %s %s\n",patient.ident.nom,patient.ident.prenom,patient.ident.date_nais);
   }
   fclose(fp_list);
}

verifier_liste(liste)
char liste[];
{
   int i, j;
   s_patient patient;

   fp_verify = fopen(PATIENT,"r");
```

```
   while (!feof(fp_verify))
   {
      read_patient(fp_verify,&patient);
      for (i = 0; i < patient.nombre_mesure; i++)
      {
         i = patient.correspond[i].num;
         liste[i] = '1';
      }
   }
   fclose(fp_verify);
}

ajout_correspond()
{
   int nombre;

   scorrespond.num = demande_nbre();
   if (patient_courant.nombre_mesure + 1 > CONNECTIVITE)
   {
      shift_correspond();
      copy_correspond(&(patient_courant.correspond[9]),&scorrespond);
      patient_courant.nombre_mesure = CONNECTIVITE;
   }
   else patient_courant.nombre_mesure++;
}

copy_correspond(cor1,cor2)
s_correspond *cor1, *cor2;
{
   strcpy(cor1->date,cor2->date);
   cor1->type = cor2->type;
   cor1->num = cor2->num;
}

saisie_correspond()
{
   printf("\nDate de saisie : "); scanf("%s",scorrespond.date);
   printf("Type de saisie : ");scanf("%c",scorrespond.type);
}

shift_correspond()
{
   int i;

   for (i = 0; i < 8; i++)
   {
      copy_correspond(&(patient_courant.correspond[i]),&(patient_courant.correspond[i + 1]));
   }
}
```

```
#include <stdio.h>
#define MAX   5000
#define ERROR -1
#define LISTFILE "/usr/gast/rhino/jojo/list"

static  char   liste[MAX];
static  int    premier_vide, nbre_utilise;

init_liste()
{
   int i;

   for (i = 0; i < MAX; i++) liste[i] = '0';
   premier_vide = nbre_utilise = 0;
}

demande_nbre()
{
   int i, j;

   if  (nbre_utilise == MAX - 1) return(ERROR);
   i = premier_vide;
   j = i + 1;
   while (liste[j] != '0') j++;
   premier_vide = j;
   liste[i] = '1';
   nbre_utilise++;
   return(i);
}

rendre_nbre(n)
int n;
{
   if (n < premier_vide) premier_vide = n;
   liste[n] = '0';
   nbre_utilise--;
}

sauver_liste()
{
   FILE  *fp;

   fp = fopen(LISTFILE,"w");
   fwrite(liste,sizeof(liste),1,fp);
   fclose(fp);
}

garnir_liste()
{
   FILE  *fp;

   fp = fopen(LISTFILE,"r");

   fread(liste,sizeof(liste),1,fp);
   verifier_liste(liste);
   fclose(fp);
}
```

MODULE ECRAN

```
#include <stdio.h>
#include "err_msg.h"
#include "struct.h"
#include "9754.h"
#include "files.h"
#include <ctype.h>

char   nom[35], prenom[35], date[15];
static s_identification patient;
char *date_du_jour;
char *imp_date();

clr_err()
{
   G_FILLRECT(1,16,1,1,638);
}

clr_body()
{
   G_FILLRECT(1,419,1,21,638);
}

clr_title()
{
   G_FILLRECT(1,479,1,424,638);
}

clr_all()
{
   clr_body();
   clr_title();
}

clr_all1()
{
   clr_body();
   clr_title();
   clr_err();
}

def_all()
{
   G_DRAWRECT(0,17,0,0,639);
   G_DRAWRECT(0,420,0,20,639);
   G_DRAWRECT(0,480,0,423,639);
}

rm_fmt(i,c)
int  i,c;
{
   int  y1, y2, y3, y4;
   switch (i)
   {
      case 1: y1 = 423; y2 = 420; y3 = 20; y4 = 17; break;
      case 2: y1 = 465; y2 = 463; y3 = 17; y4 = 15; break;
   }

   G_COLORS(c,c);
   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,c);
```

```
   G_LINE(3,y1,1,y1,648);
   G_LINE(3,y2,1,y2,648);
   G_LINE(3,y3,1,y3,648);
   G_LINE(3,y4,1,y4,648);
   G_COLORS(G_C_GREEN,G_C_GREEN);
   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
}

clr_err2()
{
   G_FILLRECT(1,14,1,1,638);
}

clr_body2()
{
   G_FILLRECT(1,462,1,16,638);
}

clr_title2()
{
   G_FILLRECT(1,479,1,466,638);
}

clr_all20()
{
   clr_body2();
   clr_title2();
}

clr_all2()
{
   clr_body2();
   clr_title2();
   clr_err2();
}

def_all2()
{
   G_DRAWRECT(0,15,0,0,639);
   G_DRAWRECT(0,463,0,17,639);
   G_DRAWRECT(0,480,0,465,639);
}

bell(i)
int  i;
{
   while (i-- > 0) printf("\7");
}

gestion_ecr1()
{
   s_variable  v;
   int         i;
   BOOLEAN     err, err_data;
   char        c;


   def_all();
   clr_err();
```

```
   v.type = INT;
   err    = TRUE;
   err_data = FALSE;

   while (err)
   {
      /* Gestion des msg err lors d'une mauvaise entree */
      if (!err_data)
      {
         clr_all1();
         ecran1();
      }
      else trt_error(1,1);

      G_GREAT(0);
      G_DRAWCAR(0,0,0);
      v.type = INT;
      get_data(&v,1,60,520,FALSE);
      err_data = FALSE;
      clr_err();
      clr_car(60,520,G_C_WHITE,G_C_MAGENTA);
      i = v.uval.ival;
      switch(i)
      {
         case  1  : clr_all1();
                    gestion_ecr11();
                    break;
         case  2  : gestion_ecr12();
                    break;
         case  3  : gestion_ecr13();
                    break;
         case  4  : gestion_ecr14();
                    break;
         case  9  : G_CLRWIN;
                    G_GROFF;
                    cls();
                    cursor(10,30);
                    err = FALSE;
                    break;
         default  : bell(10);
                    trt_error(1,0);
                    err_data = TRUE;
                    break;
      }
   }
}

gestion_ecr11()
{
   s_variable  v;
   int         i;
   BOOLEAN     err;
   char        jj[3], mm[3], aa[3], *jma, *jm;

   clr_all1();
   def_all();
   err    = TRUE;

   while (err)
   {
```

```
      clr_all();
      ecran11();
      v.type = ALPHABET;
      get_data(&v,30,340,248,FALSE);
      strcpy(patient.nom,v.uval.pval);
      get_data(&v,30,300,248,FALSE);
      strcpy(patient.prenom,v.uval.pval);
    * get_data(&v,2,260,380,FALSE);
      strcpy(patient.date_nais,v.uval.pval);

      /*
      jm  = strcat(jj,mm);
      jma = strcat(jm,aa);
      */

      err = FALSE;
   }
}

gestion_ecr12()
{
   int       rep, alpha;

   gestion_ecr120(&rep,&alpha);
   clr_all1();
   def_all();
   ecran12();
   trt_pat(rep,alpha,&patient);
   G_COLORS(G_C_MAGENTA,G_C_MAGENTA);
   G_COLWIND(0);
   G_COLORS(G_C_GREEN,G_C_GREEN);
   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
   rm_fmt(2,G_C_MAGENTA);
   clr_all2();
   def_all();
}

gestion_ecr120(rep,alpha)
int     *rep, *alpha;
{
   s_variable var;

   clr_all1();
   def_all();
   ecran120();
   *rep   = 75;
   *alpha = 5;
   var.type = INT;
   var.uval.ival = *rep;
   get_data(&var,5,300,80 + 38 * 8,TRUE); *rep   = var.uval.ival;
   var.uval.ival = *alpha;
   get_data(&var,2,260,80 + 38 * 8,TRUE); *alpha = var.uval.ival;
}

gestion_ecr13()
{
   char tampon[100];

   clr_all1();
```

```
   def_all();
   ecran12();
   sprintf(tampon,"%s1",nom);
   visual('1',2,3,G_C_GREEN,"jojo91.idx");
   sprintf(tampon,"%s2",nom);
   visual('1',2,3,G_C_YELLOW,"jojo92.idx");
   G_COLORS(G_C_MAGENTA,G_C_MAGENTA);
   G_COLWIND(0);
   G_COLORS(G_C_GREEN,G_C_GREEN);
   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
   rm_fmt(2,G_C_MAGENTA);
   clr_all2();
   def_all();

}

gestion_ecr14()
{
   char  ident[100];
   clr_all1();
   def_all();
   ecran14(&patient);
   date_du_jour = imp_date();
   strcpy(ident,patient.nom);
   strcat(ident," ");
   strcat(ident,patient.prenom);
   strcat(ident," ");
   strcat(ident,date_du_jour);
   c_plot(10,ident);
}

get_data(var,len,y,x,defaut)
s_variable  *var;
int         len, y, x;
BOOLEAN     defaut;
{
   int     i, j, l, ypos, xpos;
   char    c[31];
   BOOLEAN boucle;


   for (i = 0; i < 31; i++) c[i] = ' ';
   i = l = 0;
   boucle = TRUE;
   ypos = y;
   xpos = x;

   G_CURSIPOS(y,x);
   if (defaut)
   {
      switch (var->type)
      {
         case INT      :
                   printf("%d",var->uval.ival);
                   break;

         case ALPHABET :
                   if (len == 1) printf("%c",var->uval.cval);
                   else          printf("%s",var->uval.pval);
```

```
                    break;

            case ALPHANUM :
                    printf("%s",var->uval.pval);
                    break;
        }
    }

    G_CURSIPOS(y,x);
    switch (var->type)
    {
        case INT      :
                while (boucle && (l < len) )
                {
                    c[l++] = _getcbcar();
                    G_CURSIPOS(y,xpos);
                    printf("%c",c[i]);
                    if ((i == 0) && (c[i] != '\n') && defaut)
                        for (j = 0; j <= len; j++)
                        {
                            clr_car(y,x+j*9,G_C_WHITE,G_C_MAGENTA);
                            G_CURSIPOS(y,xpos);
                            printf("%c",c[i]);
                        }
                    xpos += 8;
                    if ((!isdigit(c[i])) && (c[i] != '\n'))
                    {
                        trt_err(c[i],y,xpos,G_C_WHITE,G_C_MAGENTA);
                        l--; i--; xpos -= 8;
                    }
                    if (c[i] == '\n')
                    {
                        if (i > 0) boucle = FALSE;
                        else
                        {
                            if (!defaut) {i--; l--; xpos -= 8;}
                            else boucle = FALSE;
                        }
                    }
                    i++;
                }
                c[l] = '\0';
                var->uval.ival = ((i == 1) && (defaut)) ? var->uval.ival : atoi(c);
                break;

        case ALPHABET :
                /*
                while (boucle && (l < len) )
                {
                    c[l++] = _getcbcar();
                    G_CURSIPOS(y,xpos);
                    printf("%c",c[i]);
                    if ((i == 0) && (c[i] != '\n') && defaut)
                        for (j = 0; j <= len; j++)
                        {
                            clr_car(y,x+j*9,G_C_WHITE,G_C_MAGENTA);
                            G_CURSIPOS(y,xpos);
                            printf("%c",c[i]);
                        }
                    xpos += 8;
```

```
                if ((!isalpha(c[i])) && (c[i] != '\n') && (c[i] != ' '))
                }
                   trt_err(c[i],y,xpos,G_C_WHITE,G_C_MAGENTA);
                   l--; i--; xpos -= 8;
                {
                if (c[0]== '\n')
                {
                   if (i > 0) boucle = FALSE;
                   else
                   {
                      if (!defaut) {i--; l--; xpos -= 8;}
                      else boucle = FALSE;
                   }
                }
                i++;
             {
             c[l] = '\0';
             */
             gets(c);
             if (len == 1)
                var->uval.cval = ((i == 1) && (defaut)) ? var->uval.cval : c[0];
             else
                var->uval.pval = ((i == 1) && (defaut)) ? var->uval.pval : c;
             break;

   case ALPHANUM :
             while (boucle && (l < len) )
             {
                c[l++] = getchcar();
                G_CURSIPOS(y,xpos);
                printf("%c",c[i]);
                if ((i == 0) && (c[i] != '\n') && defaut)
                   for (j = 0; j <= len; j++)
                   {
                      clr_car(y,x+j*9,G_C_WHITE,G_C_MAGENTA);
                      G_CURSIPOS(y,xpos);
                      printf("%c",c[i]);
                   }
                xpos += 8;
                if (!isalnum(c[i]))
                {
                   trt_err(c[i],y,xpos,G_C_WHITE,G_C_MAGENTA);
                   l--; i--; xpos -= 8;
                }
                if (c[i] == '\n')
                {
                   if (i > 0) boucle = FALSE;
                   else
                   {
                      if (!defaut) {i--; l--; xpos -= 8;}
                      else boucle = FALSE;
                   }
                }
                i++;
             }
             c[l] = '\0';
             var->uval.pval = ((i == 1) && (defaut)) ? var->uval.pval : c;
             break;
   }
}
```

```
trt_err(c,y,x,color_fore,color_back)
char    c;
int     x, y, color_fore, color_back;
{
   bell(5);
   clr_car(y,x,color_fore,color_back);
   G_CURS1POS(y, x);
}

clr_car(y,x,color_fore,color_back)
int     y, x, color_fore, color_back;
{
   G_CURSOFF;
   G_COLORS(G_C_MAGENTA,G_C_MAGENTA);
   G_DRAWRECT(0,y+13,x,y,x + 9);
   G_COLFOREG(0,color_fore);
   G_COLBACK(0,color_back);
   G_FILLRECT(0,y+13,x,y,x+8);
   G_COLORS(G_C_GREEN,G_C_GREEN);
   G_CURSON;
   G_CURS2POS(641,481);
}
```

```
   G_COLBACK(4,G_C_MAGENTA);
   G_CURSIPOS(435,80);

   printf("%s",MSG_ECR110);

   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
   G_CURSIPOS(390,150); printf("%s",MSG_ECR111);
   G_CURSIPOS(340,80); printf("%s",MSG_ECR112);
   G_CURSIPOS(300,80); printf("%s",MSG_ECR113);
   G_CURSIPOS(260,80); printf("%s",MSG_ECR114);
}

ecran120()
{

   date_du_jour = img_date();

   G_GREAT(0);
   G_COLFOREG(4,G_C_WHITE);
   G_COLBACK(4,G_C_MAGENTA);
   G_CURSIPOS(435,80);

   printf("%s",MSG_ECR120);

   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
   G_CURSIPOS(390,80); printf("%s",MSG_ECR121);
   G_CURSIPOS(300,80); printf("%s",MSG_ECR122);
   G_CURSIPOS(260,80); printf("%s",MSG_ECR123);

}

ecran12()
{

   int i;
   char c;

   rm_fmt(1,G_C_MAGENTA);
   G_COLORS(G_C_GREEN,G_C_GREEN);
   G_COLFOREG(0,G_C_WHITE);
   G_COLBACK(0,G_C_MAGENTA);
   def_all2();
   clr_all2();
   date_du_jour = img_date();
   G_GREAT(0);
   G_CURSIPOS(460,120);

   printf("%s",MSG_ECR120);

/* ------------------------------------------------------------------------ */
/* grid                                                                     */
/* ------------------------------------------------------------------------ */
   G_LINE(3,MID_Y,0,MID_Y,MAX_X - TIRET); /* horizontal axes    */
/* ------------------------------------------------------------------------ */
/* right arrow                                                              */
/* ------------------------------------------------------------------------ */
   G_LINE(3,MID_Y - 2 * TIRET,MAX_X -TIRET - 2 * TIRET,MID_Y,MAX_X - TIRET);      /* up part */
   G_LINE(3,MID_Y + 2 * TIRET,MAX_X - TIRET - 2 * TIRET,MID_Y,MAX_X -TIRET);      /* down part */
```

```
   for (i = 0; i< MAX_Y;i +=20)   /* horizontal tiret */
   {
      G_LINE(3,i,MID_X - TIRET,i,MID_X + TIRET);       /* horizontal */
   }
/* ------------------------------------------------------------------------ */
/* vertical     axes                                                        */
/* ------------------------------------------------------------------------ */
   G_LINE(3, 16,MID_X,MAX_Y - 19 - TIRET,MID_X);
/* ------------------------------------------------------------------------ */
/* UP  ARROW                                                                */
/* ------------------------------------------------------------------------ */
   G_LINE(3,MAX_Y - 19 - TIRET - 2 * TIRET,MID_X - 2 *  TIRET,MAX_Y -19 - TIRET ,MID_X );
 /* left part */
   G_LINE(3,MAX_Y - 19 - TIRET - 2 * TIRET,MID_X + 2 * TIRET,MAX_Y - 19 - TIRET ,MID_X );      /* right part */
   for ( i = 0 ; i< MAX_X ; i += 20)    /* vertical tiret */
   {
      G_LINE(3,MID_Y - TIRET,i,MID_Y + TIRET,i);
   }
/* ------------------------------------------------------------------------ */
/* circle                                                                   */
/* ------------------------------------------------------------------------ */
   G_COLORS(G_C_CYAN,G_C_CYAN);
   for ( i = 40 ; i<200; i +=40)  /* circle */
        {
         G_DRAWCIRC(0,MID_Y,MID_X,i,MID_X);
        }
}

ecran13()
{

   date_du_jour = img_date();

   G_GREAT(0);
   G_COLFOREG(4,G_C_WHITE);
   G_COLBACK(4,G_C_MAGENTA);
   G_CURSIPOS(435,80);

   printf("%s",MSG_ECR130);

   G_COLBACK(4,G_C_WHITE);
   G_COLFOREG(4,G_C_MAGENTA);
   G_GREAT(2);
   G_CURSIPOS(250,150); printf("EN COURS DE DEVELOPPEMENT");
}

ecran14(patient)
s_identification *patient;
{

   date_du_jour = img_date();

   G_GREAT(0);
   G_COLFOREG(4,G_C_WHITE);
   G_COLBACK(4,G_C_MAGENTA);
   G_CURSIPOS(435,80);

   printf("%s",MSG_ECR140);

   G_COLBACK(4,G_C_WHITE);
```

```
   G_COLFOREG(4,G_C_MAGENTA);
   G_GREAT(2);
   G_CURS1POS(250,150); printf("TRANSFERT EN COUR");
   G_GREAT(0);
   G_CURS1POS(200,30);printf("Nom    : %s",patient->nom);
   G_CURS1POS(180,30);printf("Prenom : %s",patient->prenom);
   G_CURS1POS(160,30);printf("Date de naissance : %s",patient->date_nais);
}
```

MODULE MATHEMATIQUE

```
#define v1   10
#define v2   10

static real f_distribution5[v1][v2] =
      {
        161.00, 200.00, 216.00, 225.00, 230.00, 234.00, 237.00, 239.00, 241.00, 242.00,
         18.51,  19.00,  19.16,  19.25,  19.30,  19.33,  19.36,  19.37,  19.38,  19.39,
         10.13,   9.55,   9.28,   9.12,   9.10,   8.94,   8.88,   8.84,   8.81,   8.78,
          7.71,   6.94,   6.59,   6.39,   6.26,   6.16,   6.09,   6.04,   6.00,   5.96,
          6.61,   5.79,   5.41,   5.19,   5.05,   4.95,   4.88,   4.82,   4.78,   4.74,
          5.99,   5.14,   4.76,   4.53,   4.39,   4.28,   4.21,   4.15,   4.10,   4.06,
          5.59,   4.74,   4.35,   4.12,   3.97,   3.87,   3.79,   3.73,   3.68,   3.63,
          5.32,   4.46,   4.07,   3.84,   3.69,   3.58,   3.50,   3.44,   3.39,   3.34,
          5.12,   4.26,   3.86,   3.63,   3.48,   3.37,   3.29,   3.23,   3.18,   3.13,
          4.99,   4.10,   3.71,   3.48,   3.33,   3.22,   3.14,   3.07,   3.02,   2.97,
      }

static real f_distribution01[v1][v2] =
      {
        4052.00, 4999.00, 5403.00, 5625.00, 5764.00, 5859.00, 5928.00, 5981.00, 6022.00, 6056.00,
          98.49,   99.00,   99.17,   99.25,   99.30,   99.33,   99.36,   99.37,   99.39,   99.40,
          34.12,   30.62,   29.46,   28.71,   28.24,   27.91,   27.67,   27.49,   27.34,   27.23,
          21.20,   18.00,   16.69,   15.98,   15.52,   15.21,   14.98,   14.80,   14.66,   14.54,
          16.26,   13.27,   12.06,   11.39,   10.97,   10.67,   10.45,   10.29,   10.15,   10.05,
          13.74,   10.92,    9.78,    9.15,    8.75,    8.47,    8.26,    8.10,    7.98,    7.87,
          12.25,    9.55,    8.45,    7.85,    7.46,    7.19,    7.00,    6.84,    6.71,    6.62,
          11.26,    8.65,    7.59,    7.01,    6.63,    6.37,    6.19,    6.03,    5.91,    5.82,
          10.56,    8.02,    6.99,    6.42,    6.06,    5.80,    5.62,    5.47,    5.25,    5.26,
          10.04,    7.56,    6.55,    5.99,    5.64,    5.39,    5.21,    5.06,    4.95,    8.85,
       }
```

```
#include "files.h"
#include <math.h>
#include <stdio.h>

double tab_dist[NBRE_COURBES];
int    table_presence[NBRE_COURBES + 1];
double vecteur_1[NBRE_COURBES + 1], vecteur_2[NBRE_COURBES + 1];


agglomeration_hierarchique(distance, hierarchie, ptr, taby)
s_distance    *distance;
s_hierarchie  *hierarchie;
s_lines       ptr[];
s_table       *taby;
{
   agg_hie('C',taby,distance,hierarchie,ptr);

   agg_hie('D',taby,distance,hierarchie,ptr);
}

agg_hie(type,taby,distance,hierarchie,ptr)
char          type;
s_distance    *distance;
s_hierarchie  *hierarchie;
s_lines       ptr[];
s_table       *taby;
{

   double        somme[NBRE_COURBES + 1], min_dist, table[NBRE_COURBES + 1][NBRE_COURBES + 1];
   double        h_distance[NBRE_COURBES + 1][NBRE_COURBES + 1];
   double        matvar[NBRE_COURBES + 1][NBRE_COURBES + 1];
   s_table       *temp;
   s_hierarchie  h;
   int           i, j, k, l, nc, indice_h, s, nx;


   temp = taby;

   if (type == 'C')
   {
      for (i = 0; i < ptr[0].nbre; i++)
         for (j = 0; j < ptr[0].nbrex; j++)
         {
            table[i][j]      = (double)(taby->line[i].column[j]);
         }

      for (i = 0; i < ptr[1].nbre; i++)
         for (j = 0; j < ptr[1].nbre; j++)
         {
            h_distance[i][j] = (i > j) ? distance->d2[j][i] : distance->d2[i][j];
         }
   }
   else
   {
      for (i = NBRE_COURBES, k = 0; i <= ptr[1].indice; i++, k++)
         for (j = 0; j < ptr[1].nbrex; j++)
         {
            taby = temp;
            table[k][j] = (double)(taby->line[i].column[j]);
```

```
        }

    for (i = 0; i < ptr[1].nbre; i++)
        for (j = 0; j < ptr[1].nbre; j++)
        {
            h_distance[i][j] = (i > j) ? distance->d2[i][j] : distance->d2[j][i];
        }
}

scr_taby(taby,ptr);
nc = (type == 'C') ? ptr[0].nbre : ptr[1].nbre;
nx = (type == 'C') ? ptr[0].nbrex : ptr[1].nbrex;

/* AGGLOMERATION HIERARCHIQUE DES COURBES */
/* 1. Mise a zero de la table de presence et table des distances */

/*
scr_table_presence(table_presence,ptr[0].nbre);
*/

for (i = 0; i <= NBRE_COURBES; i++)
{
    table_presence[i] = FALSE;
    tab_dist[i] = 0.0;
    somme[i] = 0.0;
}

/* 2. Recherche de la distance minimum */

min_dist = h_distance[0][1];
h.c[0].numero   = 0;
h.c[1].numero   = 1;
h.c[1].distance = min_dist;

for (i = 0; i < nc; i++)
{
    for (j = i + 1; j < nc; j++)
    {
        if (h_distance[i][j] < min_dist)
        {
            min_dist = h_distance[i][j];
            h.c[0].numero   = i;
            h.c[1].numero   = j;
            h.c[1].distance = min_dist;
        }
    }
}

indice_h = 2;

/* 3. garnissage de vecteur_1 et vecteur_2 */

k = h.c[0].numero;
l = h.c[1].numero;
table_presence[k] = table_presence[l] = TRUE;

for (i = 0; i < nx; i++)
{
    vecteur_1[i] = table[k][i];
    vecteur_2[i] = table[l][i];
```

```
   )

   /* 4. calcul de la moyenne et mise dans vecteur_1 */

   printf("Impression du vecteur_1\n");
   scr_vecteur(vecteur_1,nc);

   printf("Impression du vecteur_2\n");
   scr_vecteur(vecteur_2,nc);

   for (i = 0; i < nx; i++)
   {
      somme[i]     = (vecteur_1[i] + vecteur_2[i]);
      vecteur_1[i] = somme[i] / 2;
   }

   printf("Impression de la moyenne du vecteur_1 et du vecteur_2\n");
   scr_vecteur(vecteur_1,nc);

   printf("debut agglomeration point 5\n");
   /* 5. agglomeration hierarchique */

   while ((s = somme_table_presence(table_presence, nc)) != nc)
   {
      /* 5.1 calcul de la distance entre les autres courbes et le vecteur_1 */

      printf("n = %d ",nc);
      printf("somme = %d \n",s);

      /*
      scr_table_presence(table_presence,NBRE_COURBES);
      */

      fill_cvar_h(table, nx, s, matvar, &h);

      printf("Impression de la moyenne du vecteur_1 et du vecteur_2\n");
      scr_vecteur(vecteur_1, nx);

      printf("impression des Yi de la ieme courbe\n");
      scr_vecteur(table[i],nx);

      for (i = 0; i < nc; i++)
      {
         if (!table_presence[i])
         {
            tab_dist[i] = 0;

            for (k = 0; k < nx; k++)
            {
               tab_dist[i] += (pow(vecteur_1[k] - table[i][k], 2.0) / matvar[k][k]);
            }
         }
      }

      scr_tab_dist(tab_dist,nc);

      /*
      scr_taby(taby,ptr);
      */
```

```
      /* 5.2 recherche du premier i tq table_presence[i] = 0 */

      i = 0;

      while ((table_presence[i]) && (i < nc)) i++;

      /* 5.3 recherche de la distance minimum */

      min_dist = tab_dist[i];

      for (j = i; j < nc; j++)
      {
         if (!table_presence[j])
         {
            if (tab_dist[j] <= min_dist)
            {
               min_dist = tab_dist[j];
               h.c[indice_h].numero   = j;
               h.c[indice_h].distance = min_dist;
            }
         }
      }

      /* 5.4 nouveau calcul de la moyenne  et reinitialisation pour la prochaine iteration */

      k = h.c[indice_h].numero;

      for (i = 0; i < nx; i++)
      {
         somme[i] += table[k][i] ;
         vecteur_i[i] = (somme[i] / (s + 1));
      }

      /*
      scr_vecteur(vecteur_i,nc);
      scr_table_presence(table_presence,nc);
      */

      /*
      printf("k = %d ",k);
      */

      table_presence[k] = TRUE;
      tab_dist[k] = 0.0;
      /*
      printf(" table_presence[%d] = %d table_presence[%1d] = %d\n",k,TRUE,k,table_presence[k]);
      printf("k = %d ",k);
      scr_table_presence(table_presence,nc);
      */

      indice_h++;

      /*
      printf(" h = %d",indice_h);
      */
   }

   if (type == 'C')
   {
      for (i = 0; i < ptr[0].nbre; i++)
```

```
      {
         hierarchie->c[i].numero   = h.c[i].numero;
         hierarchie->c[i].distance = h.c[i].distance;
      }
   }
   else
   {
      for (i = 0; i < ptr[1].nbre; i++)
      {
         hierarchie->d[i].numero   = h.c[i].numero;
         hierarchie->d[i].distance = h.c[i].distance;
      }
   }

   /*
   scr_hierarchie(hierarchie, ptr[0].nbre, ptr[1].nbre);
   */
}

somme_table_presence(table_presence, n)
int  table_presence[], n;
{
   int i, somme;

   for (somme = 0, i = 0; i < n; i++) somme += table_presence[i];
   return(somme);
}
```

```
#include "files.h"
#include <math.h>

fill_cvar(stable, varcov, r, ptr)
s_table    *stable;
s_covar    *varcov, *r;
s_lines    ptr[];
{
   int    indice_i, indice_j, indicecv_l, indicecv_c, nc, nd, indice_k;
   double somme_xik_xjk, somme_xik, somme_xjk;
   double varcov_i_j;

   /* indice_i, indice_j : indices des differents Xi
      indice_k           : indice des courbes
      indicecv_l         : indice ligne de varcov et r
      indicecv_c         : indice colonne de varcov et r
      nc, nd             : nbre de courbes 'C' et 'D' respectivement
   */

   /* calcul variance/covariance et r pour 'C' */
   nc = ptr[0].nbrex;

   for (indice_i = indicecv_l = 0; indice_i < nc; indice_i++, indicecv_l++)
   {
      for (indice_j = indicecv_c = 0; indice_j < nc; indice_j++, indicecv_c++)
      {
         somme_xik_xjk = somme_xik = somme_xjk = 0.0;
         for (indice_k = 0; indice_k <= ptr[0].indice; indice_k++)
         {
            /*
            printf("y[%1d][%1d] = %5d ",indice_k,indice_i,stable->line[indice_k].column[indice_i]);
            printf("y[%1d][%1d] = %5d ",indice_k,indice_j,stable->line[indice_k].column[indice_j]);
            */

            somme_xik_xjk += (double)(stable->line[indice_k].column[indice_i] * stable->line[indice_k].column[indice_j]);

            /*
            printf("sxijk = %.6e ",somme_xik_xjk);
            */

            somme_xik     += (double)stable->line[indice_k].column[indice_i];

            /*
            printf("sik = %.6e ",somme_xik);
            */

            somme_xjk     += (double)stable->line[indice_k].column[indice_j];

            /*
            printf("sjk = %.6e\n ",somme_xjk);
            */
         }

         /* calcul de la matrice variance/covariance */

         varcov->vc[indicecv_l].c[indicecv_c] = somme_xik_xjk - (somme_xik * somme_xjk / ptr[0].nbre);

         /*
         printf("v[%1d][%1d] = %.6e\n\n",indicecv_l,indicecv_c,varcov->vc[indicecv_l].c[indicecv_c]);
         */
```

```
      }
   }

   /* calcul de la matrice de correlation r */

   for (indicecv_l = 0; indicecv_l < nc; indicecv_l++)
   {
      for (indicecv_c = 0; indicecv_c < nc; indicecv_c++)
      {
         /*
         printf("v[%ld][%ld] = %.6e ",indicecv_l,indicecv_c,varcov->vc[indicecv_l].c[indicecv_c]);
         printf("v[%ld][%ld] = %.6e ",indicecv_l,indicecv_l,varcov->vc[indicecv_l].c[indicecv_l]);
         printf("v[%ld][%ld] = %.6e ",indicecv_c,indicecv_c,varcov->vc[indicecv_c].c[indicecv_c]);
         */

         varcov_i_j = sqrt((double)(varcov->vc[indicecv_l].c[indicecv_l] * varcov->vc[indicecv_c].c[indicecv_c]));

         /*
         printf("v_i_j = %.6e ",varcov_i_j);
         */

         r->vc[indicecv_l].c[indicecv_c] = varcov->vc[indicecv_l].c[indicecv_c] / varcov_i_j;

         /*
         printf("r[%ld][%ld] = %.6e \n",indicecv_l,indicecv_c,r->vc[indicecv_l].c[indicecv_c]);
         */
      }
   }

   /* calcul variance/covariance et r pour 'D' */

   nd = ptr[1].nbrex;

   for (indice_i = indicecv_l = 0; indice_i < nd; indice_i++, indicecv_l++)
   {
      for (indice_j = indicecv_c = 0; indice_j < nd; indice_j++, indicecv_c++)
      {
         somme_xik_xjk = somme_xik = somme_xjk = 0.0;
         for (indice_k = NBRE_COURBES; indice_k <= ptr[1].indice; indice_k++)
         {
            somme_xik_xjk += (double)(stable->line[indice_k].column[indice_i] * stable->line[indice_k].column[indice_j]);
            somme_xik     += (double)stable->line[indice_k].column[indice_i];
            somme_xjk     += (double)stable->line[indice_k].column[indice_j];
         }

         /* calcul de la matrice variance/covariance */

         varcov->vc[indicecv_l].d[indicecv_c] = somme_xik_xjk - (somme_xik * somme_xjk / ptr[1].nbre);

      }
   }

   /* calcul de la matrice de correlation r */

   for (indicecv_l = 0; indicecv_l < nd; indicecv_l++)
   {
      for (indicecv_c = 0; indicecv_c < nd; indicecv_c++)
      {
         if (indicecv_l == indicecv_c)
```

```
            r->vc[indicecv_l].d[indicecv_c] = 1.0;
         else
         {
            varcov_i_i = sqrt((double)(varcov->vc[indicecv_l].d[indicecv_l] * varcov->vc[indicecv_c].d[indicecv_c]));
            r->vc[indicecv_l].d[indicecv_c] = varcov->vc[indicecv_l].d[indicecv_c] / varcov_i_i;
         }
      }
   }
}

fill_cvar_h(table, nx, n, matvar, h)
double        table[][NBRE_COURBES + 1], matvar[][NBRE_COURBES + 1];
int           n, nx;
s_hierarchie  *h;
{
   int    indice_i, indice_j, k, indice_k;
   double somme_xik_xjk, somme_xik, somme_xjk;
   double varcov_i_i;

   /* indice_i, indice_j : indices des differents Xi
      indice_k           : indice des courbes
   */


   for (indice_i = 0; indice_i < nx; indice_i++)
   {
      for (indice_j = 0; indice_j < nx; indice_j++)
      {
         /*
         printf("\ni = %d j = %d\n",indice_i,indice_j);
         */

         somme_xik_xjk = somme_xik = somme_xjk = 0.0;
         for (indice_k = 0; indice_k < n; indice_k++)
         {

            k = h->c[indice_k].numero;

            somme_xik_xjk += (table[k][indice_i] * table[k][indice_j]);

            /*
            printf("k = %d ",k);
            */

            /*
            printf("sxijk = %.6e ",somme_xik_xjk);
            */

            somme_xik     += table[k][indice_i];

            /*
            printf("sik = %.6e ",somme_xik);
            */

            somme_xjk     += table[k][indice_j];

            /*
            printf("sjk = %.6e\n",somme_xjk);
            */
```

```
        }

        /* calcul de la matrice variance/covariance */

        matvar[indice_i][indice_j] = somme_xik_xjk - (somme_xik * somme_xjk / n);

        /*
        printf("matvar[%2d][%2d] = %.6e\n\n",indice_i,indice_j,matvar[indice_i][indice_j]);
        */
      }
   }
   scr_matvar(matvar,nx);printf("\n");
}
```

ACCES_RHINO

```
#include <signal.h>
#include <stdio.h>
#define START_SEQ "st--\r" /* -- is the number of ms between scan .. */
#define STOP_SEQ  "ha\r"
#define ASK  write(fd_rhino,"\r",1)  /* sequence to ask byte */
#define DEBUG 1
char *rhino ;
time_out(sig)
int sig;
{
 signal(sig,time_out);
}
static  int fd_rhino ;
static  int tty ;
open_rhino()
{
 if (( fd_rhino = open(rhino,2)) < 0 )
        {
         fprintf(stderr,"Cannot open %s (r & w )\n",rhino);
         exit(1);
        }
 tty = isatty(fd_rhino) ;
 stty_rhino(fd_rhino);
}
read_rhino(x,y)
short *x,*y;
{
 static union tt
           {
         char buff[6] ; /* so no multiple of aligement */
         struct st
                   {
                  short a ;
                  short b ;
                  char z[2];
                 } st ;
        } tt ;
 int r = 0,l ; /* number of char read */
#if DEBUG
 if (!tty)
     {
      int i;
      read(fd_rhino,y,sizeof(y));
      read(fd_rhino,x,sizeof(x));
      for (i = 0 ; i < 10000;i++);
      return;
     }
#endif
 ASK ;
 signal(SIGALRM,time_out);
 while ( r < 5 )
        {
         alarm(2);
         if ((l = read(fd_rhino,&tt.buff[r],5 - r)) <= 0 )
                {
                 alarm(0);
                 ASK;
                 continue;
                }
         alarm(0);
```

```
        r += 1;
        if (r == 5 && tt.buff[4] != '\r' )
               {
                printf(".");
                r = 0;
                ASK;
               }
       }
 *x = tt.st.b;
 *y = tt.st.a;
}
start_rhino(sc)
short sc ;
{
 short *a;
 char *b = START_SEQ;
#if DEBUG
 if (!tty)
     {
      return;
     }
#endif
 a = (short *) &b[2]; /* !!!!!!!!!!!!!! */
 *a = sc;
 return(write(fd_rhino,b,5));
}
stop_rhino()
{
#if DEBUG
 if (!tty)
     {
      return;
     }
#endif
 write(fd_rhino,STOP_SEQ,3);
}
close_rhino()
{
 stop_rhino();
 close(fd_rhino);
}
#include <sgtty.h>
stty_rhino(fd)
int fd;
{
 struct sgttyb st;
#if DEBUG
 if (!tty)
     {
      return;
     }
#endif
 gtty(fd,&st);
 st.sg_flags |= RAW ;
 st.sg_flags &=~ECHO;
 st.sg_ispeed = B19200 ;
 st.sg_ospeed = B19200 ;
 stty(fd,&st);
}
```

ACCES_TRACEUR

```
#define P_PEN_SEL(n) {print("F,%d\r",n);}
        /* select pen  0 <= n <= 8 */
#define P_PEN_SPEED(n)  {print("F 10,%d\r",n);}
        /* select speed of pen          */
        /* n = 32 or 32                 */
#define P_PEN_UP        {print("H\r");}
        /* set the pen UP               */
#define P_PEN_DOWN      {print("I\r");}
        /* set the pen DOWN             */
#define P_COORD(x,y)    { print("C %d %d\r",x,y);}
        /* store those coordinates but not move */
#define P_ORIG(x,y)     { print("T %d %d\r",x,y);}
        /* define user origin */
#define P_SCALE(sx,sy)  { print("T 2 %d %d\r",sx,sy);}
        /* define scale factor for x */
        /* can be Float             */
#define P_CLIP_AERA(xmin,xmax,ymin,ymax) { print("W %d %d %d %d\r",xmin,xmax,ymin,ymax);}
        /* define clipping maximun aera */
#define P_VERI_CLIP {print("V\r");}
        /* draw a rectangle in clipping aera */
#define P_TYPE_LIG(n,l) { print("L %d %d\r",n,l);}
        /* n = 0 _________              */
        /* n = 1 .........              */
        /* n = 2 ---------              */
        /* n = 3 -.-.-.-.-.-.-.-.       */
        /* n = 4 _ _ _ _ _ _ _ _        */
        /* n = 5 _ . . _ . . _ . . _ .  */
        /* l = 0..255 size in milimetre of basis line */
#define P_PLOT  {print("K\r");}
        /* move or plot a line to current position */
#define P_LINE(y1,x1,y2,x2) {P_PEN_UP;P_COORD(x1,y1);P_PLOT;P_PEN_DOWN;\
                P_COORD(x2,y2);P_PLOT;}
        /* draw a line from x1,y1,x2,y2 */
#define P_REL_PLOT {print("J\r");}
        /* move relative with preceding coordinate */
#define P_DIR_SIZE(h,a,w) {print("Z %d %d %d\r",h,a,w);}
        /* h = hight of letter init = 72  unit 1/10 mm */
        /* a = line angle of printing line unit degrees */
        /* w = width of the character lettres  unit 1/10 mm */
#define P_ITALIC(n) { print("%%%d\r",n);}
        /* n = 0 or 1  0 is default */
        /* 0 = vertical   1 = 75 degrees form horizontal */
#define P_FONT(n) {print("#%d\r",n);}
        /* n = 0  standart ascii */
        /* n = 1  GERMAN         */
        /* n = 2  SPANISH        */
        /* n = 3  SWEDISH,FINNISH */
        /* n = 4  DANISH         */
        /* n = 5  FRENCH         */
#define P_CHAR(str) { print("B%s\r",str);}
        /* plot a string without  \r */
        /* next line is a \n   */
#define P_PT_MARK(n) { print("M %d\r",n);}
        /* plot a special symbol */
        /* 0 = square    */
        /* 1 = triangle  */
        /* 2 = X         */
        /* 3 = +         */
        /* 4 = Y         */
#define P_B_CIRCLE(R,a,b) { print("O %d %d %d\r",R,a,b);}
```

```
        /* plot circle or arc ; bord is current position */
        /* R = radius of circle or arc */
        /* a = starting angle in degrees */
        /* b = ending angle in degrees */
#define P_CIRCLE(R,a,b) { print("OD %d %d %d\r",R,a,b);}
        /* plot circle or arc ;centre is current position */
        /* R = radius of circle or arc */
        /* a = starting angle in degrees */
        /* b = ending angle in degrees */
#define P_SECTOR(R,a,b) { print("OI %d %d %d\r",R,a,b);}
        /* plot a sector ;centre is current position */
        /* R = radius of circle or arc */
        /* a = starting angle in degrees */
        /* b = ending angle in degrees */
#define P_XAXE(l,D,t1,t2) { print("X %d %d %d %d\r",l,D,t1,t2);}
        /* draw an axe of X */
        /* l = length of axis   */
        /* d = tick mark distance */
        /* t1 = starting point of tick mark */
        /* t2 = ending   point of tick mark */
#define P_YAXE(l,D,t1,t2) { print("Y %d %d %d %d\r",l,D,t1,t2);}
        /* draw an axe of Y */
        /* l = length of axis   */
        /* d = tick mark distance */
        /* t1 = starting point of tick mark */
        /* t2 = ending   point of tick mark */
#define P_RESET(N) { print("R %d\r",N); }
        /* n = 0 : end plot set pen upper right no  param reset */
        /* n = 1 : end plot set pen upper right plotter is set to wait status */
        /* n = 2 : all parametre are set to initial value */
#define P_GET_POS { print("!\r");}
        /* asked to send to actual position */
        /* format send is 's XXXXXX s YYYYYY' */
#define P_ON_PLOTTER { print("\01P\r");}
        /* command to plotter on */
#define P_CLOSE_PLOTTER { close_plotter(); }
#define P_OPEN_PLOTTER { open_plotter(); }
```

```
#include "err_msg2.h"
#include <stdio.h>

static FILE *fd_plot;
static int _P_STATUS = 0;

open_plotter()
{
   /*
   if (_P_STATUS == 0)
   {
   */
      if ((fd_plot = fopen("/dev/ife1600","w")) == 0)
      {
         printf("%s %s \n",ERR_MSG1,"/dev/ife1600");
         exit(-1);
      }
      _P_STATUS = 1;
   /*
   }
   */
}

close_plotter()
{
   fclose(fd_plot);
   _P_STATUS = 0;
}

print(form,strg1,strg2,strg3,strg4,strg5,strg6,strg7,strg8,strg9)
char *form,*strg1,*strg2,*strg3,*strg4,*strg5,*strg6,*strg7,*strg8,*strg9;
{

   if (_P_STATUS == 0) open_plotter();
   fprintf(fd_plot,form,strg1,strg2,strg3,strg4,strg5,strg6,strg7,strg8,strg9);
}
```

ACCES_ECRAN

```
#undef ESC
#define ESC             '\033'                  /*   ESCape */

#define CSI             "\033["                 /* CSI sequence */
#define PCSI            "\033[?"                /* Private CSI sequence */

#define G_RESET                         printf("%cc",ESC)
        /* G_RESET terminal */
#define G_GRON                        printf("%s1a",PCSI)
        /* graphic on */
#define G_GROFF                       printf("%s2a",PCSI)
        /* graphic off */
#define G_CLRWIN                      printf("%s2D",PCSI)
        /* clear windown */
#define G_TSTSCRN                       printf("%s#",PCSI)
        /* test screen */
#define G_CURSON                        printf("%s1j",PCSI)
        /* cursor ON */
#define G_CURSOFF                       printf("%s0j",PCSI)
        /* cursor off */
#define G_CURSHOME                      printf("%s0C",PCSI)
        /* cursor on home (0,0) */
#define G_CURS1POS(y,x)                 printf("%s1;%d;%dC",PCSI,y,x)
/* new position of cursor1 = y,x   */
/* 1480                                 */
/* |  Y                                 */
/* 10,0                    X            */
/* +------------------------------ 640   */
#define G_CURS2POS(y,x)                 printf("%s2;%d;%dC",PCSI,y,x)
        /* new position of cursor2 = y,x   */
#define G_CUR1SADV(dy,dx)               printf("%s3;%d;%dC",PCSI,dy,dx)
        /* new position of cursor1 = old_y1 + dy , old_x1 + dx */
        /* RELATIVE MOVE                */
#define G_CURS2ADV(dy,dx)               printf("%s4;%d;%dC",PCSI,dy,dx)
        /* new position of cursor2 = old_y2 + dy , old_x2 + dx */
        /* RELATIVE MOVE                */

#define G_WINDHOME                      printf("%s0c",PCSI)
        /* position of window is HOME (0,0) */

#define G_WINDPOS(y,x)                  printf("%s0;%d;%dc",PCSI,y,x)
        /* position of windown is y,x */
#define G_WINDADV(dy,dx)                printf("%s2;%d;%dc",PCSI,dy,dx)
        /* new position of windown = old_y2 + dy , old_x2 + dx */
        /* RELATIVE MOVE                */

#define G_DEFPIX(a1,a2,a3,a4)         printf("%s%d;%d;%d;%dd",PCSI,a1,a2,a3,a4)
        /* When using command G_PAINT describe pattern to G_PAINT */
        /*      a       b       c       d       */
        /*   XXXX     XXXX    XXXX    XXXX       */
        /*ex:1101     1110    0011    0110       */
        /*ex:13       14      3       6          */
#define G_DEFTYPE(a)                    printf("%s%dl",PCSI,a)
        /* define the form of the line for vector and rectangle */
        /* 0 and 1 are standart options */
        /* 11111111111111111111111... 0  */
        /* 11111111111111111111111... 1  */
        /* 11001100110011001100110... 2  */
        /* 11110000111100001111000... 3  */
        /* 11100100111001001110010... 4  */
```

```
        /* 1111111000111000111111... 5  */
        /* 1111111110101010111111... 6  */
        /* 1111110011111100111111O... 7  */
        /* 1111110011001100111111O... 8  */

#define G_DEFCOLOR(a,r,g,b)             printf("%s%d;%d;%d;%dm",PCSI,a,r,g,b)
        /* define color                 */
        /* a   0-7      number of color normal */
        /*    10-17     number of color blinking */
        /* r  0-7  quantities of red */
        /* g  0-7  quantities of green */
        /* b  0-3  quantities of blue  */

#define G_DEFBLIN(a,b)                  printf("%s%d;%dn",PCSI,a,b)
        /* define color which must blinking */
        /*  a  0-8                              */
        /*     0-7 color                        */
        /*     8   pause                        */
        /*  b  0-999 time between blink         */
        /*      unit : 1/16 ms                  */

#define G_COLWIND(a)                    printf("%s%dD",PCSI,a)
        /* give a color to the window           */
        /* This command work in graphic and alpha mode   */
        /* a = 0 fills the window with main color        */
        /*     1 delete the main color into window       */
        /*     2 delete all the screen at actual level   */
        /*     3 delete all the G_COLORS in all    level   */

#define G_COLORS(a,b)                   printf("%s%d;%df",PCSI,a,b)
        /* define main secondary color          */
        /* a  = number of the main color */
        /* b  = number of the secondary color */

#define G_COLFOREG(a,b)                  printf("%s%d;3%dm",CSI,a,b)
        /* define the foreground color for character in graphic mode */
        /* a = attribute     */
        /*      0 = normal      */
        /*      1 = hight intensity */
        /*      4 = underline       */
        /*      7 = reverse video   */
        /* b =  0-7 main color          */
        /*      10-17 for color 8 -15     */

#define G_COLBACK(a,b)                   printf("%s%d;4%dm",CSI,a,b)
        /* define the background color for character in graphic mode */
        /* a = attribute     */
        /*      0 = normal      */
        /*      1 = hight intensity */
        /*      4 = underline       */
        /*      7 = reverse video   */
        /* b =  0-7 main color          */
        /*      10-17 for color 8 -15     */
#define G_DRAW(a,y,x)                     printf("%s%d;%d;%dP",PCSI,a,y,x)
        /* plot a point at current position     */
        /* a = 2 plot a point at cursor 1       */
        /*     3 erase a point at cursor 1       */
        /*     4 inverse point at cursor 1       */
        /*     5 plot a point at cursor 2        */
        /*     6 erase a point at cursor 2       */
```

```
        /*      7 inverse point at cursor 2       */

#define G_DRAWVSP(a)                        printf("%s%dV",PCSI,a)
        /* a = 0 note the current position of cursor 1 as start position */
        /*     1 switch in mode 1 the end position is the new start position */
        /*     2 switch in mode 2 the startposition is for all vector cursor1 */
        /*     3 plot a vector with cursor 1                                  */
        /*     4 drop a vector with cursor 1                                  */

#define G_DRAWVECT(a,y,x)                    printf("%s%d;%d;%dV",PCSI,a,y,x)
        /* plot lines                             */
        /* a = 0 note the current position of cursor 1 as start position */
        /*     3 plot a vector with cursor 1                                  */
        /*     4 drop a vector with cursor 1                                  */

#define G_LINE(a,y1,x1,y2,x2)   printf("\033[?1;%d;%dC\033[?0V\033[?1V\033[?%d;%d;%dV",y1,x1,a,y2,x2)
        /* plot lines                             */
        /* a = 0 note the current position of cursor 1 as start position */
        /*     1 switch in mode 1 the end position is the new start position */
        /*     2 switch in mode 2 the startposition is for all vector cursor1 */
        /*     3 plot a vector with cursor 1                                  */
        /*     4 drop a vector with cursor 1                                  */

#define G_DRAWRECT(a,y1,x1,y2,x2)                  printf("%s%d;%d;%d;%d;%dR",PCSI,a,y1,x1,y2,x2)
        /* plot a rectangle                                  */
        /* a = 0 plot a rectangle contour                    */
        /*     1 erase a rectangle mais pas le rmepissage    */
        /*     4 fill with --------                          */
        /*     5 fill with points                            */
        /*     6 fill with vertical bar                      */
        /*     7           horizontal                        */
        /*     8           uphash                            */
        /*     9           downhash                          */

#define G_DRAWCIRC(a,y1,x1,y2,x2)                  printf("%s%d;%d;%d;%d;%dK",PCSI,a,y1,x1,y2,x2)
        /* plot a circle                                     */
        /* a = 0 plot a circle                               */
        /*     1 erase a circle                              */
#define G_DRAWSEGM(a,b)                 printf("%s%d;%dS",PCSI,a,b)
        /* drop a arc of circle                              */
        /* a = 0 plot a arc of circle                        */
        /*     1 erase a arc of circle                       */
        /*     2 plot a arc of circle with segment           */
        /*     3 erase a arc of circle with segment          */
        /* b = 0-360 the number of degrees                   */

#define G_FILLRECT(a,y1,x1,y2,x2)                  printf("%s%d;%d;%d;%d;%dF",PCSI,a,y1,x1,y2,x2)
        /* fill a rectangle */
        /* a = 0 fill rectangle with full G_PAINT              */
        /*     1 erase full rectangle                        */
        /*     4 fill with --------                          */
        /*     5 fill with points                            */
        /*     6 fill with vertical bar                      */
        /*     7           horizontal                        */
        /*     8           uphash                            */
        /*     9           downhash                          */

#define G_PAINT(a,y,x)                      printf("%s%d;%d;%dI",PCSI,a,y,x)
        /*  fill a undefine surface                   */
        /* a = 0 from cursor 1                        */
```

```
        /*      1 erase from cursor 1                   */
        /*      2 from cursor 2                         */
        /*      3 erase from cursor 2                   */
        /*      4 fill with --------                            */
        /*      5 fill with points                              */
        /*      6 fill with vertical bar                        */
        /*      7           horizontal                          */
        /*      8           uphash                              */
        /*      9           downhash                            */

#define G_ZOOM(a)                       printf("%s%dz",PCSI,a)
        /* perform a G_ZOOM on image from cursor 1               */
        /* a = 0-15  = G_ZOOM factor                             */
#define G_DRAWCAR(a,b,c)                printf("%s%d;%d;%dw",PCSI,a,b,c)
        /* attributes for characters in graphics mode */
        /* a = 0 normal  */
        /*     1 italic  */
        /* b = 0  horizontal*/
        /*     1  45     */
        /*     2  90     */
        /*     3  135    */
        /*     4  180    */
        /*     5  225    */
        /*     6  270    */
        /*     7  315    */
        /* c = 0  ascii  */
        /*     1  graphic*/

#define G_GREAT(a)                      printf("%s%dT",PCSI,a)
        /* large of characters */
        /* 0  :normal */
        /* 1  :dubble width */
        /* 2  :dubble width and hight  */
#define G_C_BLACK   0
#define G_C_RED     1
#define G_C_GREEN   2
#define G_C_YELLOW  3
#define G_C_BLUE    4
#define G_C_MAGENTA 5
#define G_C_CYAN    6
#define G_C_WHITE   7

#define G_INIT54 (G_RESET;G_CLRWIN;G_GROFF;G_CURSOFF;G_CURSHOME;G_WINDHOME;G_COLWIND(3);)
```

ACCES_FICHIER

MANUEL UTILISATEUR

Manuel de l'Utilisateur

1. Introduction

Le logiciel dévéloppé répond aux spécifications de fonctions définies en 3 (deuxième partie du mémoire) et a été conçu et développé pour les ordinateurs à écran graphique de la série 9754 de Siemens.

Avant d'exécuter le programme, vérifier que les appareils de saisies sont allumés et connectés à l'ordinateur.

2. Exécution du programme

Si l'ordinateur n'est pas en marche, mettez-le en route. Quand apparaitra la phrase "Identification :" sur l'écran, tappez le mot **gast**[1] suivi de RETURN. Apparaitra ensuite la phrase "Mot de passe :", entrez le mot **siemens**[2] suivi de RETURN. Ensuite tappez **cd rhino** suivi de RETURN, puis tappez **graphess** suivi de RETURN et le logiciel s'exécutera.

Si l'ordinateur est déjà en allumé, tappez :

- **cd /usr/gast/rhino** suivi de RETURN puis tappez **trt_pat** suivi de RETURN pour exécuter le logiciel.

2.1 Le programme en cours d'exécution

Quand le logiciel commence à s'exécuter, l'écran ecr0 apparaitra .Pour choisir une option, tappez le chiffre correspondant à l'option choisie.

2.1.1 Option 1

L'écran ecr1 apparaitra. Entrez le nom, prenom et date de naissance[3] du patient. Ce patient deviendra le patient courant

---

[1]Ecrit en minuscule

[2]Ecrit en minuscule

[3]Après chaque entrée, tappez RETURN

(et le restera tant qu'on n'introduit pas les coordonnées d'un autre patient) et dès lors tout les options 2,3 et 4 de ecr0 ne pourront être effectuées que pour ce patient, c'est à dire, pour le patient identifié par le nom, prénom et date de naissance introduit, on pourra saisir et traiter ses données, visualiser les données prises lors des consultations précédentes et transférer ses données vers un traceur. Après la saisie des coordonnées du patient, le programme revient à l'écran ecr0.

2.1.2 Option 2

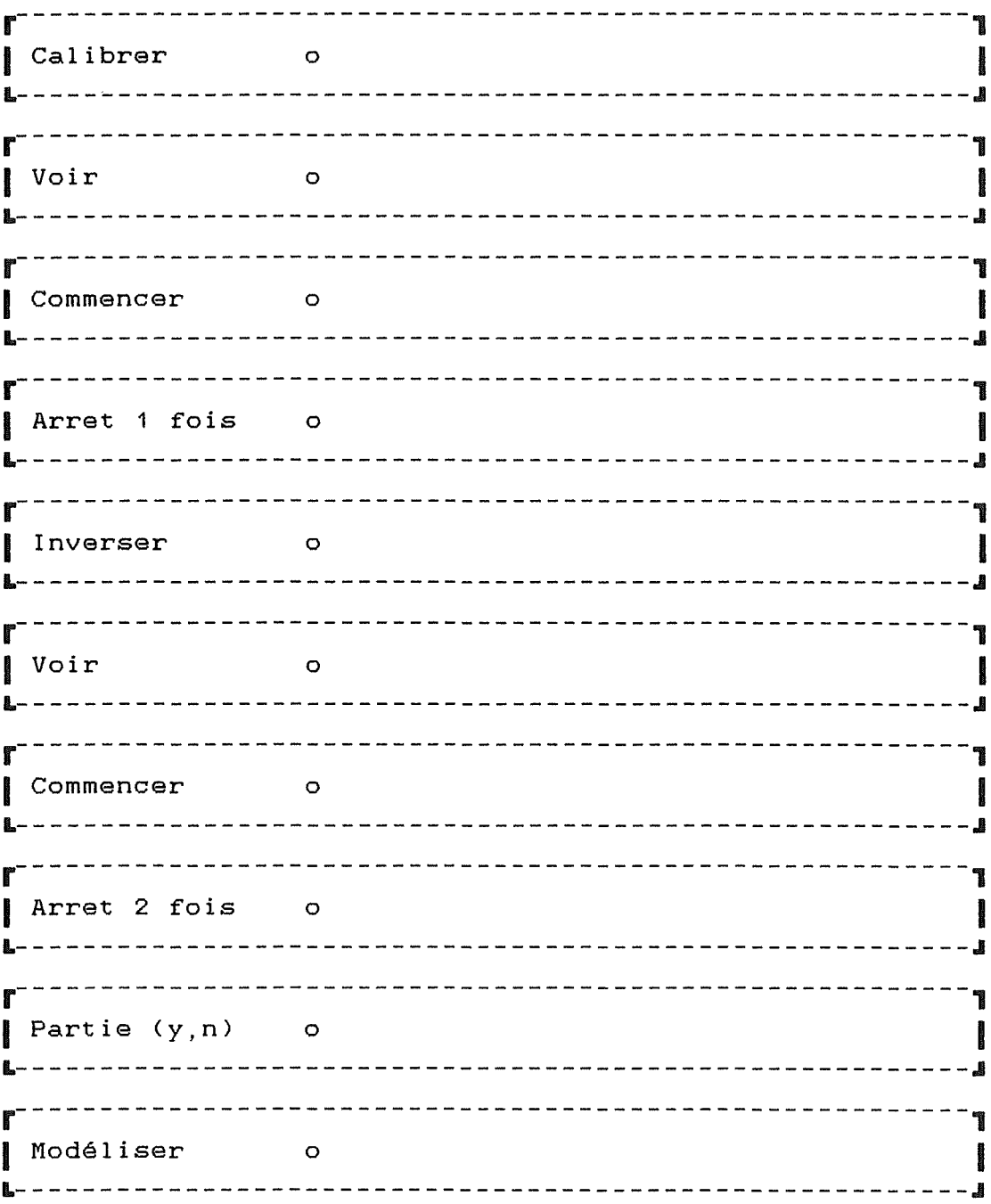
L'écran ecr2 apparait. Il faut entrer le nombre de milli-secondes entre chaque échantillonage, (qui est fixé par défaut à 75), l'interval de confiance (qui lui est fixé à 5 %) et le type de mesure. Si les valeurs par défaut sont choisies, tappez RETURN 2 fois, sinon entrez les nouvelles valeurs. En ce qui concerne le type de mesure, il a été laissé au choix de l'utilisateur de déterminer le code qu'il veut utiliser. Ce code est toutefois limité à 1 caractère.

Ensuite apparaitra l'écran ecr21. Comme on peut le constater, cet écran est composé de 3 parties: la partie entête où se trouve l'intitulé de l'écran et la date du jour, la partie où apparaitra les courbes obtenues lors de la prise de mesure, et la partie commande où se trouve les commandes qu'on peut utiliser à un moment donné. Pour choisir une commande, il suffit de tappez le premier caractère du mot de la commande.

Example

Pour voir, il suffit de tappez **V**. A tout moment de la saisie, on peut abandonner en tapant **O**, mais alors les données prises ne sont pas enregistrées.

L'écran ecr21 permet la saisie et le traitement des données et ne changera dans le temps que par sa ligne de commande et qui se succèdera comme suit:



La commande **Calibrer** comme son nom l'indique, permet de calibrer l'ordinateur pour la prise de mesure.

Remarquons, que lors de la prise de mesure, on commence par la fosse nasale gauche. La commande **Voir** permet de voir un début du tracé obtenu afin de permettre au patient de s'adapter à l'appareil. Pendant ce temps, il n'y a pas de stockage des données.

La commande suivante **Commencer** permet de commencer vraiment la prise de mesure. Il y a un stockage temporaire des données. Ces dernières ne seront enregistrées qu'à la fin de la prise de mesure. Il y a deux moyens d'arrêter la prise de mesure:

- soit tapper **Arrêt**
- soit laisser le programme se dérouler jusqu'à ce que la commande **Inverser** apparaisse.

Il est préférable d'attendre que cette dernière apparaisse car généralement elle permet de prendre un plus grand nombre de données, avec l'inconvénient de pousser le patient dans ses extrêmes limites de respiration.

Après avoir tappé **Arrêt** (si on l'a fait), la commande **Inverser** apparait. Il faut inverser les canules. De nouveau on passe par les étapes de **Voir**, **Commencer** et **Arrêt** . Pour la deuxième partie de la prise de mesure, si on ne tappe pas **Arrêt**, sur l'écran apparaitra la commande **Partie(y,n)** qui permet à l'utilisateur de choisir telle ou telle partie des courbes. Mais attention ici, il faut entrer **y** ou **n**.

Après la prise de mesure, il ne reste qu'à traiter les données. Ceci est fait grâce à la commande **Modéliser**.

Une fois la saisie et le traitement terminé, le programme revient à l'écran **ecr21** initial, et se prépare pour une nouvelle prise de mesure. S'il ne faut plus faire de prise de mesure sur le patient courant, il suffit d'entrer **o** pour que le programme revienne à l'écran **ecr0**.

2.1.3. Option 3

Cette option permet la visualisation des courbes d'un patient à l'écran. L'écran ecr3 permet de choisir le type de visualisation désirée. Entrez le numéro correspondant à l'option choisie.

2.1.4. Option 4

Cette option permet d'envoyer vers un traceur les courbes d'un patient. L'écran ecr4 permet de choisir le type de tracée désirée. Vérifiez que le traceur est à la position PLOT , puis entrez le numéro correspondant à l'option choisie.

RHINOMANOMETRIE

1. Saisie des coordonnées d'un patient

2. Saisie et traitement des données d'un patient

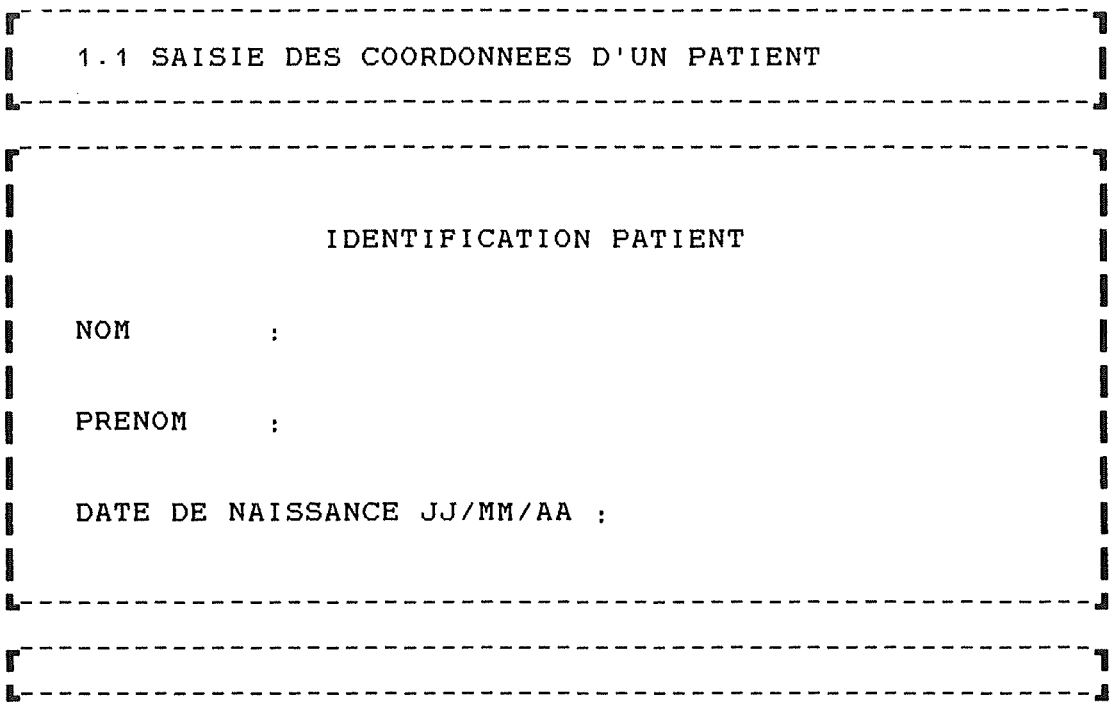
3. Visualisation des courbes d'un patient

4. Transfert vers un plotter

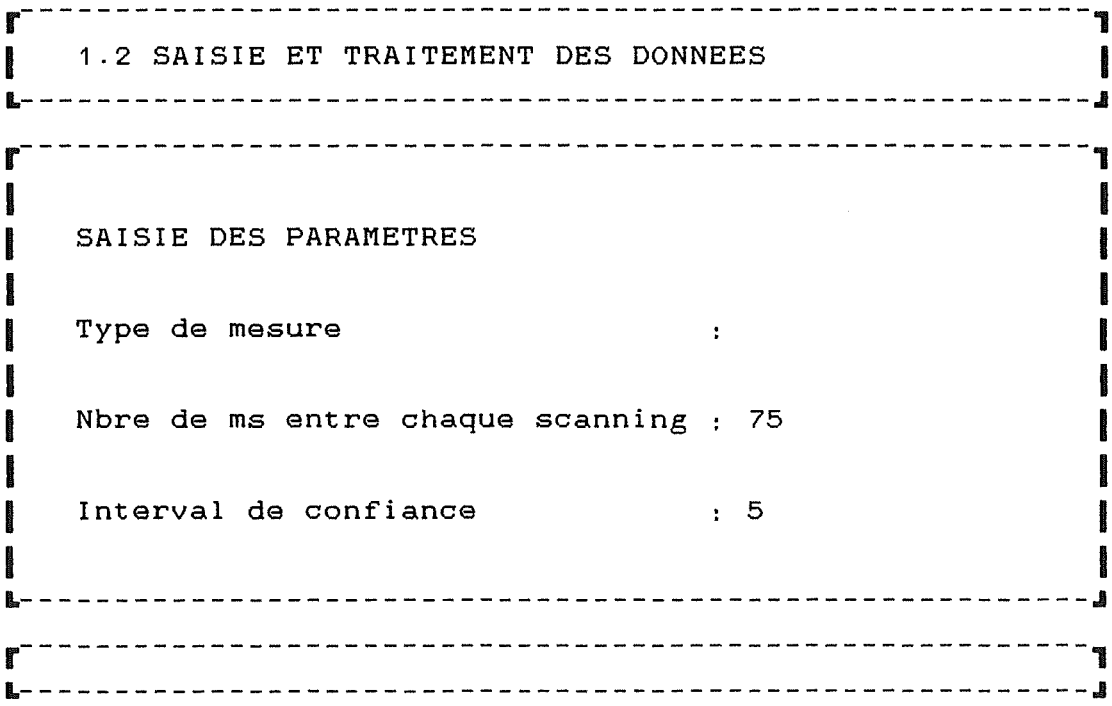
9. Retour au menu précédent

Votre choix :

ecr0


1.1 SAISIE DES COORDONNEES D'UN PATIENT
IDENTIFICATION PATIENT
NOM :
PRENOM :
DATE DE NAISSANCE JJ/MM/AA :

ecr1


1.2 SAISIE ET TRAITEMENT DES DONNEES
SAISIE DES PARAMETRES
Type de mesure :
Nbre de ms entre chaque scanning : 75
Interval de confiance : 5

ecr2

1.2 SAISIE ET TRAITEMENT DES DONNEES

Dans ce bloc, apparaitra le tracé des courbes obtenues lors de la prise de mesure.

Partie commande

ecr21

1.3 VISUALISATION DES COURBES D'UN PATIENT

1. Visualiser la/les courbes enregistrée(s) lors de la dernière prrise de mesure.

2. Visualiser toutes les courbes prises sur le patient.

Votre choix :

ecr3

1.4 TRANSFERT SUR PLOTTER

1. Transférer la/les courbes prise(s) lors de la dernière prise de mesure vers le plotter.

2. Transférer toutes les courbes prises sur le patient vers le plotter

Votre choix :

ecr4