MASTER EN SCIENCES INFORMATIQUES

Développement d'un logiciel pour l'analyse du profil de raies d'absorption infrarouge

Coupé, P.

Award date:
1992

Awarding institution:
Universite de Namur

Développement d'un logiciel pour l'analyse du profil de raies d'absorption infrarouge

P. Coupé 1991-1992

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue de Bruxelles 61, B-5000 NAMUR
Tel. 081-72.41.11
Télex 59222 facnam-b
Téléfax 081-23.03.91

Développement d'un logiciel pour l'analyse du profil de raies d'absorption infrarouge

P. COUPE

Résumé

Les techniques modernes de spectroscopie moléculaire à très haute résolution, tels que les spectromètres diode-laser, permettent l'observation fine et précise de l'absorption moléculaire en phase gazeuse; l'interprétation de ces spectres d'absorption nécessitant des méthodes de plus en plus élaborées.
A cet effet, un système d'analyse des raies d'absorption infrarouge a été développé; ce système étant constitué d'un logiciel de traitement graphique des spectres et d'un logiciel d'ajustements de profils sur ces raies. La partie graphique de ce système, développée sur PC, a notamment permis d'améliorer la détermination du profil expérimental de la raie en rendant possible des déplacements du fond par rapport au spectre.
Le second logiciel réalise l'ajustement des profils suivants sur les données expérimentales : Doppler, collisionnel, Voigt, Voigt généralisé et Galatry. Les ajustements sont réalisés au sens des moindres carrés au moyen d'une procédure basée sur l'algorithme de Levenberg-Marquardt. Il est ainsi possible d'extraire un certain nombre de paramètres caractéristiques des différentes raies étudiées ainsi que de résoudre le problème de la superposition des raies non résolues.

Abstract

Modern technologies in very high-resolution spectroscopy, like tunable diode-laser spectrometers, allow sharp and accurate observation of molecular absorption in gazeous phase; interpretation of these absorption spectra requires more and more elaborated methods.
For this purpose, an infrared absorption lines analysis system has been developed; this system consists of a graphical spectra processing program and a spectral lines

profile fitting program. The graphical component of this system, developed on PC, allow in particular the improvement of the experimental profile determination by possible shifting of the continuum spectrum.
The second program performs profile fitting on experimental data : Doppler, collisionnal, Voigt, general Voigt and Galatry.The fits are performed by a least-squares procedure using the well-known Levenberg-Marquardt algorithm. It is now possible to extract caracteristical parameters from spectral lines and to resolve the problem of multiple overlapping spectral lines.

Mémoire de licence et maîtrise en Informatique
Année académique 1991-1992
Promoteurs : J.-P. LECLERCQ et G. BLANQUET

Je tiens à adresser mes plus vifs remerciements à toutes les personnes qui m'ont permis de mener à bien ce travail et particulièrement à

- Mr. J.-P. Leclercq pour avoir accepté de co-promouvoir ce travail ainsi que pour le suivi qu'il a apporté à sa rédaction.

- Mr. le professeur G. Blanquet, co-promoteur de ce mémoire, pour m'avoir accueilli dans son laboratoire ainsi que pour tous les conseils qu'il m'a prodigués tout au long de ce travail et durant ces 3 dernières années passées aux Facultés.

- Mr. J. Walrand pour son aide efficace et sa généreuse disponibilité.

- Tous les membres du laboratoire de spectroscopie moléculaire que j'ai côtoyés et qui m'ont permis de travailler dans une ambiance amicale au cours des années passées en leur compagnie.

- Mr. J.-P. Bouanich (Paris, Orsay) pour son aimable collaboration à la réalisation de ce travail.

Enfin, je remercie toutes les personnes qui m'ont aidé et soutenu durant ces 6 années passées aux Facultés et particulièrement ma famille.

Table des matières

Introduction

Jusqu'il y a quelques années, le dépouillement et l'étude des spectres d'absorption moléculaire étaient effectués manuellement; ce qui n'allait pas sans poser quelques inconvénients majeurs. Citons entre autres le fait que l'analyse des résultats ne pouvait se faire en temps réel, que des erreurs humaines pouvaient éventuellement venir s'ajouter aux erreurs expérimentales, ... L'avènement des techniques d'acquisition digitale a permis d'éliminer une grande partie de ces inconvénients et a offert aux expérimentateurs des mesures de plus en plus précises et reproductibles. De plus, la digitalisation de l'information conjuguée à l'évolution des environnements graphiques a ouvert de nouvelles perspectives dans le traitement des spectres d'absorption.

Ce travail qui s'est orienté selon deux axes a un double objectif. Il se propose tout d'abord de développer un système informatique permettant l'étude des profils d'absorption des spectres enregistrés au laboratoire de spectroscopie moléculaire. Ensuite, il se propose d'offrir un outil graphique sur PC permettant un traitement visuel, simple et rapide des spectres d'absorption.

Les mesures effectuées en spectroscopie moléculaire sont le résultat de l'observation de phénomènes physiques pour lesquels des modèles d'abord simples mais ensuite plus élaborés ont été développés. La précision actuellement atteinte dans ces mesures permet de faire ressortir des structures relativement complexes ainsi que des effets très fins prévus théoriquement depuis plusieurs années mais qui n'étaient pas observables jusqu'il y a peu.

Le premier objectif de ce travail est la mise au point d'un système d'analyse des mesures effectuées au laboratoire de spectroscopie moléculaire afin de déterminer, parmi un ensemble de modèles de profils, celui qui s'adapte le mieux aux données en fonction des conditions expérimentales. Les modèles qui sont étudiés dans ce travail sont tous des modèles non-linéaires; certains étant relativement simples comme le profil Doppler tandis que d'autres sont plus complexes comme les profils de Voigt généralisé et de Galatry. Les ajustements de profils sont effectués à l'aide d'une procédure d'ajustement au sens des moindres carrés non-linéaires basée sur l'algorithme de Levenberg-Marquardt pour la recherche de la solution optimale.

Le système d'analyse a été développé dans un premier temps pour l'étude de spectres de raies isolées. Ensuite, ce système a été adapté au cas de l'ajustement de profils sur des spectres à raies multiples afin d'en déduire les paramètres associés à chacune des raies prise individuellement.

Le second objectif de ce travail consiste en la réalisation d'un environnement graphique interactif permettant le traitement visuel des spectres d'absorption. Cet environnement permet de réaliser les opérations d'"ajustement" entre le fond et le spectre de manière instantanée; il permet également de visualiser les spectres d'absorption ainsi que les spectres redressés. Les résultats obtenus lors des ajustements de profils étant plus parlant lorsqu'on les représente de manière graphique que lorsqu'ils sont donnés sous leur forme brute (*i.e.* sous forme de nombres), nous avons inclu dans cet environnement la possibilité de les visualiser ainsi que de rendre compte de l'écart entre le spectre original et le spectre calculé sur base du modèle ajusté. Cette application utilise un outil de gestion d'interface (Turbo Vision) faisant intensément appel aux notions de programmation orientée objets; nous avons donc étudié cette technique de programmation que nous avons également utilisée dans le reste de l'application.

Après une brève présentation de l'existant logiciel, le premier chapitre présente les notions de spectroscopie moléculaire nécessaires à la compréhension de ce travail. Le second chapitre décrit le problème de la modélisation des données ainsi que la résolution des problèmes de moindres carrés non-linéaires. Le troisième chapitre explique les différents traitements que doivent subir les spectres d'absorption avant de pouvoir faire l'objet d'ajustements de profils; la réalisation de ces ajustements étant explicitée dans le chapitre IV. Dans le cinquième chapitre, les notions de base de la programmation orientée objets sont présentées; ces notions trouvent leur application dans le gestionnaire d'interface Turbo Vision qui est décrit dans le sixième et dernier chapitre de ce travail.

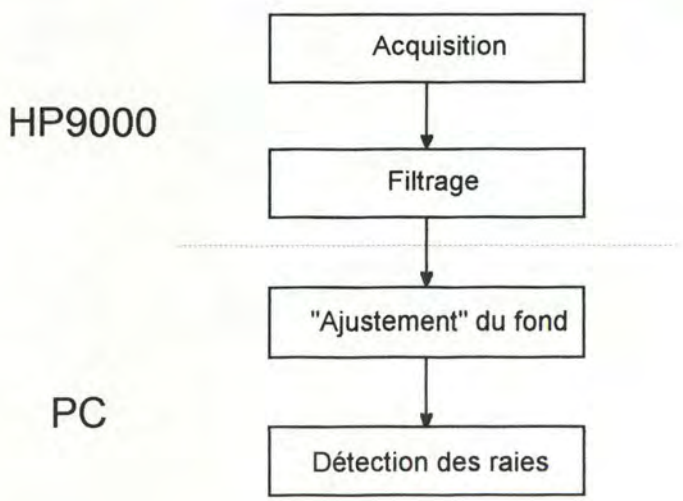
Description de l'existant et apport personnel

Nous présentons ici brièvement l'existant logiciel développé par les membres du laboratoire de spectroscopie moléculaire ainsi que la contribution apportée par ce travail dans le traitement des spectres d'absorption infrarouge. Les spectres d'absorption enregistrés au laboratoire suivent une chaîne de traitements qui sont présentés dans les chapitres III et IV de ce travail et auxquelles correspondent des programmes.

1. Description de l'existant

La chaîne des traitements effectués sur les spectres comprend les étapes suivantes : l'acquisition, le filtrage, l'"ajustement" du fond et la détection des raies. Les deux premières étapes sont effectuées sur un ordinateur HP9000 qui est couplé à un spectromètre diode-laser afin de réaliser les acquisitions. Ensuite, afin de disposer d'un environnement graphique pour ajuster le fond et le spectre, les fichiers de données sont transférés sur un PC (386 avec carte graphique VGA) à l'aide du logiciel Kermit. Chacune de ces étapes est réalisée de manière automatique par un programme (rédigé en FORTRAN) à l'exception de l'"ajustement" qui est effectué de manière interactive avec l'utilisateur.

Le schéma ci-dessous reprend les différentes phases disponibles avant la réalisation de ce travail.

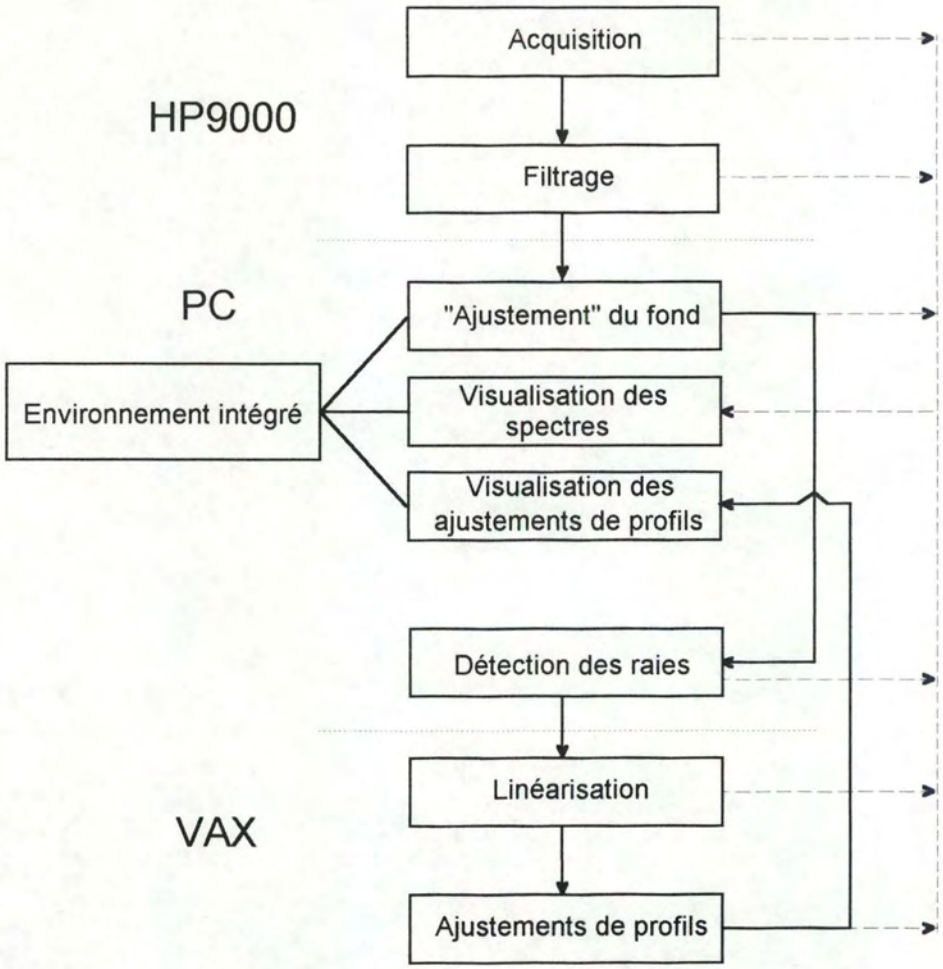


2. Contributions apportées par ce travail

Ce travail apporte essentiellement deux contributions. D'une part, il prolonge la chaîne des traitements des spectres en réalisant leur linéarisation en fréquence et en permettant d'effectuer des ajustements de profils sur ces spectres. Ces traitements sont réalisés sur le VAX des Facultés après que les fichiers de données y aient été transférés; les programmes qui leur sont associés sont écrits en FORTRAN. L'utilisation du VAX est motivée par le fait qu'il met à notre disposition les librairies mathématiques IMSL dont nous employons des routines dans nos programmes. Les interactions entre les deux programmes et l'utilisateur se limitent essentiellement au choix des raies à prendre en compte. Une fois ces traitements effectués, les fichiers sont retransférés sur PC afin de visualiser graphiquement les résultats obtenus.

D'autre part, ce travail a permis de regrouper au sein d'un environnement intégré standard sur PC les étapes d'"ajustement" du fond et de visualisation des résultats des ajustements de profils effectués sur le VAX. La phase d'"ajustement" du fond existante a notamment été enrichie par la possibilité d'effectuer des rotations du fond et de réaliser l'"ajustement" par rapport au spectre ou au spectre redressé. De plus, une phase de visualisation des spectres est incorporée dans cet environnement et est utilisable avec tous les spectres, quel que soit leur état d'avancement dans la chaîne des traitements. Cette partie du travail est réalisée en TURBO PASCAL orienté objets sur PC et est entièrement compatible avec les structures de fichiers existants antérieurement à son développement.

Le schéma de la page suivante reprend les différentes phases disponibles après la réalisation de ce travail.


HP9000
PC
VAX
Acquisition
Filtrage
"Ajustement" du fond
Environnement intégré
Visualisation des spectres
Visualisation des ajustements de profils
Détection des raies
Linéarisation
Ajustements de profils

Chapitre I

La spectroscopie moléculaire

1. Introduction

La spectroscopie moléculaire est une branche de la physique et de la chimie qui étudie la structure des molécules ainsi que les interactions entre atomes, molécules, électrons et photons[1]. On peut classer les travaux effectués en spectroscopie moléculaire selon deux catégories. D'une part, il y a les recherches fondamentales qui essayent d'interpréter et de comprendre comment les choses se passent réellement avec pour but de pouvoir modéliser la réalité. D'autre part, il y a les travaux qui, sur base des résultats des recherches fondamentales, permettent de déterminer l'identité et l'abondance des atomes et des molécules d'un échantillon ainsi que leur pression partielle, ... Ces travaux trouvent des applications dans des domaines divers tels que les études de pollution, les recherches de composants des atmosphères planétaires, ...

Le but de ce travail est de permettre de confronter un certain nombre de modèles de profils de raies d'absorption aux résultats expérimentaux enregistrés afin de déterminer des paramètres caractéristiques des molécules étudiées. Pour cela, un minimum de notions de spectroscopie moléculaire s'impose et est proposé dans ce chapitre.

2. La structure moléculaire

Jusqu'à la fin du siècle dernier, il était communément accepté que la matière possédait une quantité d'énergie qui pouvait prendre n'importe quelle valeur. Cependant, un certain nombre d'expériences réalisées à cette époque fournissaient des résultats incompatibles avec cette idée et il est apparu que l'énergie ne pouvait pas varier de manière continue mais bien de façon discrète. On dit que l'énergie est quantifiée et les valeurs qu'elle peut prendre pour une molécule particulière sont appelées les niveaux (ou états) d'énergie de cette molécule; on les note E_i ($1 \leq i \leq n$).

[1] Un photon est une 'particule' d'énergie lumineuse.

On représente généralement les niveaux d'énergie sous forme de traits tels que le plus bas corresponde à l'énergie la plus basse et inversement le plus haut à l'énergie la plus élevée. Une molécule dans un certain état d'énergie est représentée par un point sur le trait correspondant (*cfr.* figure I.1).

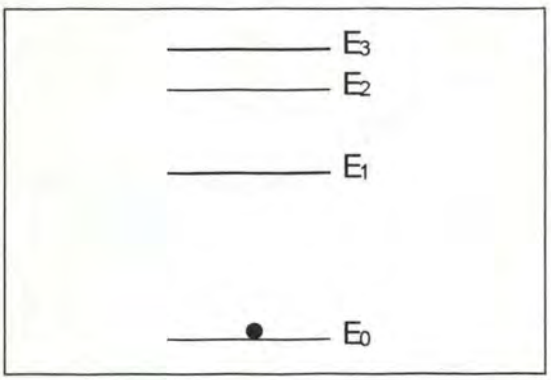


Figure I.1 Les niveaux d'énergie d'une molécule.

Une molécule ne pourra donc exister que si son énergie possède certaines valeurs bien précises qui sont différentes d'une molécule à l'autre. C'est à dire que plusieurs molécules de C_2H_2 possèdent les mêmes niveaux d'énergie permis tandis qu'une molécule de C_2H_2 et une molécule de CO_2 ont des niveaux d'énergie permis différents. Au cours de sa vie, une molécule peut sauter d'un niveau d'énergie à un autre par émission ou absorption d'énergie; ces sauts sont appelés *transitions*. Considérons deux niveaux d'énergie d'une molécule que nous notons E_1 et E_2 ($E_1 < E_2$) et $\Delta E = E_2 - E_1$ la différence d'énergie entre ces deux niveaux. Une transition entre ces deux niveaux ne pourra se produire que pour autant que la quantité d'énergie appropriée ΔE soit émise ou absorbée par la molécule. Dans le cas d'une émission, la molécule saute de l'état E_2 à l'état E_1 puisqu'elle perd de l'énergie; au contraire, dans le cas d'une absorption, elle en gagne et passe de l'état E_1 à l'état E_2.

L'énergie qui est absorbée ou émise par une molécule l'est sous forme d'une radiation électromagnétique de fréquence ν telle que la relation suivante soit vérifiée :

$$\Delta E = h\nu$$

où h est la constante universelle de Planck.

En réalité, une molécule peut également émettre ou absorber de l'énergie à des fréquences voisines de ν comme nous le verrons plus loin dans la section consacrée aux profils d'absorption.

3. La spectroscopie d'absorption

La spectroscopie d'absorption consiste à soumettre un échantillon gazeux à une radiation dont la fréquence ν est ajustable, à faire varier cette fréquence et à observer simultanément l'intensité de la radiation après traversée du gaz. On obtient alors un spectre d'absorption tel que celui de la figure I.2 où on distingue deux régions : une région où il n'y a pas absorption (zones I et III) et une région centrée sur la fréquence ν_0 où le gaz absorbe une partie du rayonnement le traversant (zone II).

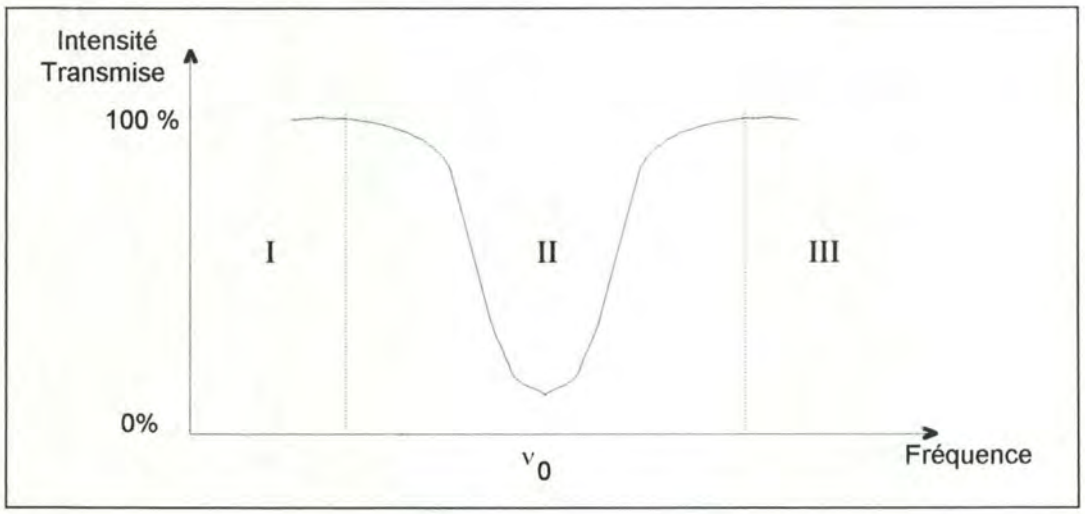


Figure I.2 Un spectre d'absorption.

Examinons ce qui se passe dans chacune des zones du spectre d'absorption et pour cela, considérons une substance gazeuse constituée d'un très grand nombre de molécules identiques dont deux niveaux d'énergie particuliers sont E_1 et E_2 ($E_1 < E_2$); la majorité des molécules étant dans l'état le plus bas E_1. Considérons également un rayonnement électromagnétique de fréquence ν constitué d'un ensemble de photons d'énergie $h\nu$ que l'on dirige sur le mélange gazeux. Il faut envisager deux cas.

$h\nu \neq E_2-E_1$ (Zones I et III)

Etant donné que l'énergie des photons est différente de celle qui sépare les deux niveaux d'énergie E_1 et E_2, aucune transition n'est possible et le mélange gazeux n'est pas perturbé par le passage du rayonnement; c'est à dire que les molécules possèdent la même énergie avant et après son passage. De plus, l'intensité de la radiation à la sortie du mélange est identique à celle à l'entrée puisque aucun photon n'a été absorbé.

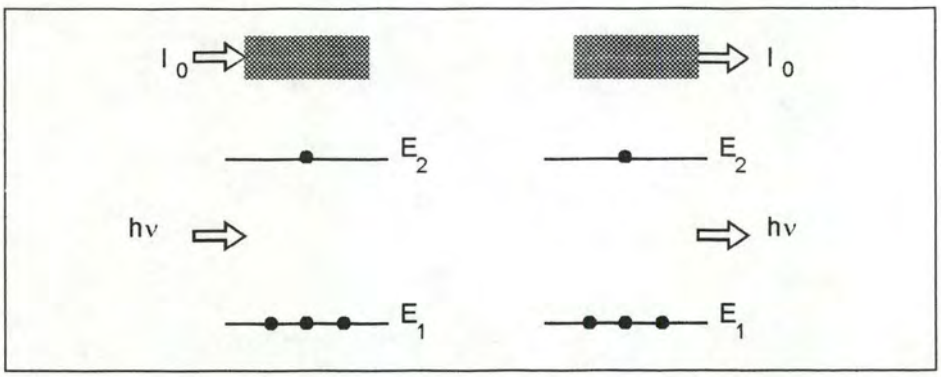


Figure I.3

$h\nu \approx E_2-E_1$ (Zone II)

Des photons sont absorbés par la substance traversée; ce qui se traduit par une excitation des molécules qui sautent dans un état d'énergie plus élevé. L'intensité du faisceau après traversée du gaz étant quant à elle réduite suite à cette absorption. Il faut noter que l'absorption ne se situe pas à une fréquence unique ν_0 telle que $h\nu_0 = E_2-E_1$ mais sur un ensemble de fréquences telles que la raie d'absorption soit centrée sur ν_0.

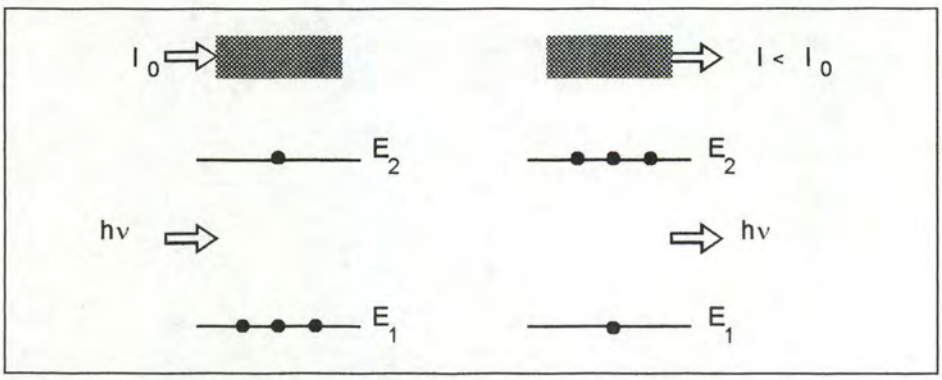


Figure I.4

4. Les profils d'absorption

Lorsqu'une radiation de fréquence σ[1] traverse un milieu absorbant, l'intensité absorbée par celui-ci est proportionnelle à l'élément de parcours δx et à l'intensité incidente $I_0(\sigma)$.

$$-\delta I(\sigma) = K(\sigma)\, I_0(\sigma)\, \delta x$$

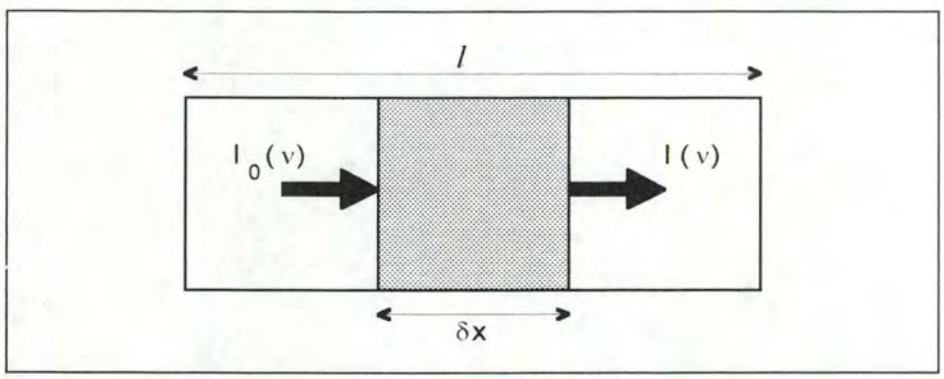


Figure I.5 Absorption à travers un milieu absorbant.

La constante de proportionnalité $K(\sigma)$ est définie comme étant le coefficient d'absorption; si δx s'exprime en *cm*, alors $K(\sigma)$ s'exprimera en cm^{-1}. Pour une épaisseur l du milieu absorbant, il suffit d'intégrer la relation précédente pour obtenir l'intensité $I_l(\sigma)$ transmise à la sortie de l'échantillon.

$$I_l(\sigma) = I_0(\sigma) \exp\left[-\int_0^l K(\sigma)\ dx \right]$$

Dans le cas particulier où le milieu est homogène, c'est à dire lorsque $K(\sigma)$ est indépendant de la position dans l'échantillon, on aura la relation de Beer-Lambert :

$$I_l(\sigma) = I_0(\sigma) \exp\left[-K(\sigma)\, l\right]$$

[1] En spectroscopie moléculaire, on a l'habitude d'exprimer les fréquences non pas en Hz mais en cm^{-1} tel que $\sigma = \nu/c$ où c est la vitesse de la lumière.

A partir de cette relation, on définit l'absorption $A(\sigma)$ et la transmission $T(\sigma)$ d'une radiation de fréquence σ à travers un milieu homogène de la façon suivante :

$$T(\sigma) = \frac{I_l(\sigma)}{I_0(\sigma)} = \exp\left[-K(\sigma)\, l\right]$$

$$A(\sigma) = 1 - T(\sigma) = 1 - \exp\left[-K(\sigma)\, l\right]$$

Le coefficient d'absorption $K(\sigma)$ (et par conséquent la transmission et l'absorption) prend des valeurs non négligeables non pas pour une fréquence σ_0 unique mais sur un intervalle de fréquences plus ou moins grand et on dit alors qu'il possède un profil $\Phi(\sigma)$[1] . C'est en réalité ce profil qui contribue au fait qu'une raie d'absorption ne soit pas infiniment étroite mais qu'elle soit élargie autour d'une fréquence σ_0. Différentes causes physiques sont à l'origine de cet élargissement du profil d'absorption; examinons les.

4.a. Le profil naturel.

En premier lieu, il y a un principe (le principe d'Heisenberg) qui dit qu'un niveau d'énergie d'une molécule n'est pas infiniment étroit mais possède une certaine largeur δE liée au temps de vie du niveau. Par conséquent, quand on dit qu'une molécule est dans un état d'énergie E, cela signifie que son énergie prend sa valeur dans l'intervalle $[E-\delta E/2, E+\delta E/2]$ et pour un ensemble de molécules, leur énergie est distribuée comme sur la figure I.6 où le maximum de la distribution se trouve au centre de la bande des énergies permises.

Le profil d'absorption correspondant à un tel élargissement prend la forme d'une lorentzienne. Cet élargissement est cependant très faible et n'est pas observable au laboratoire vu la résolution du spectromètre utilisé. De plus la largeur de ce profil est tout à fait négligeable vis-à-vis des autres facteurs d'élargissement; c'est pourquoi nous ne nous en préoccuperons pas dans ce travail.

[1] Les profils $\Phi(\sigma)$ présentés dans ce travail sont à un coefficient de normalisation près les coefficients d'absorption $K(\sigma)$ mesurés.

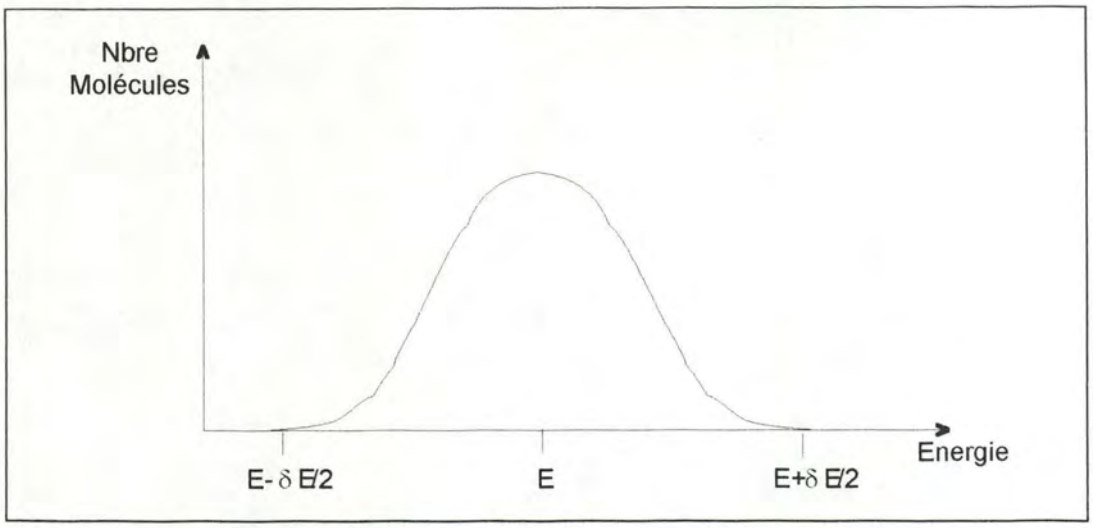


Figure I.6 Distribution des molécules en fonction de l'énergie

4.b. Le profil Doppler.

Sous l'agitation thermique, toutes les molécules sont animées d'une vitesse propre et chaque molécule voit alors la radiation incidente non pas avec sa fréquence d'émission σ mais avec une fréquence légèrement différente $\sigma \pm \Delta\sigma$ (= effet Doppler). La variation de fréquence $\Delta\sigma$ est liée à la vitesse d'une molécule et par conséquent chaque molécule a un $\Delta\sigma$ qui lui est associé. Le profil d'absorption $\Phi^D(\sigma)$ dû à cet effet Doppler a la forme d'une gaussienne :

$$\Phi^D(\sigma) = \frac{1}{\gamma^D}\sqrt{\frac{\ln 2}{\pi}}\exp\left[-\ln 2\left[\frac{\sigma-\sigma_0}{\gamma^D}\right]^2\right]$$

où σ_0 est le centre du profil (cm^{-1}).

γ^D est la demi-largeur à mi-hauteur du profil (cm^{-1}).

La demi-largeur à mi-hauteur du profil d'absorption est donnée par la relation suivante :

$$\gamma^D = 3.58\ 10^{-7}\sqrt{\frac{T}{M}}\ \sigma_0$$

où M est la masse moléculaire (*u.m.a.*).

T est la température (*Kelvin*).

4.c. Le profil collisionnel.

Les mesures de spectres d'absorption ne sont pas effectuées sur des molécules isolées mais bien sur des ensembles de molécules qui vont inévitablement interagir entre elles (notamment par collisions). Ces interactions entre les molécules d'un échantillon gazeux vont légèrement modifier la position de leurs niveaux d'énergie; par conséquent lors de l'absorption de radiations, toutes les molécules ne vont pas absorber à la même fréquence. Le profil $\Phi^L(\sigma)$ d'une raie d'absorption où l'élargissement est causé de façon prépondérante par des collisions moléculaires sera une lorentzienne :

$$\Phi^L(\sigma) = \frac{1}{\pi} \frac{\gamma^L}{(\sigma - \sigma_0)^2 + (\gamma^L)^2}$$

où γ^L est la demi-largeur collisionnelle à mi-hauteur du profil (cm^{-1}).

$$\gamma^L = \sum_i \gamma_i^0 P_i$$

où γ_i^0 est le coefficient d'élargissement par le gaz i du mélange (cm^{-1}/atm).

P_i est la pression partielle du gaz (atm).

4.d. Le profil de Voigt.

Le profil d'une raie d'absorption est fortement lié à la pression du gaz étudié. En effet à basse pression, où les collisions moléculaires sont relativement peu nombreuses du fait du peu de molécules présentes, le profil sera un profil de type Doppler; tandis qu'à haute pression, le profil sera collisionnel. Entre ces deux types extrêmes de profils, le coefficient d'absorption est une superposition des deux contributions Doppler et collisionnelle. Le profil du coefficient d'absorption s'exprime alors par une courbe de Voigt qui est en fait un produit de convolution[1] d'une gaussienne et d'une lorentzienne; sa forme est donnée par l'expression suivante :

[1] Le produit de convolution de deux fonctions g et h que l'on note $g * h$ est défini par :

$$g * h = \int_{-\infty}^{+\infty} g(\tau)\, h(t-\tau)\, d\tau$$

$$\Phi^{V}(\sigma) = \frac{1}{\gamma^{D}} \sqrt{\frac{\ln 2}{\pi}}\ k(x,y)$$

$$x = \sqrt{\ln 2}\ \frac{\sigma - \sigma_0}{\gamma^{D}}$$

$$y = \sqrt{\ln 2}\ \frac{\gamma^{L}}{\gamma^{D}}$$

$$k(x, y) = \frac{y}{\pi} \int_{-\infty}^{+\infty} \frac{\exp[-t^2]}{y^2 + (x - t)^2}\, dt$$

L'intégrale présente dans l'expression de $k(x,y)$ n'est malheureusement pas analytique et doit être évaluée numériquement ou approchée par une fonction. On trouve plusieurs méthodes d'évaluation de cette fonction de Voigt dans la littérature; voir notamment à ce sujet les références [8,9,10,11]. Dans ce travail, nous nous sommes basés sur la technique d'évaluation développée par A.K.Hui, B.H. Armstrong et A.A. Wray (Ref. [8]).

4.e. Les profils de Voigt généralisé et de Galatry.

Les profils d'absorption mesurés à haute résolution sont généralement modélisés par des profils de Voigt qui ont des comportements limites corrects à la fois à haute et à basse pression; en effet, ils tendent vers des profils collisionnels à haute pression et vers des profils Doppler à basse pression. Cependant, avec l'évolution des techniques d'enregistrement des spectres d'absorption moléculaire (notamment les spectromètres à diode laser), les mesures sont devenues de plus en plus précises et fines et des désaccords systématiques très faibles ont pu être observés entre les résultats expérimentaux et le modèle du profil de Voigt. Ces différences systématiques ont pu être gommées en utilisant des modèles plus raffinés tels que les profils de Voigt généralisé ou les profils de Galatry. Ces profils tiennent compte du fait que lorsqu'il y a des collisions moléculaires, on observe un rétrécissement de l'élargissement Doppler du coefficient d'absorption. Ce phénomène est présent à haute pression lorsque les molécules effectuent un très

grand nombre de collisions sur un faible intervalle de temps; ce qui leur confère une vitesse moyenne relativement faible et même parfois nulle.

Le profil de Voigt généralisé qui tient compte de cet effet ainsi que de la présence d'un gaz perturbateur dans le mélange étudié est donné par :

$$\Phi^{VG}(a_0, \Delta_0, \lambda, q, x) = \frac{A}{\sqrt{\pi^3}} \int_{-\infty}^{+\infty} \exp[-\frac{u^2}{\lambda}] \frac{[-(u-X)\,h'(u) + h(u)]}{[u - X + D\,h(u)]^2 + [A\,h(u)]^2}\, du$$

$$A = a_0 \sqrt{\lambda (1+\lambda)^{-p}}$$

$$X = x \sqrt{\lambda}$$

$$D = \Delta_0 \sqrt{\lambda (1+\lambda)^{-p}}$$

$$p = \frac{(q-3)}{(q-1)}$$

$$h(u) = M(-\frac{p}{2}, \frac{3}{2}, -u^2)$$

$$\lambda = \frac{m_p}{m_a}$$

$$a_0 = \frac{\gamma^L}{\gamma^D}$$

$$M(a,b,z) = 1 + \frac{a}{b}\frac{z}{1!} + \frac{a(a+1)}{b(b+1)}\frac{z^2}{2!} + \frac{a(a+1)(a+2)}{b(b+1)(b+2)}\frac{z^3}{3!} + \ldots$$

où γ^L est la demi-largeur collisionnelle à mi-hauteur du profil.

γ^D est la demi-largeur Doppler à mi-hauteur du profil.

x est la distance par rapport au centre de la raie.

m_p est la masse du gaz perturbateur.

m_a est la masse du gaz absorbant.

Δ_0 est le déplacement du centre de la raie dû aux effets collisionnels.

q est une mesure de la force du potentiel interatomique.

$M(a, b, z)$ est une fonction hypergéométrique.

$h'(u)$ est la dérivée de $h(u)$.

Ces profils de Voigt généralisés permettent entre autres de représenter des profils d'absorption asymétriques et de rendre compte des déplacements éventuels des centres des raies sous l'influence de perturbateurs. Remarquons que lorsque q=3, il y a plusieurs termes qui se simplifient dans la définition du profil de Voigt généralisé et celui-ci se ramène à un profil de Voigt classique.

Le profil de Galatry est quant à lui donné par l'expression suivante :

$$\Phi^{Gal} = \frac{1}{\gamma^D}\sqrt{\frac{\ln 2}{\pi}}\,G(x, y, z)$$

Avec

$$G(x, y, z) = \frac{1}{\sqrt{\pi}} * \mathrm{Re}\left\{ \int_0^{+\infty} \exp\left[-ixt - yt + \frac{1}{2z^2}\left(1 - zt - \exp(-zt)\right)\right] dt\right\}$$

$$z = \frac{\beta\sqrt{\ln 2}}{2\pi\gamma^D}$$

$$\beta = \frac{kT}{mD}$$

où T est la température du gaz.

β est la fréquence effective des collisions.

D est le coefficient de diffusion.
x et y ont les mêmes expressions que pour le profil de Voigt.

Notons que dans le cas particulier où la variable z prend une valeur nulle, le profil de Galatry se ramène à un profil de Voigt classique. Pour les profils de Voigt généralisé et de Galatry, nous utilisons respectivement les techniques d'approximation proposées par D.Cope, R. Khoury et R.J. Lovett (Ref. [6]) et P.L. Varghese et R.K. Hanson (Ref. [12]).

5. Les grandeurs caractéristiques des profils d'absorption

Les grandeurs caractéristiques communes à chacun des modèles de profils sont : la position du centre du profil (σ_0), la demi-largeur à mi-hauteur de ce profil (γ) et l'intensité du profil en son centre ($I(\sigma_0)$); ces quantités sont représentées sur la figure I.7.

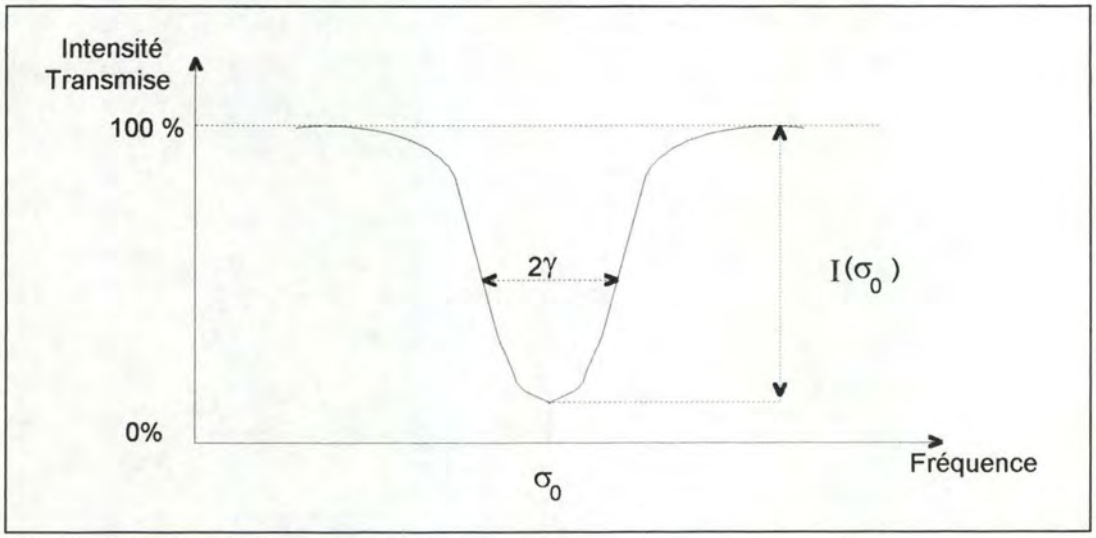


Figure I.7 Les grandeurs caractéristiques d'un profil.

Chapitre II

La modélisation des données

1. Le problème de la modélisation de données

Etant donné un ensemble d'observations, on désire souvent les analyser et les interpréter en adaptant sur celles-ci une équation contenant un certain nombre de paramètres ajustables. On peut distinguer deux approches de ce problème en fonction de l'utilité et de l'interprétation des résultats de l'ajustement de l'équation. Ces deux approches sont respectivement l'ajustement de courbes et l'ajustement de modèles.

1.a. L'ajustement de courbes.

D'un point de vue pratique, il est commode de pouvoir condenser ou résumer un ensemble de données sous une forme compacte. Il arrive en effet bien souvent que l'on emmagasine un grand nombre de données et afin d'optimiser leur stockage ainsi que leur manipulation, on a tout intérêt à les représenter sous forme d'une fonction analytique simple ne dépendant que de quelques paramètres; cette fonction correspond en fait à une courbe passant aussi près que possible de chacune des observations. C'est ce qu'on appelle la *réduction des données* dans le sens de réduire la quantité de données sans pour autant réduire la quantité d'information que ces données contenaient. La représentation d'un ensemble de données numériques sous forme d'une équation permet aussi l'interpolation pour des valeurs spécifiques des variables indépendantes de l'équation; valeurs pour lesquelles des observations n'étaient pas disponibles mais pour lesquelles il serait intéressant de posséder des résultats.

Dans ce genre de situation, la forme des courbes que l'on ajuste est quelque peu arbitraire et n'est pas dérivée des 'lois de la nature'; par conséquent, les valeurs des paramètres ajustables de la courbe n'ont pas de signification physique précise et les incertitudes qui leur sont associées peuvent ne pas avoir grand intérêt. Il est toutefois recommandé d'examiner les différences entre les observations et les valeurs de la courbe ajustée en chacun des points. Si ces différences sont distribuées

de manière aléatoire autour de zéro, il est fort probable que le choix de la courbe ajustée soit approprié et les différences observées correspondent alors à du bruit sur les observations. Par contre si on rencontre des effets systématiques tels que des différences fortement négatives sur le versant droit d'un pic et des différences fortement positives sur son versant gauche, c'est que la courbe n'est pas adéquate et il faut en chercher une autre. Les figures II.1 et II.2 montrent deux cas où les fonctions ajustées sont respectivement adéquate et non appropriée.

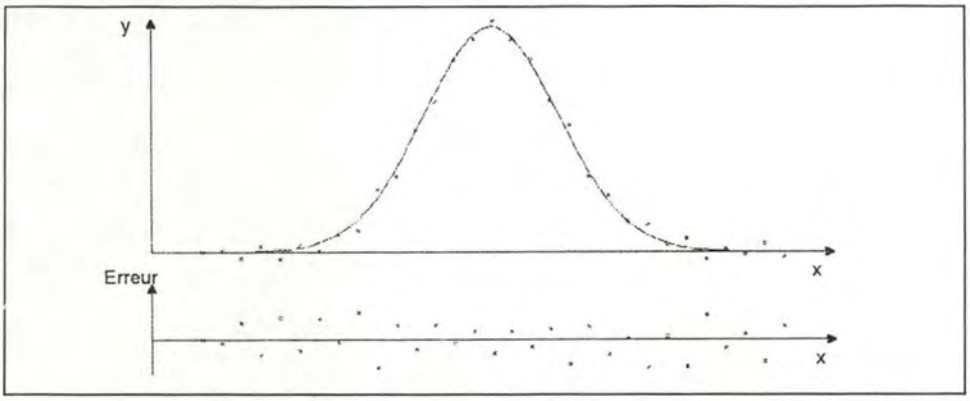


Figure II.1 Un ajustement approprié.

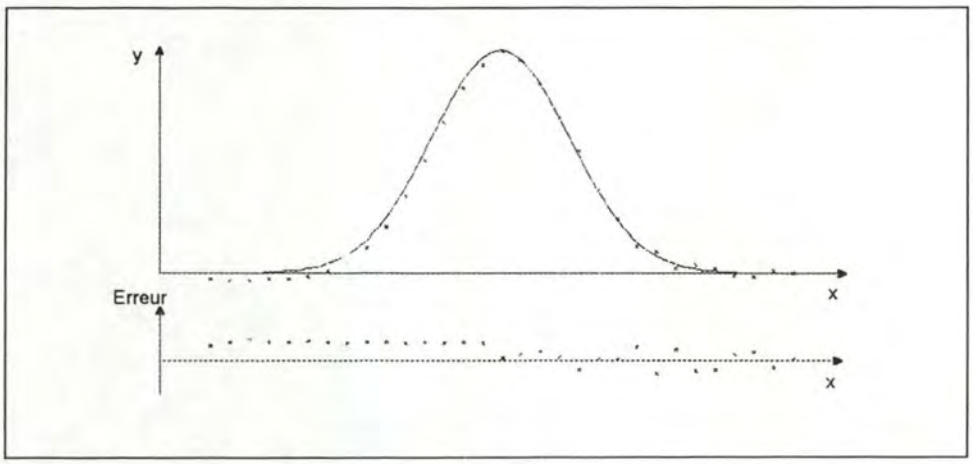


Figure II.2 Un ajustement non approprié.

1.b. L'ajustement de modèles.

La seconde approche concerne des ajustements d'équations sur des observations pour lesquelles il existe un phénomène physique sous-jacent au processus qui les génère. En particulier, en spectroscopie moléculaire, les paramètres ajustables de l'équation correspondent à des constantes moléculaires

telles que la position des niveaux d'énergie, les constantes rotationnelles et vibrationnelles, ... Dans ce genre de situations, les équations sont des représentations algébriques de modèles physiques incluant ces constantes comme paramètres ajustables. Le but de l'ajustement est d'identifier les valeurs numériques spécifiques de ces paramètres puisque ceux-ci ont désormais une signification physique. Il faut cependant noter que ces valeurs ne sont que des estimations des valeurs réelles et que les incertitudes estimées sur ces valeurs représentent une mesure de la probabilité que les paramètres prennent effectivement ces valeurs.

Il est vital que le modèle utilisé pour l'ajustement soit correct; dans le cas contraire, les valeurs numériques déterminées pour les paramètres n'auraient aucune signification physique quelles que soient les différences entre les observations et les valeurs calculées sur base du modèle. Il serait en effet possible qu'une fonction passe merveilleusement bien à travers l'ensemble des observations mais que le modèle associé à celle-ci ne corresponde pas du tout au phénomène physique.

Selon cette approche, la modélisation des données est d'une utilité considérable aussi bien d'un point de vue pratique que théorique. En spectroscopie moléculaire, la connaissance des valeurs exactes des paramètres d'un modèle moléculaire va entre autres permettre d'élargir l'ensemble des données spectroscopiques. Par exemple, après avoir ajusté les paramètres du modèle sur les observations, nous serons capables d'interpoler pour trouver des données manquantes. Ayant réussi à déterminer les paramètres d'un modèle particulier, ces valeurs pourront également être injectées dans d'autres modèles où elles deviendront des constantes.

1.c. Les méthodes d'ajustement.

Il existe une grande variété de techniques d'ajustement d'équations sur des données; celles-ci s'étendent des méthodes graphiques simples et manuelles jusqu'aux méthodes numériques nécessitant des calculs assez complexes. En spectroscopie, avant l'apparition des ordinateurs dans tous les laboratoires, les ajustements étaient effectués à la main. Cette façon de faire avait le désavantage de ne pas être d'une très grande précision mais par contre elle offrait l'avantage d'être subjective en ce sens que le spectroscopiste pouvait utiliser son intuition; ce que ne peut pas faire la machine. Actuellement, il serait inconcevable d'encore travailler de la sorte et ces techniques sont remplacées par des méthodes numériques beaucoup plus rapides et performantes. Il ne faut toutefois pas automatiser ces techniques à 100% mais plutôt laisser une certaine marge de liberté à l'utilisateur qui peut forcer

la recherche dans certaines directions en fonction de son expérience et de son intuition.

Dans les techniques numériques d'ajustement de modèle, l'approche de base est la suivante : on construit une fonction, appelée *fonction de mérite*, qui mesure l'accord entre les données et le modèle pour un choix particulier de ses paramètres ajustables. On s'arrange généralement pour que des valeurs faibles de cette fonction correspondent à un bon accord et les paramètres du modèle sont alors ajustés pour obtenir un minimum de cette fonction de mérite. Le problème de l'ajustement de modèle se ramène donc à un problème de minimisation dans un espace à plusieurs dimensions; une dimension par paramètre à ajuster.

Les problèmes de la modélisation de données correspondant à des observations expérimentales possèdent certaines caractéristiques importantes qui leur sont propres. Les données ne sont généralement pas exactes parce qu'elles sont sujettes aux inévitables erreurs de mesures que l'on appelle le bruit; par conséquent il est impossible de déterminer un ensemble de paramètres tels que le modèle ajusté colle exactement sur les données même si le modèle utilisé est correct. Dans ce cas, on doit donc prévoir un moyen de déterminer si le modèle est approprié et pour cela il faut pouvoir mesurer l'exactitude de l'ajustement. On doit également prévoir une mesure de l'erreur sur chacun des paramètres déterminés par l'ajustement. Une autre caractéristique de ces problèmes est qu'il peut exister plusieurs minima de la fonction de mérite et le minimum atteint lors de l'ajustement n'est peut être pas le minimum global recherché mais un minimum local; dans certains cas celui-ci est suffisant mais dans d'autres pas.

2. Idée intuitive de l'ajustement de modèles non linéaires

Si nous envisageons l'ensemble des valeurs prises par la fonction de mérite pour toutes les valeurs possibles des paramètres ajustables du modèle, nous obtenons une surface dans un espace à n dimensions; n étant le nombre de paramètres que l'on a à ajuster. Cette surface peut être comparée à un paysage montagneux dans lequel les minima de la fonction de mérite sont représentés par des vallées tandis que les maxima coïncident avec des crêtes (figure II.3). Le problème que nous voulons résoudre est de minimiser la fonction de mérite; ce qui est équivalent à trouver le point le plus bas dans la vallée la plus basse de notre surface.

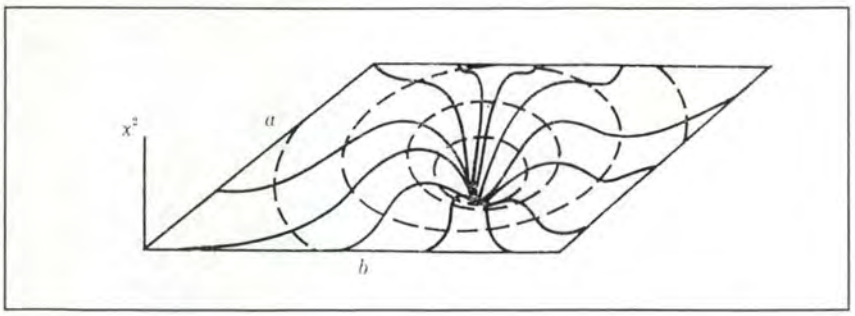


Figure II.3 Exemple de variation du χ^2 en fonction de 2 paramètres a et b.

L'algorithme de recherche de l'ajustement optimal pourrait donc se résumer de la manière suivante : partant d'un point quelconque, il faut descendre dans la vallée jusqu'à ce qu'on ne puisse plus aller plus bas. La solution n'est cependant pas si simple et le premier problème qui surgit est de savoir depuis où commencer la recherche. Le choix du point de départ est en effet crucial et il doit impérativement se trouver dans la vallée qui contient le minimum global de la surface associée à la fonction de mérite faute de quoi l'algorithme sera bien capable de trouver un minimum mais ce sera un minimum local. C'est à l'utilisateur que revient la lourde tâche de choisir ce point de départ en donnant une valeur approchée pour chacun des paramètres; ensuite la recherche peut se faire de manière autonome.

La recherche du minimum global est une recherche itérative qui, à chaque étape, produit une solution qui confère à la fonction de mérite une valeur plus faible. Tout l'art de la recherche étant de savoir dans quelle direction se déplacer et de quelle distance. Si l'algorithme effectue de trop grands sauts, il est probable qu'il se retrouve dans une vallée voisine; par contre, si les pas sont trop petits la recherche risque de prendre un temps considérable. En ce qui concerne la direction dans laquelle s'effectue la recherche, il suffit de calculer le gradient de la fonction de mérite par rapport aux paramètres ajustables pour avoir la direction optimale puisque le gradient indique la direction selon laquelle la fonction change le plus rapidement.

3. Description mathématique du problème des moindres carrés non linéaires

Soit un ensemble d'observations que nous représentons sous la forme de couples de points (x_i, y_i) $(1 \leq i \leq n)$; ces observations possédant une déviation standard σ. Nous désirons adapter sur ces données un modèle comportant M paramètres ajustables a_j $(1 \leq j \leq M)$. Ce modèle prévoit une relation fonctionnelle entre les variables indépendantes x_i et les variables dépendantes y_i mesurées :

$$y_i = f(x_i; \mathbf{a})$$

$$\mathbf{a} = a_1, a_2, ..., a_M$$

Nous désirons estimer les valeurs des paramètres ajustables a_j $(1 \leq j \leq M)$ de telle sorte que la quantité suivante soit minimisée :

$$\chi^2 = \sum_{i=1}^{n} \frac{(f(x_i) - y_i)^2}{\sigma^2}$$

Cette minimisation fournit des valeurs aux paramètres ajustables de telle sorte que l'ajustement soit optimal au sens des moindres carrés.

Dans le cas particulier où le modèle à adapter (donc la fonction f) est linéaire par rapport aux paramètres ajustables, la solution est obtenue en résolvant le système linéaire suivant par rapport aux paramètres a_j $(1 \leq j \leq M)$:

$$\frac{\partial \chi^2}{\partial a_j} = 0 \qquad j = 1, 2, ..., M$$

Lorsque la fonction à adapter est telle qu'une forme analytique de ses dérivées partielles n'est pas disponible, cette façon de faire ne convient pas et on utilise alors d'autres techniques de recherche du minimum d'une fonction. C'est notamment le cas pour des modèles non-linéaires dont l'expression comporte des intégrales.

4. Méthodes de recherche du minimum d'une fonction

Dans ce travail, nous utilisons l'algorithme de Levenberg-Marquardt, lequel sera développé plus loin, pour la recherche de la solution optimale des paramètres à ajuster sur les données; cet algorithme utilise deux méthodes de recherche du minimum que nous décrivons ci-dessous. L'algorithme de Levenberg-Marquardt effectue une recherche itérative de la valeur minimale du χ^2 : étant donné un ensemble de valeurs comme solution initiale des paramètres du modèle, l'algorithme améliore cette solution à chaque itération; la procédure étant répétée jusqu'à ce que la quantité χ^2 ne puisse plus diminuer.

4.a. La méthode de la matrice Hessienne.

Développons la quantité χ^2 sous forme d'une série de Taylor :

$$\chi^2(x; \mathbf{a}_0 + \delta\mathbf{a}) = \chi^2(x; \mathbf{a}_0) + \sum_j \frac{\partial \chi^2}{\partial a_j} \delta a_j + \frac{1}{2} \sum_{j,k} \frac{\partial^2 \chi^2}{\partial a_j \partial a_k} \delta a_j \delta a_k + \ldots$$

Si nous posons :

$$c \equiv \chi^2(x; \mathbf{a}_0)$$

$$\mathbf{b} \equiv -\nabla \chi^2 \big|_{x, \mathbf{a}_0}$$

$$[\mathbf{H}]_{j,k} \equiv \frac{\partial^2 \chi^2}{\partial a_j \partial a_k} \big|_{x, \mathbf{a}_0}$$

et que nous négligeons les termes supérieurs aux termes quadratiques, nous pouvons réécrire l'équation ci-dessus sous la forme matricielle suivante :

$$\chi^2(x; \mathbf{a}_0 + \delta\mathbf{a}) = c - \mathbf{b} \cdot \delta\mathbf{a} + \frac{1}{2} \delta\mathbf{a} \cdot \mathbf{H} \cdot \delta\mathbf{a}$$

où la matrice **H**, dont les composantes sont les dérivées secondes partielles de χ^2 par rapport aux paramètres ajustables a_j, porte le nom de *matrice Hessienne*.

Si nous prenons le gradient de l'équation précédente, nous obtenons la relation suivante :

$$\nabla \chi^2 |_{x;\mathbf{a}_0+\delta\mathbf{a}} = \mathbf{H} \,.\, \delta\mathbf{a} - \mathbf{b}$$

Le minimum de la quantité χ^2 est obtenu lorsque sa dérivée par rapport à chacun des paramètres est nulle; c'est à dire lorsque son gradient s'annule. On trouve donc ce minimum en résolvant l'équation suivante :

$$\mathbf{H} \,.\, \delta\mathbf{a} = \mathbf{b}$$

En se rappelant la définition de **b** et en posant $\delta\mathbf{a} = \mathbf{a}_{min} - \mathbf{a}_0$, on peut sauter à la solution optimale en résolvant l'équation :

$$\mathbf{a}_{min} = \mathbf{a}_0 + \mathbf{H}^{-1}.\left[-\nabla \chi^2(x; \mathbf{a}_0) \right]$$

Cette méthode est appelée la méthode de la matrice Hessienne et pour pouvoir être appliquée, il est nécessaire que la fonction de mérite puisse être développée en série de Taylor. Cela signifie que cette méthode est uniquement applicable dans le voisinage de la solution qui minimise le χ^2. Elle offre cependant l'avantage de se rendre à la solution optimale en un seul saut.

4.b. La méthode du gradient.

Si on n'est pas suffisamment près du minimum, le développement de la fonction χ^2 sous forme d'une série de Taylor n'est plus applicable et il faut donc trouver une autre méthode. Dans cette méthode de recherche du minimum du χ^2, tous les paramètres ajustables a_j $(1 \leq j \leq M)$ sont incrémentés simultanément d'un accroissement δa_j tel que la direction de déplacement soit celle du gradient du χ^2. Le choix de cette direction est motivé par le fait que le gradient d'une fonction scalaire mesure la direction de variation maximale de cette fonction; c'est donc la direction qui nous amènera le plus rapidement vers le minimum. Le signe négatif apparaissant

devant le gradient dans l'équation précédente indique que le sens de déplacement est opposé à celui du gradient.

Le principe de cette méthode est simple : partant d'une position quelconque, nous calculons le gradient du χ^2 en ce point et nous nous déplaçons d'une distance arbitraire dans cette direction. Cette distance est choisie de telle sorte que la nouvelle position atteinte donne une valeur du χ^2 plus faible. On répète la même procédure jusqu'à ce qu'on ne puisse plus diminuer le χ^2.

La position du nouveau point atteint lors de chaque itération est donnée par la relation suivante :

$$\mathbf{a}_{new} = \mathbf{a}_{initial} - cste \cdot \left[-\nabla \chi^2(x; \mathbf{a}_{initial}) \right]$$

Où la constante mesure l'amplitude du déplacement.

La méthode du gradient est parfaitement adaptée aux situations où la recherche se trouve assez loin de la solution optimale; par contre elle converge très lentement lorsque la recherche est proche du minimum. Cette méthode convient donc très bien pour effectuer une approche de la solution.

Quelle que soit la méthode utilisée, nous avons besoin de calculer le gradient de χ^2 qui a pour composantes :

$$b_j = -2 \sum_{i=1}^{n} \frac{\left[y_i - f(x_i; \mathbf{a}) \right]}{\sigma^2} \frac{\partial f(x_i; \mathbf{a})}{\partial a_j}$$

Si en plus nous avons besoin de la matrice Hessienne, nous devons être capables de calculer les éléments suivants :

$$H_{j,k} \equiv 2 \sum_{i=1}^{n} \frac{1}{\sigma^2} \left(\frac{\partial f(x_i; \mathbf{a})}{\partial a_j} \frac{\partial f(x_i; \mathbf{a})}{\partial a_k} - \left[y_i - f(x_i; \mathbf{a}) \right] \frac{\partial^2 f(x_i; \mathbf{a})}{\partial a_j \partial a_k} \right)$$

Lorsque nous désirons ajuster une fonction simple sur un ensemble d'observations, nous en connaissons généralement sa forme analytique et par conséquent nous sommes à même de calculer ses dérivées partielles premières et secondes; ce qui nous permet d'évaluer le gradient et la matrice Hessienne de la fonction de mérite en tout point. Lorsque les fonctions deviennent plus compliquées telles que celles que nous devons modéliser dans ce travail (profils de Voigt, Voigt généralisé, ...), ces évaluations ne sont plus possibles pour la simple raison que, bien que connaissant leur forme analytique, on n'est pas capable d'obtenir une représentation analytique de leurs dérivées partielles. A titre d'exemple, l'intégrale intervenant dans le profil de Voigt n'étant pas analytique, nous sommes incapables d'en calculer les dérivées partielles par rapport aux paramètres ajustables.

Les éléments du gradient et de la matrice Hessienne doivent dans ces cas là être construits de manière itérative de telle sorte qu'à chaque itération nous en obtenions une bonne approximation. Au fur et à mesure de ses itérations, l'algorithme construit donc des matrices **b** et **H** sur base d'informations telles que l'évaluation de la fonction en un point, la direction de minimisation de la fonction,... En particulier, l'algorithme construit une séquence de matrices $\mathbf{H}^i$ telle que pour un nombre d'itérations infini la relation suivante soit vérifiée :

$$\lim_{i \to \infty} \mathbf{H}^i = \mathbf{H}$$

5. L'algorithme de Levenberg-Marquardt

Marquardt a suggéré une méthode élégante, basée sur une suggestion précédente de Levenberg, qui combine les avantages des deux méthodes de recherche de la solution optimale que nous avons décrites précédemment. Loin de la solution optimale cet algorithme utilise la méthode du gradient; par contre lorsque la recherche aboutit suffisamment près de cette solution optimale, on peut développer le χ^2 en série de Taylor, et c'est la méthode de la matrice Hessienne qui est utilisée. Cet algorithme fonctionne très bien et est actuellement devenu un standard dans les routines de moindres carrés non linéaires.

Comme nous l'avons déjà mentionné précédemment, la méthode du gradient ne donne aucune information quant à la valeur que doit prendre la constante dans la relation

$$\mathbf{a}_{\mathbf{new}} = \mathbf{a}_{\mathbf{initial}} - \text{cste}.\left[-\nabla \chi^2(\mathrm{x}; \mathbf{a}_{\mathbf{initial}})\right]$$

que l'on peut encore écrire sous la forme

$$a_{i,new} = a_{i,initial} - cste_i \times b_i \qquad i = 1, 2, \ldots, M$$

Marquardt a enrichi cette méthode en donnant un moyen d'évaluer cette constante. Si on examine l'équation aux dimensions de l'équation précédente, il ressort que la constante doit avoir la dimension de a_i au carré puisque b_i a une dimension inverse de celle de a_i. Il faut noter que chaque composante de $\boldsymbol{a}$, c'est-à-dire chaque paramètre, a sa propre dimension qui peut être différente des autres et c'est généralement le cas. Or nous disposons d'éléments possédant cette dimension; il s'agit des éléments inverses des éléments diagonaux de la matrice Hessienne (et uniquement ceux-là). Par conséquent nous pouvons les prendre comme valeur des constantes $cste_i$; on leur affecte cependant un facteur multiplicatif λ (sans dimension) car ils peuvent ne pas avoir la bonne échelle.

Avec ces considérations, la relation suivante remplace l'équation précédente et elle permet de déterminer les accroissements des paramètres à chaque étape.

$$\delta a_j = \frac{b_j}{\lambda H_{jj}} \qquad j = 1, 2, ..., M$$

Il est dés lors très simple de combiner les méthodes de la matrice Hessienne et du gradient en définissant une nouvelle matrice **H'** telle que :

$$H'_{jj} = H_{jj}\,(1+\lambda)$$

$$H'_{kj} = H_{kj} \qquad (k \neq j)$$

Les accroissements des paramètres sont alors déterminés en résolvant les équations

$$\sum_{j=1}^{n} H'_{kj}\,\delta a_j = b_k \qquad k = 1, 2, ..., M$$

Quand λ est très grand, les termes diagonaux de la matrice **H'** sont dominants dans la somme de l'équation précédente et c'est la méthode du gradient qui est utilisée. On retrouve en effet la relation

$$H'_{kk}\,\delta a_k = b_k$$

Par contre, au fur et à mesure que λ approche de zéro, on retrouve la méthode de la matrice Hessienne où l'on doit résoudre l'équation

$$\mathbf{H}\,.\,\delta\mathbf{a} = \mathbf{b}$$

Voici à présent l'algorithme que Marquardt a développé sur ces idées afin de trouver l'ensemble des valeurs optimales des paramètres ajustables.

1. Choisir un point de départ arbitraire $\mathbf{a}_0$

2. Calculer la fonction de mérite en ce point χ^2 ($\mathbf{a}_0$)

3. Prendre une valeur faible pour λ; par exemple 10^{-3}

4. Evaluer la matrice **H** et donc **H'** [1]

5. Résoudre l'équation suivante par rapport à $\delta\mathbf{a}$ et évaluer $\chi^2(\mathbf{a}_0+\delta\mathbf{a})$

$$\sum_{j=1}^{n} H'_{kj}\delta a_j = b_k \qquad k = 1, 2, \ldots, M$$

6. Si $\chi^2(\mathbf{a}_0+\delta\mathbf{a}) > \chi^2(\mathbf{a}_0)$ alors multiplier λ par 10 et retourner en 4

7. Si $\chi^2(\mathbf{a}_0+\delta\mathbf{a}) < \chi^2(\mathbf{a}_0)$ alors diviser λ par 10, remplacer $\mathbf{a}_0$ par $\mathbf{a}_0+\delta\mathbf{a}$ et retourner en 4

[1] L'évaluation de la matrice Hessienne est effectuée dans notre cas par la routine IMSL de façon à assurer la convergence des matrices $\mathbf{H}^{\mathbf{i}}$.

Chapitre III

Le traitement des spectres d'absorption infrarouge

1. Introduction

Les spectres d'absorption enregistrés au laboratoire de spectroscopie moléculaire ne sont pas utilisables tels quels. Afin de pouvoir les interpréter et d'en retirer de l'information "utile", il est nécessaire de leur faire subir quelques traitements préalables. Dans le cas particulier qui nous préoccupe, à savoir l'ajustement de modèles sur des profils d'absorption, la chaîne des traitements comprend les étapes suivantes :

- Acquisition des spectres.
- Filtrage des spectres.
- "Ajustement" du fond.
- Détection des raies des spectres.
- Linéarisation des spectres.

Nous allons expliciter chacune de ces étapes qui une fois exécutées nous fourniront un spectre sur lequel il sera désormais possible de travailler.

2. L'acquisition des spectres (programmes **AVERI** et **SEPFI** sur HP9000)

Nous ne décrivons pas ici la technique d'acquisition des spectres utilisée au laboratoire de spectroscopie moléculaire mais plutôt le résultat qu'elle va nous fournir; le lecteur qui serait intéressé par des explications sur cette technique peut toujours consulter la référence [13]. Le résultat d'une acquisition est un fichier contenant quatre spectres d'absorption digitalisés ainsi que des informations sur les conditions expérimentales d'enregistrement; ces quatre spectres sont :

1. **Le continuum** (ou **fond**) : Enregistrement du signal laser en l'absence de gaz dans la cellule d'absorption; ce spectre

permet de fixer le niveau de 100 % de transmission (ou 0 % d'absorption).

2. **L'étalon** : Enregistrement des franges de l'étalon de Fabry-Perot qui permet une calibration relative des spectres en fréquence.

3. **Le spectre Doppler** : Enregistrement du spectre d'absorption pour une faible pression du gaz dans la cellule afin de se situer dans un régime où on peut considérer que le profil d'absorption est un profil Doppler (*cfr.* chapitre I).

4. **Le spectre** : Enregistrement du spectre d'absorption proprement dit avec le gaz étudié dans la cellule. La pression du gaz ainsi que la température de la cellule étant enregistrés dans l'ensemble des informations qui sont stockées dans le fichier.

La figure suivante donne un exemple de chacun de ces quatre spectres.

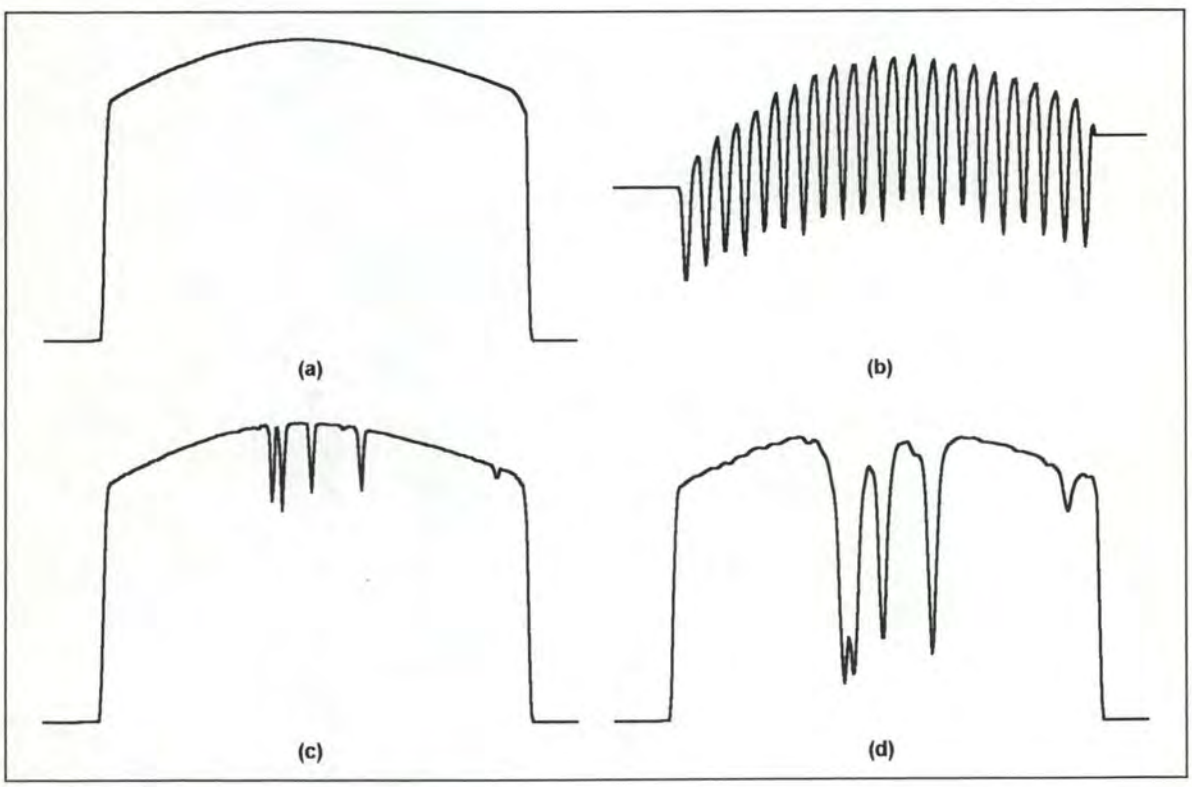


Figure III.1 Les 4 spectres d'absorption enregistrés : (a) continuum, (b) étalon, (c) Spectre Doppler, (d) Spectre.

Ces quatre enregistrements sont stockés dans le fichier résultat sous forme de blocs de 1024 (ou 2048 ou encore 4096 selon le type de spectre enregistré) points

chacun où la valeur en ces points représente une mesure relative de la transmission en fonction de la fréquence.

3. Le filtrage des spectres (Programme **AVECI** sur HP9000)

En sciences expérimentales, les mesures obtenues sont presque toujours entachées d'un bruit du fond et les spectres d'absorption enregistrés au laboratoire n'échappent malheureusement pas à cette règle. Ce bruit inévitable résulte de phénomènes aussi bien optiques (réflexions parasites, ...) qu'électroniques (dans la chaîne des composants allant du détecteur à l'ordinateur effectuant les acquisitions). Il doit être réduit ou éliminé avant de pouvoir utiliser efficacement les mesures effectuées. A titre d'exemple, ce bruit peut induire une absorption analogue à des raies peu intenses dans le spectre ou bien au contraire cacher certaines raies dont l'intensité est relativement faible et qui seraient noyées dans ce bruit.

La technique d'enregistrement mise en place au laboratoire a déjà permis de réduire considérablement une partie de ce bruit; il s'agit de la contribution aléatoire du bruit. L'autre contribution qui est périodique est plus difficile à enlever et il faut filtrer les spectres de manière digitale afin de réduire les effets de cette contribution.

Le bruit que l'on enregistre lors des mesures expérimentales est essentiellement composé de signaux sinusoïdaux à hautes fréquences tandis que les spectres d'absorption sont quant à eux constitués par la superposition de signaux sinusoïdaux aussi bien à basses que à hautes fréquences; les composantes basses fréquences étant sensiblement plus intenses que celles à hautes fréquences. Si on supprime les composantes hautes fréquences de ces spectres, on perd évidemment de l'information mais il est toujours possible de les reconstituer de manière très satisfaisante; il serait donc intéressant de supprimer ces hautes fréquences afin d'éliminer le bruit. Cette suppression peut s'effectuer en multipliant le spectre des fréquences[1] constituant un spectre d'absorption par une fonction rectangulaire. Or multiplier par une fonction rectangulaire le spectre des fréquences revient à

---

[1] Tout signal périodique $s(t)$ de période T peut se décomposer en une somme de signaux sinusoïdaux telle que :

$$s(t) = s(t+T) \qquad \forall\, t$$

$$s(t) = \sum_{i=0}^{\infty} a_i \sin(2\pi\nu_i t + \varphi_i)$$

Le spectre des fréquences du signal étant donné par la relation $a_i = a_i(\nu_i)$.

Lorsque l'on considère une telle composition de signaux sinusoïdaux dont les fréquences varient de manière continue et non plus de façon discrète, la somme de l'expression précédente est remplacée par une intégrale qu'on appelle intégrale de Fourier. Cette intégrale permet de représenter un signal non périodique sous forme d'une superposition de sinusoïdes.

convoluer le spectre d'absorption par un opérateur *sin(x)/x* puisque la transformée de Fourier d'une fonction rectangulaire est de cette forme. Pour des raisons de limitation des temps calculs, nous multiplions la fonction *sin(x)/x* par une fonction triangulaire avant de calculer la convolution; de cette manière, on restreint l'étendue de l'opérateur. C'est cette technique que nous employons pour réaliser un filtrage des spectres enregistrés; pour plus d'informations, le lecteur peut consulter la référence [16]. La figure III.2 montre un spectre d'absorption avant et après l'opération de filtrage; on peut constater que le spectre ainsi obtenu est plus lisse.

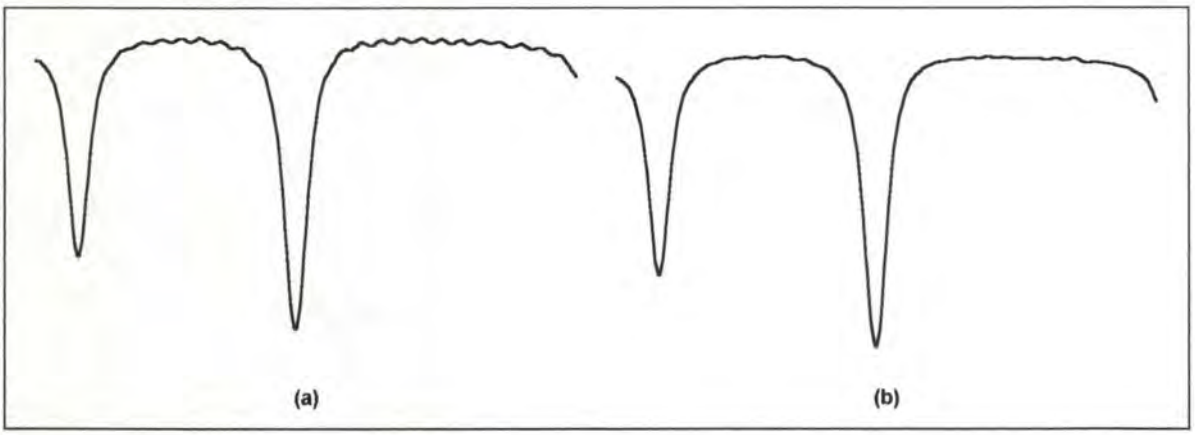


Figure III.2 Un spectre d'absorption avant (a) et après (b) filtrage.

4. L' "ajustement" du fond (Programme **AJUSPC** sur PC)

Lors d'une acquisition, les quatre différents spectres ne sont pas enregistrés simultanément mais les uns à la suite des autres à quelques secondes, voire quelques minutes d'intervalle. A cause de diverses instabilités, il arrive fréquemment que ces spectres ne soient pas superposables tels que nous les avons enregistrés. Par exemple, lorsque nous superposons le fond et le spectre, on constate très souvent un léger décalage entre ces deux enregistrements; ce décalage étant nettement visible lorsqu'on construit le spectre redressé (*i.e.* le spectre obtenu en divisant le spectre enregistré par le fond). La figure III.3 illustre ce décalage où on peut observer que les franges d'interférence (*i.e.* du bruit résiduel) ne sont pas en phase dans les enregistrements du fond et du spectre; cela se traduit par toute une série d'oscillations dans le spectre redressé. Les sommets des franges apparaissant à des fréquences bien précises, cela signifie que ces spectres sont quelque peu décalés.

Ce décalage qu'on appelle décalage "horizontal" n'est pas le seul et on peut également observer des décalages "verticaux"; ce qui signifie que les niveaux de 100

% de transmission du fond et du spectre ne coïncident pas. Or, il est indispensable que les spectres soient superposables; c'est-à-dire que le point n° *i* de l'enregistrement du spectre coïncide en fréquence avec le point n° *i* de l'enregistrement du fond et que leur niveau de 100 % de transmission aient la même valeur. Dans ce travail, lorsque l'on ajustera des modèles sur les profils d'absorption, il faudra calculer les rapports du fond sur le spectre en chacun des points; par conséquent il faut absolument que ces deux spectres coïncident le mieux possible.

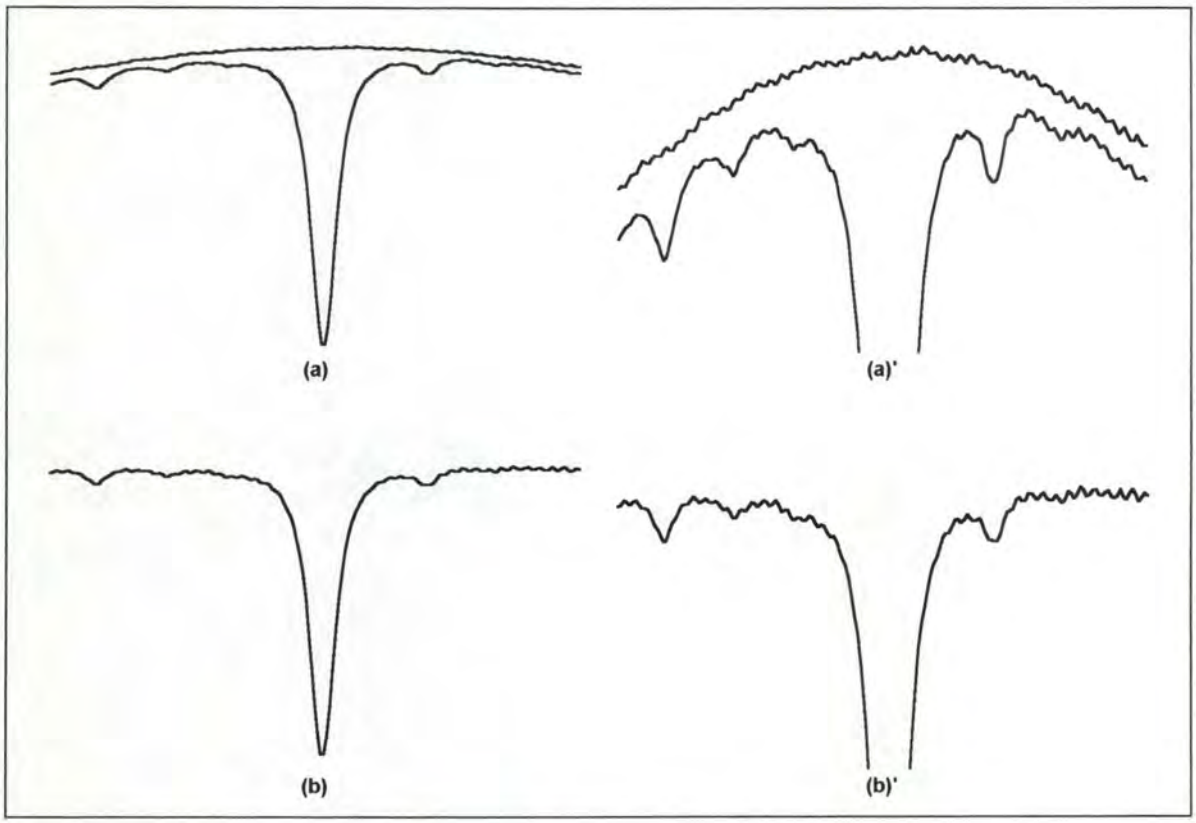


Figure III.3 (a) le spectre et le fond enregistrés, (b) le spectre redressé, (a)' et (b)' idem mais amplifiés.

Il faut donc être capable de modifier légèrement les spectres que nous avons enregistrés; nous avons décidé de ne pas modifier le spectre mais de déplacer uniquement le fond de façon à ce qu'il se superpose au mieux sur le spectre. Les déplacements du fond sont effectués de manière interactive avec une visualisation du résultat soit sous forme de superposition du spectre et du fond, soit sous forme du spectre redressé.

Il arrive parfois que ces déplacements ne soit pas suffisants et c'est pourquoi on permet également à l'utilisateur de faire une rotation du fond afin que celui-ci s'ajuste au mieux sur le spectre. La figure III.4 représente les mêmes spectres que ceux de la figure III.3 mais après déplacement du fond; on peut constater que le

spectre et le fond sont en phase et que cela se répercute évidemment sur le spectre redressé où les oscillations sont nettement atténuées.

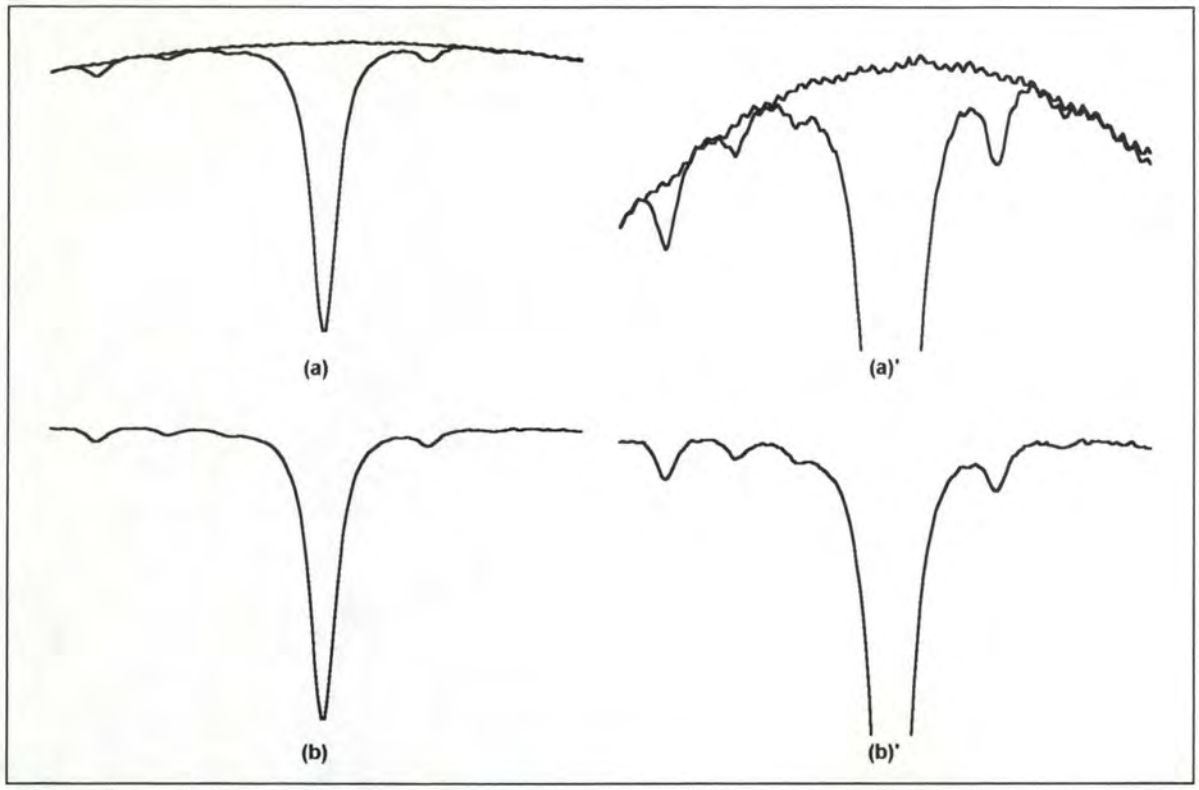


Figure III.4 (a) le spectre enregistré et le fond déplacé, (b) le spectre redressé, (a)' et (b)' idem mais amplifiés.

5. La détection des raies des spectres (Programme **POSIPC** sur PC)

L'étape suivante dans la chaîne de traitement des spectres est la détection des raies qu'ils contiennent; cette détection s'applique au spectre Doppler et au spectre proprement dit. Le résultat est une liste reprenant pour chacune des raies détectées :

- sa position en point fichier.
- son intensité, c'est-à-dire une mesure relative de sa hauteur.
- le numéro du spectre auquel elle appartient.

ces informations étant ajoutées à la fin du fichier contenant les enregistrements.

Au cours de ce traitement, on détecte également la position des franges de l'étalon (en point fichier); ces positions permettront par la suite de calibrer chaque point du spectre en fréquence relative en faisant l'hypothèse qu'entre deux franges

successives la fréquence varie linéairement. Nous verrons l'utilité de cette détection dans la section consacrée à la linéarisation des spectres.

La détection des raies d'un spectre est réalisée en repérant toutes celles pour lesquelles la dérivée seconde de ce spectre dépasse un niveau arbitraire constant. Cette méthode de détection est très utile car elle permet d'éliminer le fond continu du spectre qui varie en fonction de la fréquence; en effet, lorsqu'on observe un spectre d'absorption non redressé, le niveau de 100 % de transmission ne prend pas une valeur constante pour toutes les fréquences mais varie continûment avec la fréquence. La figure III.5 illustre un spectre d'absorption ainsi que ses dérivées première et seconde; on peut observer 5 raies qui sont effectivement détectées en recherchant les valeurs de la dérivée seconde qui dépasse un niveau constant représenté par une ligne horizontale.

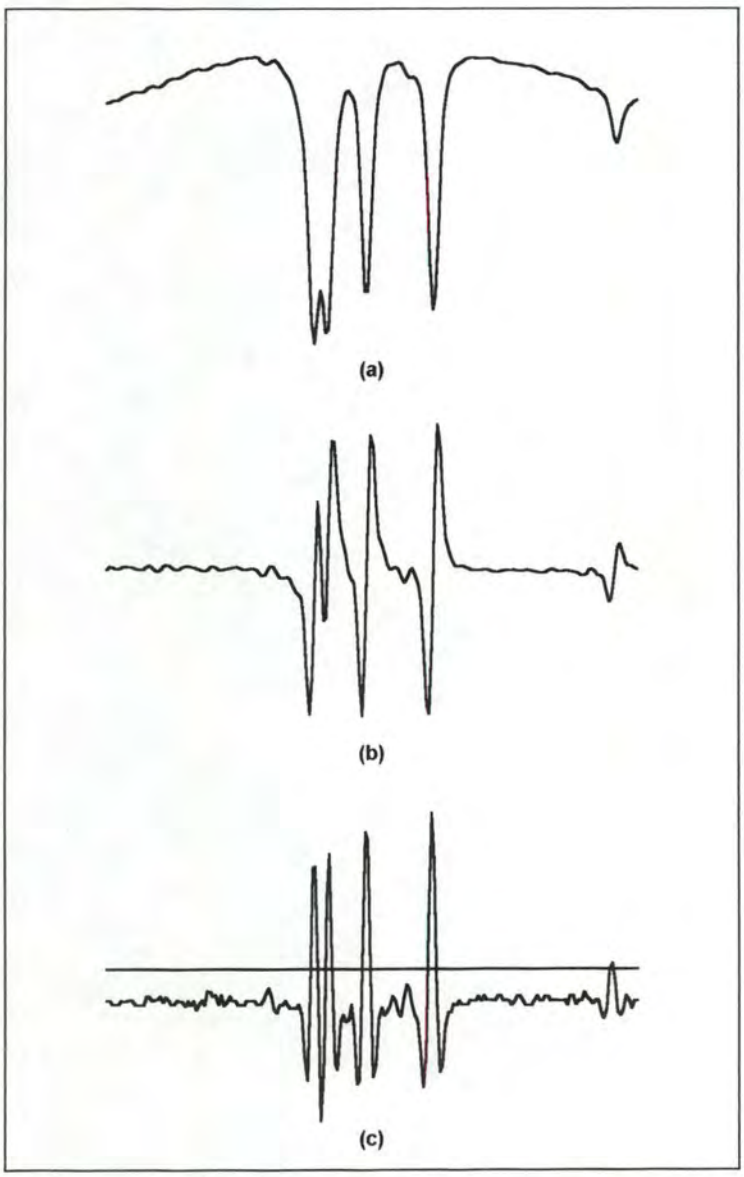


Figure III.5 Un spectre d'absorption (a) et ses dérivées première (b) et seconde (c).

Les positions des raies sont tout d'abord approchées en recherchant les valeurs maximales de la dérivée seconde du spectre. Ensuite elles sont affinées de la manière suivante : pour chaque maximum détecté, on définit un intervalle de longueur arbitraire s'étendant de part et autre de sa position. Dans cet intervalle, on recherche alors les plus grande et plus petite valeurs; ce qui donne la hauteur apparente de la raie que l'on divise en neuf niveaux équidistants. La position de la raie est alors déterminée par la valeur moyenne des milieux des trois cordes les plus proches du sommet de la raie (cfr. figure III.6); c'est-à-dire les cordes 1, 2 et 3.

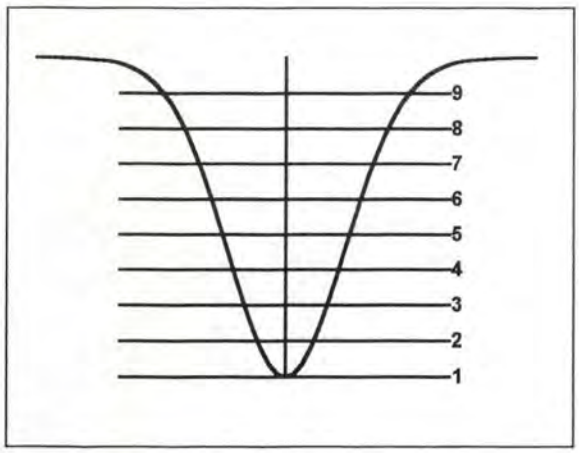


Figure III.6 Recherche de la position maximale par la méthode des cordes.

Lorsqu'il y a des épaulements dans les raies ou bien lorsque des raies superposées ne sont pas résolues, on n'effectue pas la recherche des cordes et la position de la raie est la position du maxima de sa dérivée seconde.

Le choix du niveau de détection est important et influence considérablement le résultat. Par exemple, si le niveau de détection est trop bas, on détectera des raies qui correspondent à du bruit; par contre s'il est trop élevé, on risque de ne pas détecter des petites raies. Il faut trouver un compromis entre ces deux situations. De même, si l'intervalle autour du maximum est mal choisi (trop grand ou trop petit), on court le risque d'obtenir une position quelque peu faussée en appliquant la méthode des cordes.

Notons qu'à la fin de cette étape, c'est-à-dire après la détection des raies, on réalise un nouveau lissage des spectres afin de réduire au minimum le bruit. Ce lissage complète le précédent qui ne pouvait être trop intense sous peine d'éliminer de petites raies qui auraient été considérées comme étant du bruit.

6. La linéarisation des spectres (Programme **LINR** sur VAX)

Les spectres enregistrés qui représentent une mesure de la transmission en fonction de la fréquence ne sont pas linéarisés en ce sens que la différence de fréquence entre deux points fichier successifs n'est pas constante; cette non-linéarité étant induite par des phénomènes physiques lors de l'enregistrement. Pour la suite des traitements, il serait plus commode d'avoir à notre disposition des spectres linéarisés afin de ne pas devoir effectuer des calculs complexes pour attribuer une fréquence relative à tel ou tel point fichier; opération très souvent effectuée dans l'ajustement de modèles.

Connaissant avec une grande précision la distance théorique entre deux franges successives de l'étalon de Fabry-Perot (0.0097711 cm^{-1}) et possédant suite au traitement précédent les positions en point fichier de chacune de ces franges enregistrées, il nous est possible de déterminer une relation, en l'occurrence un polynôme de degré 3, qui nous donne pour un point fichier quelconque sa fréquence relative par rapport à la première des franges détectées. Nous disposons ainsi pour chaque point fichier d'une fréquence et d'une mesure de la transmission. Nous ajustons alors localement des *splines* sur ces résultats; ce qui nous permet d'interpoler la valeur de la transmission en une fréquence particulière (pour autant que celle-ci soit comprise entre les fréquences des premier et dernier points fichier).

Les *splines* sont des objets mathématiques qui permettent d'interpoler des courbes sur un ensemble ordonné de points $x_0 \leq x_1 \leq x_2 \leq \ldots \leq x_n$. Prenant deux points successifs de cet ensemble (x_i et x_{i+1}), il est possible de faire passer un polynôme de degré 3 par ces points

$$\Phi(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

$$x_i \leq x \leq x_{i+1}$$

de telle façon qu'il existe une continuité de position, de tangence et de courbure avec les *splines* définis sur les points immédiatement voisins. Si on a n points à interpoler, on construit n intervalles et on détermine $4.(n-1)$ coefficients $a_{0,1}$, $a_{1,1}$, $a_{2,1}$, $a_{3,1}$, ..., $a_{0,n-1}$, $a_{1,n-1}$, $a_{2,n-1}$, $a_{3,n-1}$. On peut par conséquent maintenant déterminer la valeur de la fonction d'interpolation en n'importe quel point x en évaluant la fonction $\Phi(x)$ avec les quatre coefficients appropriés.

Nous pouvons donc construire à l'aide des *splines* calculés un spectre linéarisé en fréquence en interpolant la valeur de la transmission pour des fréquences régulièrement espacées de 10^{-4} cm^{-1} et telles que le spectre ainsi construit soit centré (*i.e.* que la raie la plus intense se trouve au centre du spectre). La figure III.7 montre un spectre avant et après linéarisation.

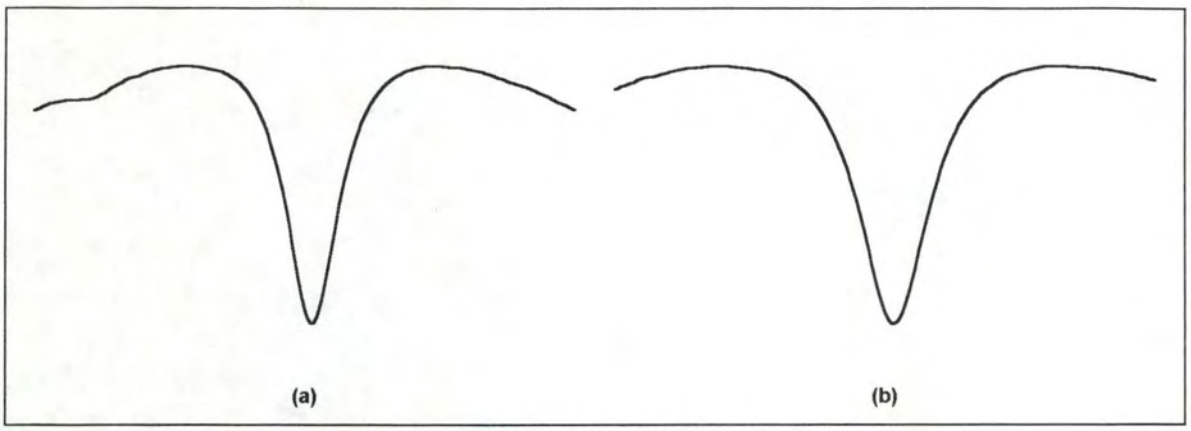


Figure III.7 Un spectre d'absorption avant (a) et après (b) linéarisation.

Chapitre IV

Les ajustements de profils

1. Introduction

Dans ce chapitre, nous allons présenter les programmes d'ajustements de profils que nous avons réalisés; ces programmes écrits en FORTRAN fonctionnent sur une machine VAX et utilisent des procédures des librairies mathématiques IMSL (Ref. [17]). Deux programmes de traitements ont été écrits : un pour l'analyse des profils de raies isolées (PROFIL), l'autre pour l'analyse des profils de raies multiples (MULTI).

Le but de ce chapitre n'est pas de décrire ce que font les différentes procédures et fonctions des programmes, cela se trouve dans les listings, mais plutôt de présenter les problèmes rencontrés dans la réalisation ainsi que d'expliquer les solutions apportées. Le code des programmes n'est pas compliqué en lui-même mais renferme quelques subtilités que nous présentons ici.

Dans ce chapitre, nous présentons également des résultats que nous avons obtenus en appliquant nos programmes à des spectres enregistrés au laboratoire.

2. L'ajustement de profils sur des raies isolées

Dans un premier temps, nous nous sommes limités à adapter des profils d'absorption sur des spectres ne contenant que des raies isolées. Par raies isolées, nous entendons des raies pour lesquelles il n'existe pas d'interférences avec d'autres raies voisines; c'est-à-dire que les spectres étudiés sont choisis dans la mesure du possible de telle sorte qu'ils ne contiennent qu'une seule raie ou que les raies voisines les plus proches de la raie étudiée soient suffisamment éloignées pour ne pas perturber son profil.

Les spectres sur lesquels nous travaillons ici auront été au préalable filtrés, ajustés et linéarisés; la détection des raies aura également été effectuée. Les spectres dont nous disposons pour l'ajustement des modèles de profils semblent donc être

parfaits et devraient contenir toute l'information nécessaire. On pourrait dès lors penser qu'il suffit de passer les données des enregistrements des spectres à une routine de moindres carrés non linéaires pour que celle-ci nous sorte les valeurs optimales des paramètres. Ce n'est évidemment pas aussi simple et si on travaille de cette façon, on court tout droit à la catastrophe : la routine est incapable de nous sortir le moindre résultat significatif d'un point de vue "physique".

Nous avons en effet rencontré tout un ensemble de problèmes lorsque nous avons réalisé le programme d'ajustement; ces problèmes ayant des origines multiples liées à la méthode des moindres carrés utilisée ou encore aux données elles-mêmes. Il y a donc tout un ensemble de considérations à prendre en compte et la réalisation du programme d'ajustement de profils n'est pas aussi simple qu'il n'y parait si on veut obtenir des résultats significatifs ayant une certaine valeur aux yeux des spectroscopistes. Les solutions apportées aux différents problèmes sont réalisées automatiquement par le programme; à l'exception du choix de l'intervalle à prendre en considération qui peut être déterminé automatiquement ou par l'utilisateur.

2.a. Les données prises en compte pour les ajustements.

La première chose à faire lorsqu'on veut ajuster un profil sur nos enregistrements de spectres, c'est de calculer le profil d'absorption observé. Ce profil d'absorption en un point i se calcule aisément de la manière suivante (*cfr.* Chapitre I) :

$$K(i) = \frac{1}{l} \ln \frac{I_0(i)}{I_1(i)}$$

où $I_0(i)$ est la mesure de l'intensité du fond au point i,
$I_1(i)$ est la mesure de l'intensité du spectre au point i,
l est l'épaisseur de l'échantillon traversé.

La question qui se pose ici est de savoir si on doit réaliser l'ajustement sur le profil d'absorption calculé en entier ou bien si on peut se limiter à une partie de celui-ci. D'une part il semble que plus on a de points expérimentaux, meilleur sera l'ajustement; il faudrait donc prendre le profil en entier. Mais malgré les traitements, les spectres ne sont pas encore parfaits; il subsiste toujours un léger bruit de fond sur les mesures. Lors de l'ajustement ce bruit est négligeable sur la raie mais lorsqu'on s'éloigne de son centre et qu'on se retrouve dans les ailes, ce n'est plus du

tout le cas. Le bruit a en moyenne une intensité constante quelle que soit sa position; qu'il s'agisse du centre de la raie et de son voisinage ou bien des ailes. En son centre, le profil prend des valeurs élevées et le bruit y est négligeable; par contre dans les ailes, la valeur du profil tend vers 0 et là le bruit ne peut plus être considéré comme négligeable. A titre d'exemple si le bruit représente 1% du signal au centre de la raie (donc négligeable à cet endroit), ce pourcent est important par rapport à 0 dans les ailes. Il ne faut donc pas s'aventurer trop loin dans celles-ci.

Une autre constatation en faveur de cette idée est que les spectres enregistrés ne contiennent pas toujours une seule raie et si l'on tente d'ajuster un profil sur un tel spectre, on va rencontrer des problèmes. En effet, la routine des moindres carrés non linéaires va tenter d'ajuster les paramètres du modèle de manière à ce que celui-ci colle le mieux possible sur le profil du spectre enregistré. Or le modèle ajusté concerne uniquement une seule raie tandis que le spectre en contient plusieurs; par conséquent la routine va tenter son ajustement en prenant en compte des contributions inappropriées.

Nous proposons deux méthodes à l'utilisateur pour déterminer l'étendue de l'intervalle à prendre en compte pour réaliser l'ajustement. La première est une méthode automatique qui limite l'intervalle d'étude lorsque les valeurs dans les ailes du profil observé sont inférieures à x % de la valeur du profil au centre de la raie (valeur où il est maximum); une valeur de 4% donne des résultats très satisfaisants. La seconde méthode permet à l'utilisateur de déterminer lui même l'intervalle en donnant les positions des points fichiers de ses extrémités.

2.b. L'influence de l'appareillage de mesures sur les données enregistrées.

Les appareils de mesures (et notamment notre spectromètre) n'ont malheureusement pas des résolutions infinies et les spectres enregistrés sont en fait le résultat d'une convolution du spectre réel avec une fonction qui dépend de l'appareillage utilisé; fonction qu'on appelle *fonction d'appareil* $f(\sigma)$. Le signal observé pour nos mesures d'absorption est donc:

$$A_{obs}(\sigma) = \int_{-\infty}^{+\infty} A_{réel}(\sigma') * f(\sigma-\sigma')\, d\sigma'$$

avec

$$A_{réel}(\sigma') = 1 - \exp\left[- K(\sigma')\, l\right]$$

Le profil d'absorption que nous avons calculé précédemment est en fait le profil observé et pas le profil réel; c'est pourtant ce dernier qui nous préoccupe. Pour obtenir ce profil réel, il faudrait être capable de déconvoluer le signal observé; cette opération est très difficile à effectuer et nous allons utiliser une astuce pour contourner le problème de la fonction d'appareil.

La fonction d'appareil de notre spectromètre est induite par des phénomènes aléatoires; ce qui nous autorise à la représenter sous forme d'une fonction gaussienne de la forme suivante :

$$f(\sigma - \sigma') = \frac{1}{\gamma^{app}} \sqrt{\frac{\ln 2}{\pi}} \exp\left[-\ln 2 \left[\frac{\sigma - \sigma'}{\gamma^{app}}\right]^2\right]$$

Nous disposons d'un spectre, en l'occurrence le spectre de référence, qui est enregistré à faible pression de telle sorte que l'on puisse faire l'hypothèse que son profil est de type Doppler. Or un profil de type Doppler est représenté par une gaussienne. De plus la somme de deux gaussiennes (de largeur à mi-hauteur γ_1 et γ_2) est également une gaussienne dont la largeur à mi-hauteur γ est telle que :

$$\gamma^2 = \gamma_1^2 + \gamma_2^2$$

Nous pouvons donc tenter d'ajuster une gaussienne sur le profil du spectre enregistré à basse pression telle que celle-ci contienne à la fois le profil d'absorption Doppler et la fonction d'appareil. Par la suite lorsque nous ajustons des profils plus compliqués, nous faisons l'hypothèse que la contribution Doppler du profil contient implicitement la fonction d'appareil. De cette manière, nous pouvons manipuler les profils comme s'il n'y avait plus de fonction d'appareil. Evidemment, nous ne pourrons jamais obtenir une valeur pour la largeur Doppler réelle mais ce n'est pas grave car nous disposons des valeurs théoriques de ces largeurs.

2.c. La normalisation des profils et l'intensité des raies d'absorption.

Les expressions théoriques des profils données au chapitre consacré à la spectroscopie moléculaire sont relatives à des profils normalisés; c'est-à-dire des profils pour lesquels la surface totale comprise entre l'axe des abscisses et le profil lui-même est unitaire. Or les profils que nous enregistrons ne sont pas normalisés. Nous avons décidé de ne pas effectuer les opérations de normalisation qui prennent du temps calcul mais plutôt de placer devant chaque expression de profil un paramètre qui sera un facteur d'échelle et qui fournira une mesure de la surface. En spectroscopie, cette surface permet de déterminer une grandeur caractéristique d'une raie d'absorption qui est son intensité S définie par :

$$S = \frac{1}{p} \int_0^{+\infty} \frac{1}{l} \ln \frac{I_0(\sigma)}{I_t(\sigma)} \, d\sigma$$

où p est la pression du gaz et l'intégrale est la valeur de la surface engendrée par le profil.

2.d. Les valeurs initiales des paramètres.

Comme nous l'avons mentionné précédemment (*cfr.* le chapitre II sur la modélisation des données), il faut attribuer des valeurs approchées aux paramètres que l'on veut ajuster pour initialiser la recherche de leur valeur optimale. Lors de la mise au point du programme, nous avons pu constater que le choix de ces paramètres est très important et si les valeurs initiales ne sont pas données de manière appropriée, la recherche nous donne des résultats tout à fait non significatifs. Nous sommes alors dans un minimum local qui fournit une solution qui n'est pas acceptable et par conséquent les valeurs initiales des paramètres ne sont pas arbitraires.

Nous déterminons principalement trois types de paramètres : des largeurs, des déplacements en fréquence et des intensités. En ce qui concerne ces deux derniers, il n'y a pas trop de problèmes pour en évaluer une valeur approchée. En effet, les déplacements en fréquence étant très minimes, nous pouvons leur donner une valeur nulle pour débuter la recherche. Quant aux intensités, il est toujours possible, quel que soit le modèle, de trouver une relation qui les relie à la hauteur au centre du profil et il n'y a donc pas non plus de difficulté puisque cette valeur est

calculée. Pour approcher les valeurs des paramètres associés aux largeurs, c'est un petit peu plus subtil et leur détermination est fonction du profil étudié.

Parmi les cinq modèles que nous étudions, il y en a des simples (Doppler et collisionnel) et des complexes (Voigt, Voigt généralisé et Galatry). Les modèles simples ne font intervenir qu'une seule largeur dans leur expression tandis que les modèles complexes comprennent à la fois la largeur Doppler et la largeur collisionnelle.

Pour le profil Doppler, nous approchons la valeur de la largeur par sa valeur théorique qui peut être obtenue par la relation suivante :

$$\gamma^D = 3.58\ 10^{-7} \sqrt{\frac{T}{M}}\ \sigma_0$$

où T est la température du gaz.
M est la masse moléculaire du gaz.
σ_0 est la fréquence du centre de la raie.

Evidemment la largeur qui sera déterminée par ajustement sera supérieure à cette valeur théorique puisque la contribution de la fonction d'appareil y est incluse.

Pour les profils collisionnels, nous déterminons leur largeur approchée sur base du profil enregistré : nous recherchons la hauteur maximale *Max* du profil et sa largeur est l'intervalle entre les points situés de part et d'autre de cette valeur maximale pour lesquels le profil à une valeur égale à *Max* / 2.

En ce qui concerne les profils complexes, nous avons décidé de considérer la largeur Doppler non pas comme un paramètre mais comme une constante qui est déterminée par ajustement d'un profil Doppler sur le spectre de référence enregistré à basse pression. Cette hypothèse se justifie par le fait que la largeur Doppler est indépendante de la pression. La largeur collisionnelle par contre est approchée en ajustant un profil collisionnel sur le spectre étudié.

Par conséquent lorsqu'on adapte des profils complexes, il y a toujours deux ajustements qui sont réalisés avant celui du modèle que l'on a choisi : un ajustement Doppler sur la référence et un ajustement collisionnel sur le spectre.

3. L'ajustement de profils sur des raies multiples

Nous nous attaquons ici à la modélisation de profils sur un ensemble de raies multiples. Les spectres analysés ici comprennent plusieurs raies isolées ou ce qui est nettement plus intéressant des raies non résolues suite à la superposition de raies voisines. Les profils que nous ajustons sur ces spectres sont en réalité des sommes de profils; avec la restriction que les profils sommés sont tous de même type : somme de gaussiennes, somme de profils de Voigt, ...

Les mêmes considérations que celles de la section précédente peuvent s'appliquer pour ces ajustements et d'ailleurs, le programme est fort semblable; la seule différence notoire concerne le choix des paramètres à ajuster. Dans le cas de raies isolées le nombre de paramètres à ajuster est relativement faible, de l'ordre de deux ou trois selon le profil étudié, et il n'y a aucun problème lié à ce nombre. Par contre lorsque l'on a plusieurs raies il faut multiplier ce nombre de paramètres par le nombre de raies; on arrive ainsi très vite à une dizaine de paramètres à déterminer.

Le nombre de paramètres à ajuster correspond au nombre de degrés de liberté accordés à la fonction de mérite χ^2 ; si ce nombre est trop élevé, la recherche de la solution optimale est évidemment plus difficile et le temps calcul va augmenter de manière très importante. Il faut donc essayer de réduire la quantité de paramètres variables de notre somme de profils.

Dans chaque élément de la somme (*i.e.* pour chaque raie), il y a un paramètre qui correspond au déplacement en fréquence de la raie par rapport à sa position détectée. Or nous avons pu constater que ce déplacement est constant pour un ensemble de raies voisines. Par conséquent, nous pouvons réduire ces paramètres à un seul et nous gagnons $n-1$ degrés de liberté pour l'analyse d'un ensemble de n raies. De cette façon nous réduisons le temps de calcul nécessaire aux ajustements.

4. Résultats obtenus

Nous donnons ci-après un exemple des résultats d'ajustements que nous avons pu obtenir pour des spectres enregistrés au laboratoire. Le premier exemple est l'ajustement d'un profil de Voigt généralisé sur une raie isolée; le spectre noir est le spectre original enregistré tandis que le spectre vert est le spectre calculé après ajustement du modèle.

Les deux exemples suivants concernent des spectres à raies multiples. Le spectre original apparaît en noir, le spectre reconstruit en rouge tandis que les différentes contributions (*i.e.* les différentes raies) apparaissent en vert.

FUNDP-Namur
25-Feb-92

Fichier :
J05601

Diode :
B5

Temp :
1.0688

Courant :
0.1651

Dial :
4047

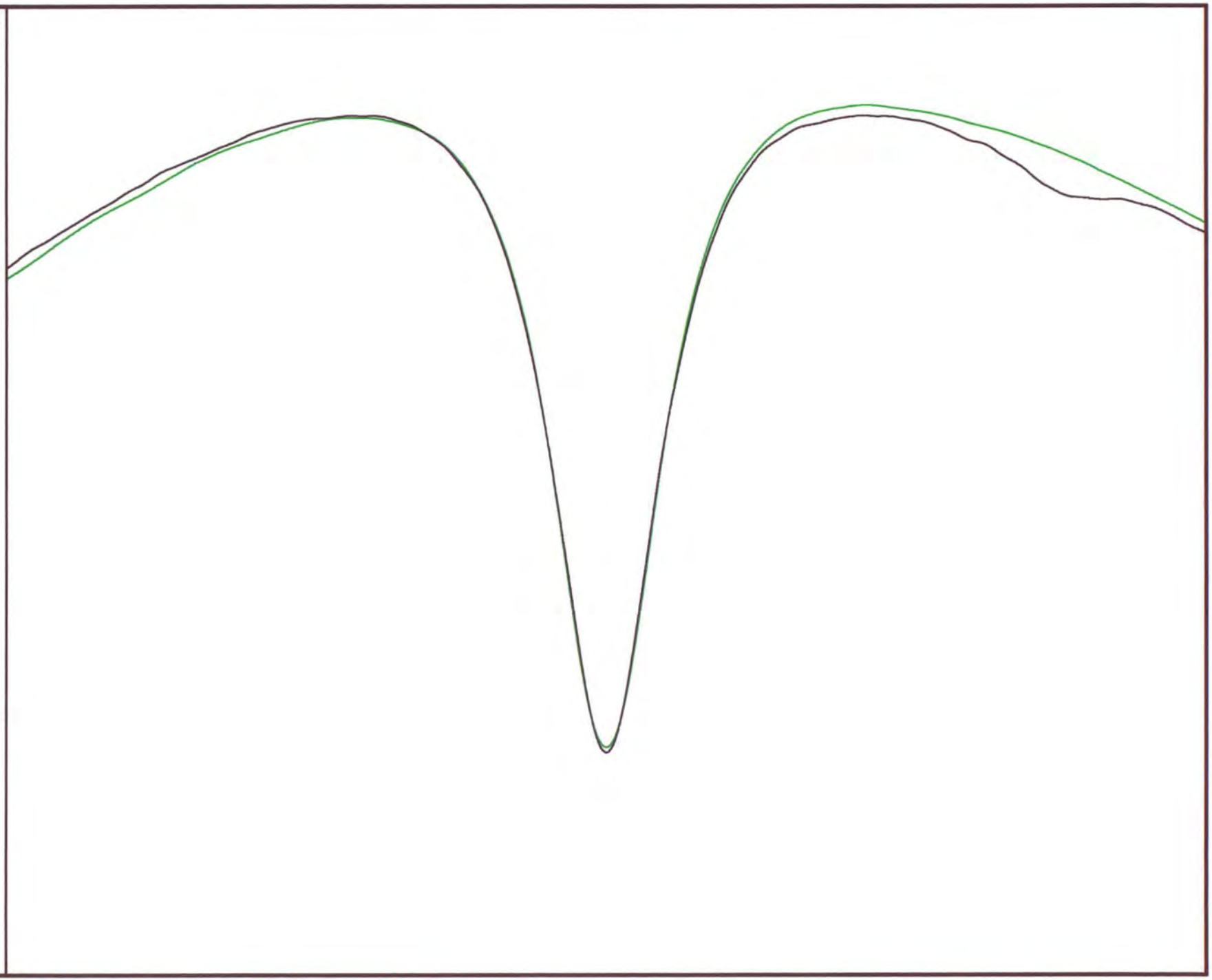
Run(ma) :
135

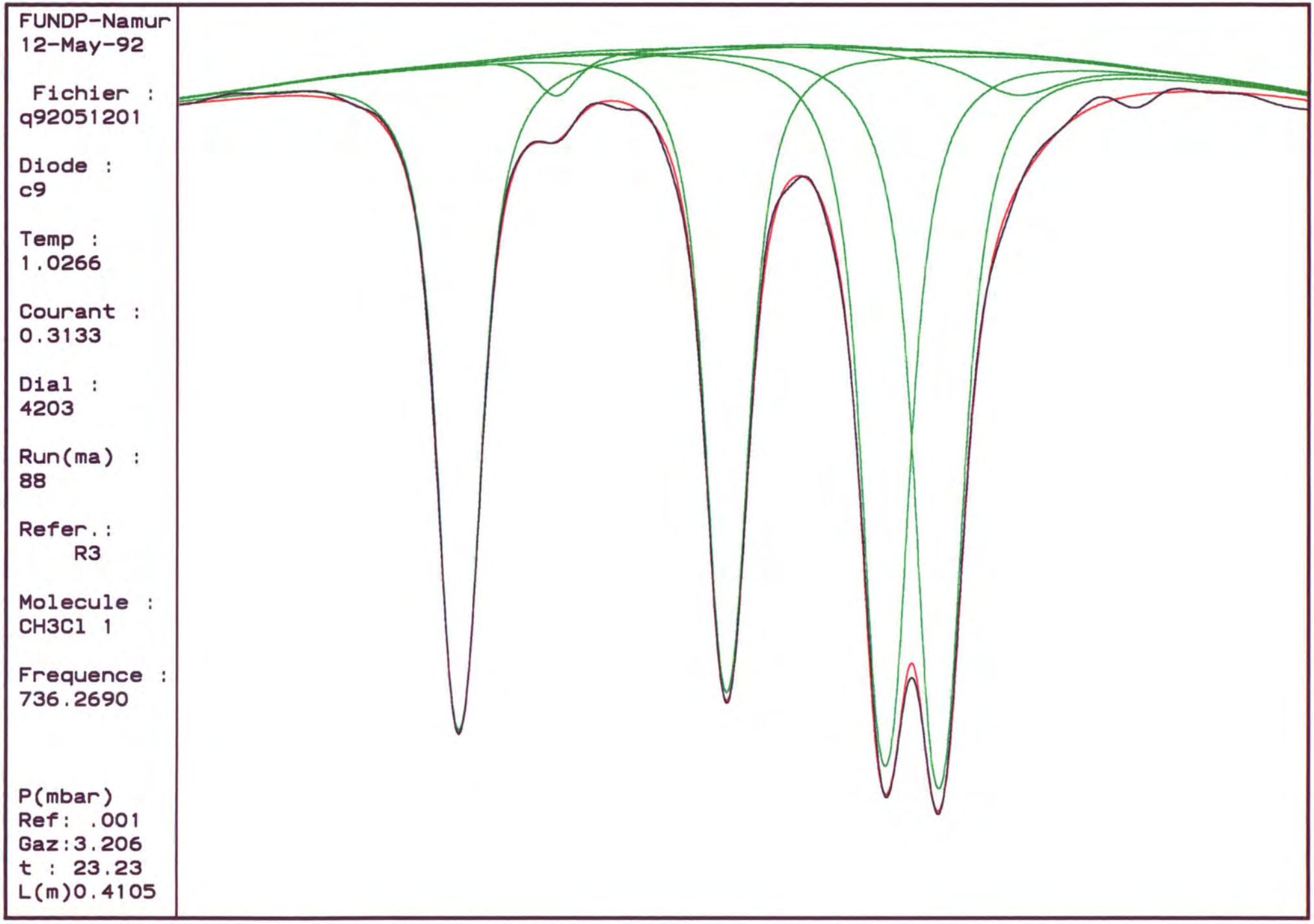
Refer.:
R(14)

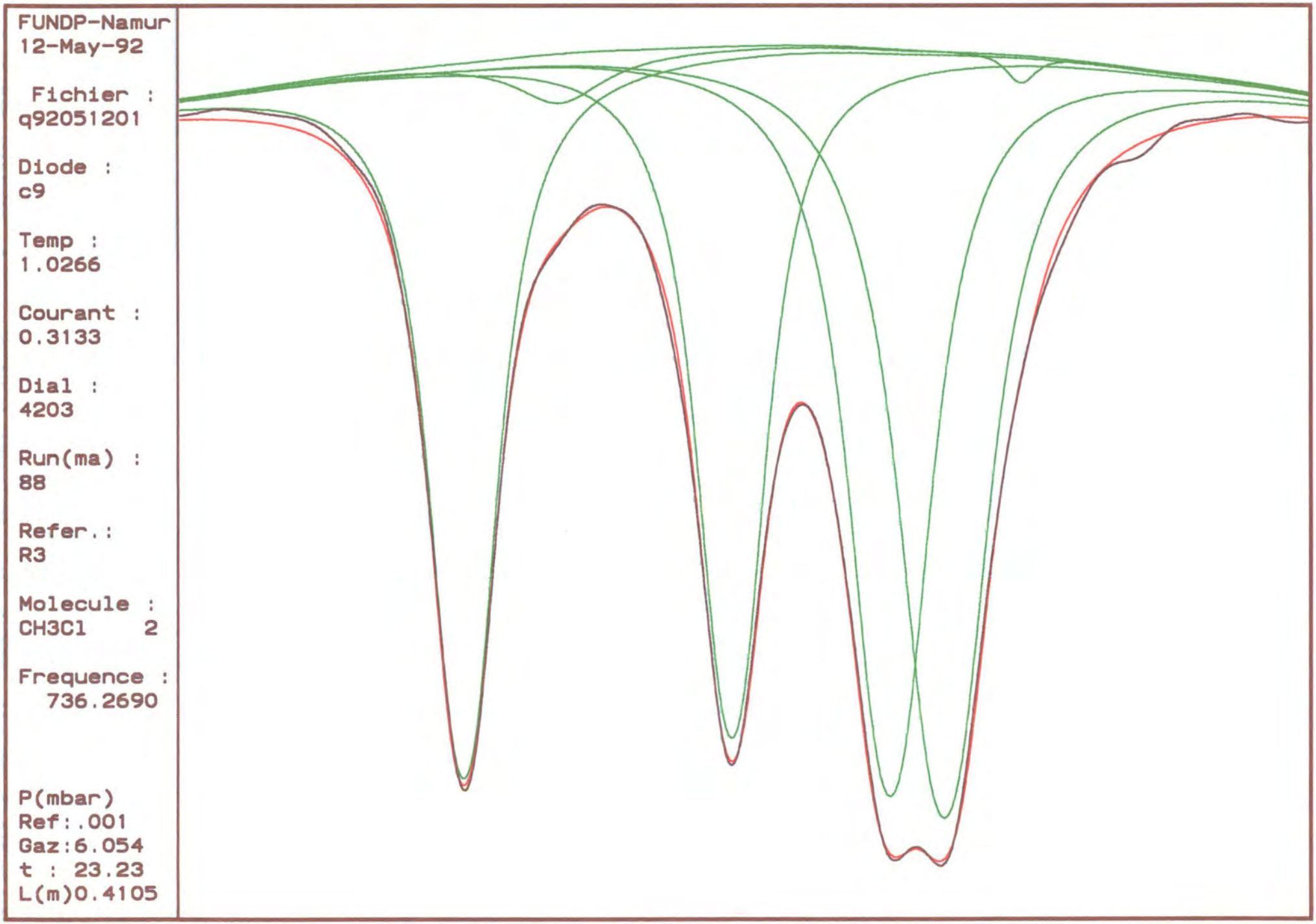
Molecule :
C2H2+Kr 1

Frequence :
764.3821

P(mbar)
Ref:.209
Gaz:153.1
t : 23.3
L(m)0.4105




FUNDP-Namur
12-May-92
Fichier :
q92051201
Diode :
c9
Temp :
1.0266
Courant :
0.3133
Dial :
4203
Run(ma) :
88
Refer.:
R3
Molecule :
CH3Cl 1
Frequence :
736.2690
P(mbar)
Ref: .001
Gaz:3.206
t : 23.23
L(m)0.4105


FUNDP-Namur
12-May-92
Fichier :
q92051201
Diode :
c9
Temp :
1.0266
Courant :
0.3133
Dial :
4203
Run(ma) :
88
Refer.:
R3
Molecule :
CH3Cl 2
Frequence :
736.2690
P(mbar)
Ref:.001
Gaz:6.054
t : 23.23
L(m)0.4105

Chapitre V

La programmation orientée objets

1. Introduction

Ce chapitre est une brève introduction à la programmation orientée objets qui est intensivement utilisée dans l'outil Turbo Vision que nous avons utilisé pour réaliser la partie interface de l'application sur PC. Il n'est pas question ici de refaire une présentation complète des langages orientés objets mais plutôt d'en donner les concepts essentiels et de situer ces concepts dans le langage que nous avons utilisé pour réaliser l'application.

La programmation orientée objets est un nouveau mode de programmation selon lequel un programme réalise une modélisation abstraite d'une partie du monde réel. Dans cette approche, le monde est constitué d'un ensemble d'objets partageant des propriétés communes et interagissant entre eux. Ces objets modélisent à la fois les caractéristiques et les comportements des éléments du monde qui nous entoure.

Le langage que nous utilisons dans ce travail, à savoir le Turbo Pascal 6.0, n'est pas un langage objets pur et dur mais doit être classé dans les langages dits "plus plus" au même titre que le C++. Ces langages consistent en un mélange d'un langage très répandu comme le Pascal ou le C et d'un ensemble de concepts issus du monde des objets. Les extensions objets apportées à ces langages permettent notamment d'écrire des programmes plus structurés, réutilisables et beaucoup plus facile à maintenir.

2. Propriétés des langages orientés objets

Ces langages orientés objets ont trois propriétés caractéristiques qui sont :

- l'encapsulation,
- l'héritage,
- le polymorphisme.

Examinons chacune de ces propriétés.

2.a. L'encapsulation.

Avec des langages de programmation classiques tels que le C ou le Pascal, les variables et le code exécutable d'un programme forment deux entités bien distinctes. D'un côté, on définit des types de données et on déclare le type des variables utilisées dans le programme; d'un autre côté, on écrit des fonctions et des procédures totalement indépendantes des données. En programmation orientée objets il en va tout autrement. En effet, avec ce mode de programmation, on doit dès la conception du programme penser les données et le code comme formant un tout. Ni le code n'existe indépendamment des données; ni les données n'existent indépendamment du code.

L'encapsulation est la propriété selon laquelle les données et le code exécutable sont intimement liés. En programmation orientée objets, un objet est une structure de données, similaire à un ***record*** en Pascal, qui contient des variables mais en outre des procédures et/ou fonctions se rapportant à ces variables; ces procédures et fonctions étant appelées *méthodes*. A la place d'écrire des procédures et des fonctions qui agissent sur des données, en programmation orientée objets on crée des objets qui savent comment réaliser certaines actions sur eux-mêmes.

De la même façon que nous déclarons les variables d'un programme classique comme étant de type entier, réel, caractère, ... nous devons déclarer les objets qui sont manipulés par un programme comme appartenant à un type d'objets (dans le jargon de la programmation orientée objets on parle de *classe* et non de type d'objets). Une classe d'objets constitue un moule à partir duquel nous pourrons créer un ensemble d'objets que l'on appelle *instances* de cette classe. Une classe d'objets définit les propriétés (*i*.e. les données) ainsi que les comportements (*i.e.* les méthodes) communs à toutes les instances issues de celle-ci; elle définit donc le contenu d'un objet. En Turbo Pascal 6.0, la définition d'une classe d'objets est semblable à celle d'un *record* mais comprend en plus les en-têtes des méthodes se rapportant à cette classe; le code associé à ces méthodes n'étant pas défini à l'intérieur de la définition de la classe.

Il est intéressant de remarquer que chaque instance d'une classe possède ses propres champs de données mais que le code de ses méthodes est partagé par tous les objets de la classe (figure V.1). Il existe en effet un lien entre chaque instance et

la classe à laquelle elle appartient qui permet de retrouver les méthodes accessibles de cette instance.

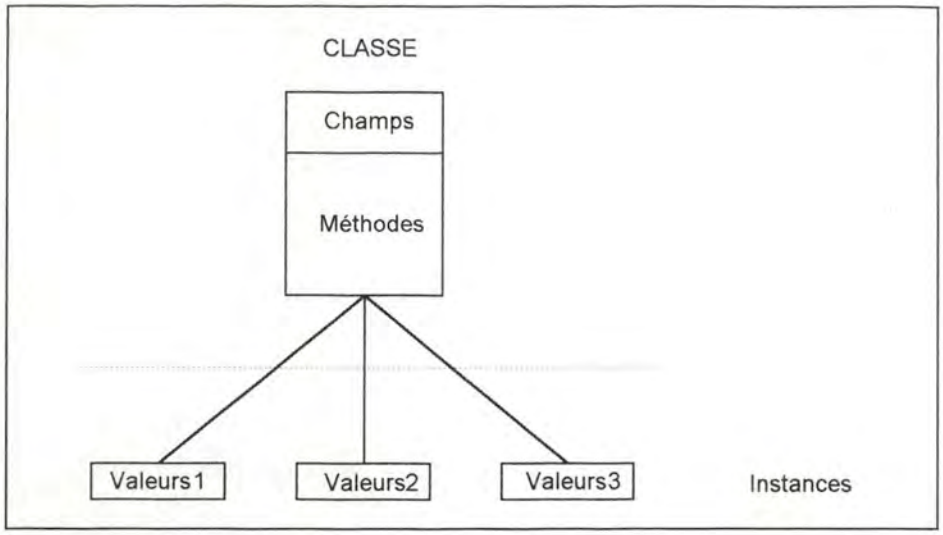


Figure V.1

Dans les vrais langages orientés objets, il est interdit (et d'ailleurs c'est impossible) d'accéder directement aux valeurs des champs d'un objet et le seul moyen d'y parvenir est d'implémenter pour chaque champ un couple de méthodes : l'une permettant de prendre connaissance de la valeur du champ, l'autre permettant de la modifier. Cette façon de pratiquer peut sembler lourde à première vue mais offre l'avantage de réduire au minimum la dépendance entre l'objet et le code du programme ou de ses méthodes.

A titre d'exemple, supposons que nous ayons défini la classe d'objets POINT suivante avec les deux champs X et Y de type réel qui représentent les coordonnées cartésiennes d'un point ainsi qu'un ensemble de méthodes dont une, GetCoord, retourne les coordonnées d'une instance de POINT.

```
POINT = Object
        X : Real;
        Y : Real;
        { Autres champs éventuels }
        Procedure GetCoord (Var X, Y : Real);
        { Autres méthodes éventuelles }
End;
```

Si nous décidons de modifier la structure interne de notre objet POINT et de travailler en coordonnées polaires et non plus en coordonnées cartésiennes, la maintenance du code est grandement facilitée en ce sens que les modifications à effectuer se limitent à la méthode GetCoord dans laquelle nous effectuons une conversion des coordonnées polaires en coordonnées cartésiennes et tout le code des autres méthodes et du programme reste valable. Ce ne serait évidemment pas le cas si on pouvait accéder directement aux valeurs des champs à partir de n'importe où !

En Turbo Pascal 6.0, il est possible d'accéder directement aux champs des objets mais il n'est bien sûr pas conseillé de le faire et il est recommandé de n'accéder à ces champs que via les couples de méthodes appropriées.

2.b. L'héritage.

Cette propriété réfère à la possibilité qu'ont de nouveaux objets d'hériter des propriétés, c'est-à-dire des données et des méthodes, d'autres objets. La programmation orientée objets permet en effet de créer facilement de nouveaux objets en complétant ou en modifiant ceux qui existent déjà. De cette manière, la création de nouveaux programmes est facilitée par la réutilisation de code déjà écrit et se trouvant dans des librairies; le code source de ce code exécutable n'étant même pas nécessaire.

Avec la programmation orientée objets, au lieu de réécrire le code complet pour chaque nouvelle application, on choisit simplement un ensemble d'objets qui approximent plus ou moins les données et les méthodes dont nous avons besoin; ensuite, on complète ces objets afin qu'ils représentent ce que nous désirons.

L'héritage est donc un concept qui permet de définir des classes qui sont presque semblables à d'autres. Les classes qui partagent leurs propriétés avec d'autres sont appelées *ancêtres* tandis que les classes héritières sont appelées les *descendants*. En Turbo Pascal, une classe peut avoir plusieurs descendants directs mais au plus un seul ancêtre direct; c'est ce qu'on appelle l'*héritage simple* par opposition à l'héritage multiple qui autorise une classe à hériter des propriétés de plusieurs autres classes.

La relation d'héritage peut être représentée par un graphe (figure V.2), appelé *graphe d'héritage*, dans lequel plus on descend, plus le degré de spécialisation augmente. Cette spécialisation ayant soit lieu par ajout de données et/ou méthodes soit par substitution de certains champs et/ou méthodes. Remarquons qu'en Turbo Pascal, seules les méthodes peuvent être substituées dans les classes descendantes.

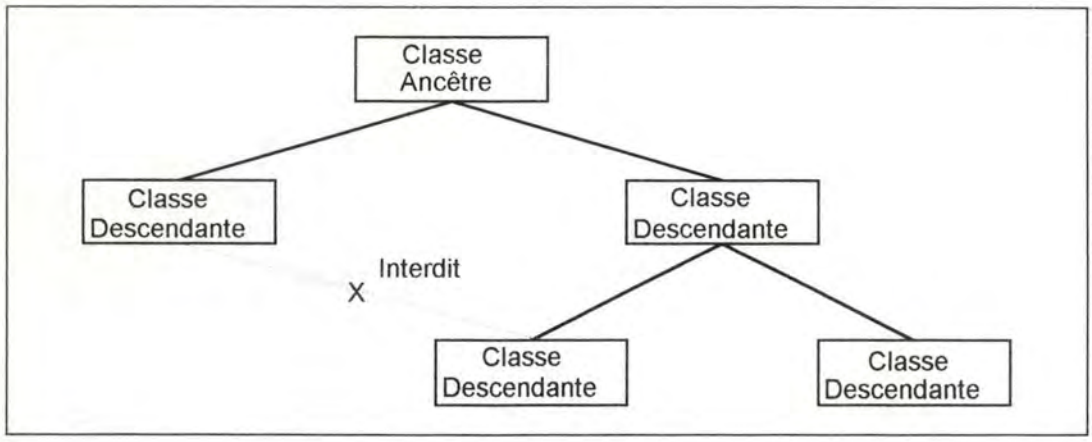


Figure V.2 L'héritage simple.

3.c. Le polymorphisme.

Le polymorphisme est l'aptitude qu'ont des objets à prendre différentes formes lorsque le programme s'exécute; un objet polymorphique peut prendre sa propre forme ou bien la forme de n'importe quel de ses descendants.

Supposons que nous définissions deux classes d'objets *classe1* et *classe2* telles que *classe2* hérite des propriétés de *classe1*. *Classe1* définit deux méthodes *A* et *B* tandis que *classe2* hérite de la méthode *B* tout en substituant une nouvelle version de la méthode *A*. De plus la méthode *B* lorsqu'elle s'exécute appelle toujours la méthode *A* (cfr. figure V.3).

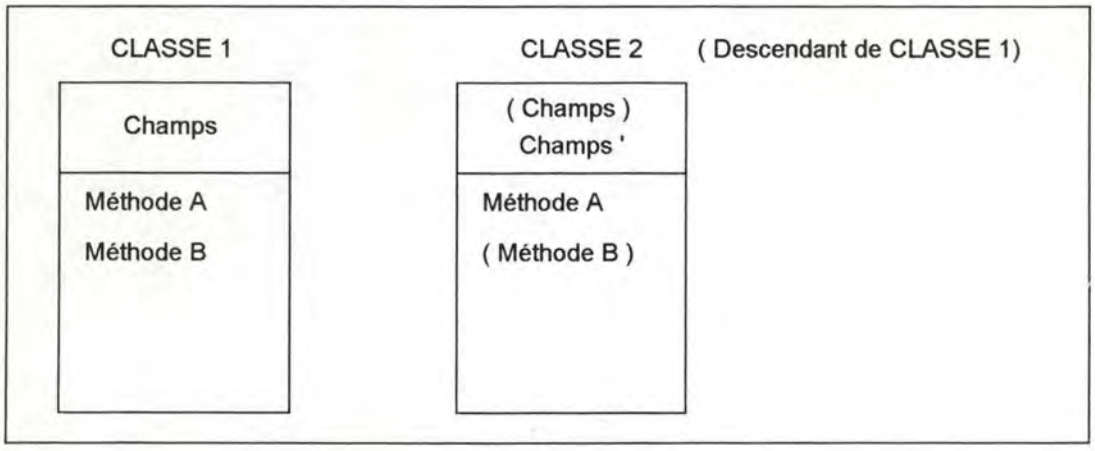


Figure V.3

Soit un objet de la *classe2*. Lorsque sa méthode *B* est exécutée, elle va faire appel à la méthode *A*. Mais quelle méthode *A*; celle de la *classe1* ou celle de la *classe2* ? Le turbo Pascal est un langage compilé et par conséquent toutes les

références sont résolues lors de la compilation. La méthode appelée résulte donc de la logique utilisée par le compilateur pour résoudre les références des appels de méthodes. Le compilateur du Turbo Pascal travaille de la manière suivante : lorsqu'une méthode est appelée par un objet, il recherche tout d'abord si la méthode appelée est définie dans la classe dont l'objet est instancié; si c'est le cas, il remplace l'appel par l'adresse de cette méthode. Par contre, si cette méthode n'est pas déclarée dans la classe de l'objet, c'est qu'elle a été héritée et le compilateur effectue sa recherche parmi les méthodes de l'ancêtre immédiat de la classe; ce processus de recherche étant répété d'ancêtre en ancêtre tant que la méthode n'a pas été localisée. En appliquant cette logique à notre cas, il en résulte que c'est la méthode *A* de la *classe1* qui sera exécutée puisque *B* est définie dans cette classe.

Or si on a redéfini la méthode *A* dans la *classe2*, c'est qu'on aurait voulu que ce soit celle là qui soit exécutée. Ce problème peut être résolu en ne résolvant les références que lors de l'exécution du code et non plus à la compilation; en Turbo Pascal, il suffit de déclarer une méthode comme étant *virtuelle* pour que ses références soient résolues lors de l'exécution du programme.

Toutes les classes d'objets possédant des méthodes virtuelles doivent proposer une méthode d'un type particulier qu'on appelle *constructeur* (le mot clé ***procedure*** étant remplacé par ***constructor***) et qui crée dans le segment des données une table des méthodes virtuelles disponibles. Cette table contient pour chaque méthode virtuelle (et uniquement pour celles-là) un pointeur vers l'implémentation de son code ainsi que la taille des objets appartenant à cette classe. Chaque instance d'une classe doit alors obligatoirement être initialisée par un appel à son constructeur qui crée ainsi un lien entre elle et sa table des méthodes virtuelles. Il faut noter qu'il n'existe qu'une seule table de méthodes virtuelles par classe d'objets et que chaque instance de cette classe contient un lien vers cette table.

De cette manière, en déclarant les méthodes *A* comme étant virtuelles, la méthode *B* d'un objet de la *classe2* appellera bien la méthode *A* définie dans cette classe et non plus celle de la *classe1*.

Le terme polymorphisme prend ici tout son sens puisque lors de la compilation le code ne sait pas précisément quel type d'objet il va manipuler; il sait simplement que ce sera un objet de la classe où est déclarée la méthode ou bien un objet d'une classe descendante. Il faut toutefois bien distinguer le fait qu'à la compilation un objet peut être polymorphique mais pas à l'exécution !

En Turbo Pascal, les objets sont fréquemment des variables dynamiques auxquelles on alloue une place mémoire sur le tas. Lorsque le moment est venu de détruire un objet polymorphique en restituant la zone mémoire qu'il occupait sur le tas, un problème se pose : combien de bytes doit-on libérer ? En effet la taille des objets polymorphiques n'est pas connue lors de la compilation puisqu'on ne sait pas encore quelle forme ils vont prendre; par conséquent il n'est pas possible de prévoir la quantité de mémoire à libérer. Une méthode particulière permet de résoudre ce problème : il s'agit du *destructeur* (***destructor***). Le destructeur d'un objet va consulter la table des méthodes virtuelle de sa classe et y retrouve l'information puisque la taille de ses objets y est indiquée.

Illustrons ce concept de polymorphisme par un exemple. Soit une classe POINT avec les champs *X* et *Y* et une classe CERCLE héritant des propriétés de POINT et possédant un champ supplémentaire *Rayon*; un cercle n'étant rien d'autre qu'un point de rayon non nul (figure V.4).

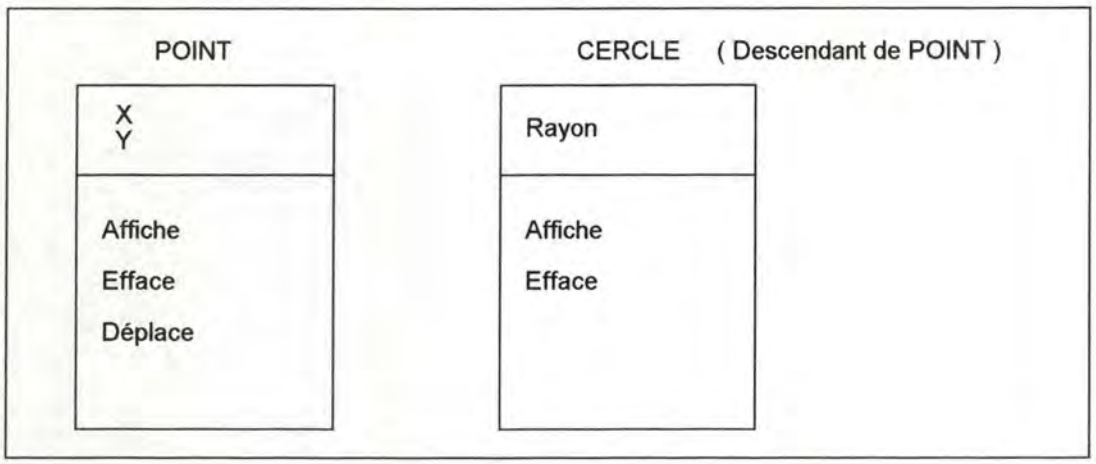


Figure V.4

Lorsque l'on veut déplacer un cercle ou un point, on effectue les mêmes opérations : on l'efface, on modifie ses champs *X* et *Y* et on l'affiche de nouveau. Il n'est dès lors pas nécessaire que CERCLE écrive sa propre méthode de déplacement puisqu'elle est en tout point semblable à celle de POINT. Grâce à la notion de méthode virtuelle, le polymorphisme devient possible et la méthode *Déplace* appellera les méthodes *Affiche* et *Efface* appropriées en fonction de l'objet (point ou cercle) manipulé.

3. Applications

Dans ce travail, la présence de la programmation orientée objets était au départ essentiellement motivée par l'utilisation des librairies Turbo Vision qui sont des librairies d'objets permettant de réaliser des interfaces d'applications standards (cela est explicité dans le chapitre suivant). Il est cependant apparu qu'il serait élégant de représenter les structure de données que l'on manipule dans le programme sous forme d'objets et nous avons donc étendu ces concepts à l'entièreté du programme. Bien sûr, le Turbo Pascal n'est pas un langage objet pur et on retrouve donc dans le programme un mélange de programmation classique (structurée) et de programmation orientée objets.

Le programme est destiné à manipuler des fichiers contenant des spectres d'absorption; ces fichiers ayant toujours la même structure. Ils contiennent un en-tête renfermant des informations sur les spectres enregistrés, une série de spectres sous forme d'un ensemble constant de points, une détection éventuelle des positions des franges et des raies des spectres.

Le contenu d'un fichier représente donc en lui-même le résultat complet d'une acquisition et forme un tout; pourquoi dès lors ne pas le considérer comme étant un objet ? C'est ce que nous avons fait. En regardant un petit peu plus loin dans le contenu du fichier, on s'aperçoit que ses constituants forment également des entités en elles-mêmes. Un spectre est un objet, de même que l'en-tête du fichier ou que les franges et les raies.

Nous avons donc créé une classe d'objet TEntete pour représenter les en-têtes des fichiers, une classe TRecord pour représenter un spectre et les classes TFrangesCollection et TRaiesCollection pour représenter respectivement les franges et les raies détectées. Finalement, nous avons créé une classe TSpectrum dont une instance représente le résultat d'une opération d'acquisition. Chaque instance de TSpectrum contenant un objet de type TEntete, une collection d'objet de type TRecord et éventuellement un objet de type TFrangesCollection ainsi qu'un objet de type TRaiesCollection. Cela peut être représenté sous forme d'un arbre d'appartenance comme sur la figure V.5 qu'il faut veiller à ne pas confondre avec un arbre d'héritage. De manière à pouvoir manipuler tous les spectres du fichier en une seule fois, nous avons également introduit une classe TRecordCollection qui contient des objets de type TRecord.

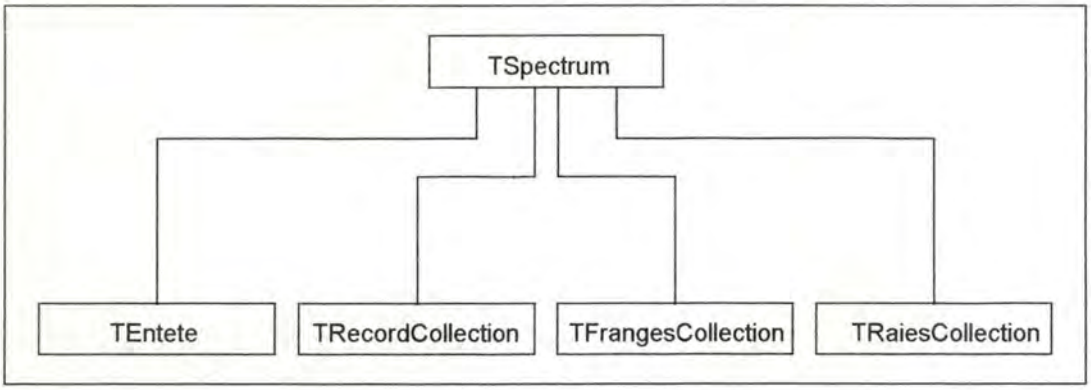


Figure V.5 L'arbre d'appartenance de TSpectrum.

Ces différents objets sont autonomes et savent comment répondre à des ordres; c'est-à-dire à des instructions du programme. Un spectre sait comment il doit se dessiner, de même que les pointés des franges et des raies. Un spectre sait également comment il doit effectuer ses déplacements horizontaux et verticaux ou bien comment effectuer sa rotation en réponse aux sollicitations de l'utilisateur lors de l'étape d'ajustement du fond sur le spectre, ...

Comme on l'a mentionné précédemment, le fait d'adopter une vue "objet" des éléments de l'application va en faciliter sa maintenance. Si on décide un jour de modifier la structure d'un spectre ou bien de l'en-tête, il faudra un minimum de modifications du code (seulement quelques unes des méthodes qui leur sont associées).

Les concepts objets utilisés dans la partie non-interface de ce programme se limitent essentiellement à la notion d'encapsulation afin de conserver les données avec leurs méthodes et à la notion d'héritage qui nous permet d'utiliser intensément la classe d'objet TCollection. TCollection nous est fourni par Turbo Vision et nous offre un moyen très efficace de stockage dynamique; un objet de cette classe permet de stocker un nombre variable d'éléments quelconques simultanément dans une même variable (qu'il s'agisse d'objets divers ou d'entiers, ...).

Chapitre VI

La programmation avec Turbo Vision

1. Introduction

De nos jours, il est devenu inconcevable d'écrire une application informatique ne disposant pas d'une interface utilisateur un temps soit peu évoluée. Toute application se doit d'offrir à ses utilisateurs un moyen simple et efficace de communiquer tel que l'utilisation des menus déroulants pour choisir une action, de la souris pour sélectionner un objet, ... Du point de vue de l'utilisateur, l'utilisation de l'interface ne doit demander qu'un minimum d'effort d'apprentissage, doit être simple d'emploi et doit lui masquer toutes sortes de problèmes internes tels que des contrôles de validité, ... Pour réaliser ces objectifs, le programmeur est amené à passer plus de temps dans le développement de son interface que sur la partie "utile" de son application qui résout le problème pour lequel elle à été écrite.

L'application que nous avons réalisée pour visualiser les spectres d'absorption et les résultats des ajustements de profils ainsi que pour réaliser l'"ajustement" du fond sur le spectre a été écrite à l'aide d'un outil de gestion d'interfaces qui se nomme Turbo Vision. Cet outil est avant tout destiné à la réalisation d'applications en mode texte alors que notre application est essentiellement graphique. Pourquoi dès lors avoir choisi cet outil ? D'une part, notre application est avant tout graphique de par ses résultats et seules quelques touches doivent être gérées en mode graphique. D'autre part, le dialogue entre l'utilisateur et l'application concerne principalement la sélection de fichiers, la sélection d'action à réaliser, des contrôles de validité, ... ; toutes des opérations nécessaires au bon déroulement de l'application mais pouvant très bien être réalisées en mode texte. Turbo Vision nous offrait un moyen de développer rapidement et efficacement notre interface; c'est pourquoi nous l'avons choisi. De plus, Turbo Vision permet de créer des interfaces standards de telle sorte que l'utilisateur puisse immédiatement se sentir à l'aise dans leur manipulation.

2. Présentation de Turbo Vision

Turbo Vision est un outil de gestion du dialogue homme/machine qui facilite le développement d'applications en mode texte sur PC en permettant au programmeur de construire très aisément la partie interface de ses applications. Cet outil consiste en une bibliothèque hiérarchisée d'objets comprenant entre autres :

- des fenêtres multiples, superposables et retaillables à souhait,
- des menus déroulants,
- une gestion du clavier et de la souris,
- des boîtes de dialogue personnalisables,
- des boutons, des barres de défilement, des champs de saisie,
- ...

Turbo Vision offre également un squelette de base pour la réalisation d'applications pilotées par événements; ces applications se manifestant essentiellement par la présence de fenêtres à l'écran. Le programmeur habille alors ce squelette au moyen des caractéristiques d'extensibilité de la programmation orientée objet afin de donner corps à la partie interface de son application. Comme squelette, Turbo Vision propose la classe d'objets TApplication à partir de laquelle le programmeur crée un descendant qu'il enrichit de nouvelles méthodes afin qu'il corresponde aux besoins de son application.

2.a. Les composants d'une application Turbo Vision.

Une application Turbo Vision représente une société dans laquelle coopèrent trois types de composants :

- des éléments visuels,
- des événements,
- des objets muets.

(i) Les éléments visuels (= *elvis* ou *views* en anglais).

Un elvis représente n'importe quel élément d'un programme qui soit visible à l'écran. Citons par exemple les fenêtres, les barres de menus, les barres de

défilement, les champs de saisie, ... qui sont tous des objets de Turbo Vision et qui se matérialisent par un elvis à l'écran. Turbo Vision offre des elvis simples tels que des boutons, des champs de saisie, ... ou des elvis plus complexes tels que des boîtes de dialogue ou des fenêtres qui sont des combinaisons d'elvis simples. Ces combinaisons d'elvis constituent ce que l'on appelle des *groupes* et au sein d'un groupe, tous les elvis collaborent entre eux de manière à ce que le groupe forme un tout.

Les elvis étant des objets, le programmeur peut créer ses propres elvis (simples ou complexes) et ainsi personnaliser les constituants de l'interface de son application.

(ii) Les événements.

Les événements sont des sollicitations auxquelles le programme doit répondre. Ces sollicitations ayant une origine soit externe comme par exemple la frappe d'une touche au clavier ou encore un click du bouton de la souris, soit interne telle que l'envoi d'un message d'un objet du programme vers un autre. Le squelette d'application TApplication (et tous ses descendants) dispose d'un mécanisme qui intercepte tous les événements internes ou externes et qui les stocke dans une file d'attente selon leur ordre d'arrivée. Cette classe dispose également d'un gestionnaire d'événements qui les analyse et les distribue aux elvis appropriés afin que ceux-ci y répondent.

(iii) Les objets muets.

Ce sont tous les objets d'un programme qui ne sont pas des elvis; ils sont dits muets car ils n'interagissent pas avec l'écran. Ces objets permettent généralement de réaliser le contenu d'une application; c'est-à-dire des calculs, des échanges d'informations avec les périphériques, ... Parmi ces objets, on retrouve notamment les collections qui peuvent être assimilées à des tableaux de tailles variables contenant des éléments quelconques.

2.b. La structure d'une application Turbo Vision.

Le code principal de n'importe quelle application Turbo Vision ne comporte que trois instructions : une initialisation, une exécution et une destruction de l'application. Tout le code réalisant le contenu de l'application est défini à l'intérieur

des méthodes des objets utilisés; toute application contenant obligatoirement un objet d'une classe descendante de TApplication.

(i) **L'initialisation** (méthode TApplication.Init).

La majorité des méthodes des objets définis dans Turbo Vision sont virtuelles et il est donc indispensable de créer leur table des méthodes virtuelles par l'intermédiaire de leur constructeur (*cfr.* chapitre 5). La méthode d'initialisation de l'application a justement pour effet d'appeler les constructeurs des classes de base de Turbo Vision. Au cours de l'initialisation, l'écran est effacé et un ensemble de variables (cachées au programmeur) initialisées; le bureau, la barre des menus ainsi que la barre des statuts sont également affichés à l'écran par cette méthode.

(ii) **L'exécution** (méthode TApplication.Run).

C'est cette méthode qui va rester active pendant la majeure partie de l'exécution de l'application; on peut schématiquement la représenter par une boucle *repeat ... until* telle que :

repeat

Saisie d'un événement

Traitement de l'événement

until Quitter

Une application Turbo Vision effectue donc principalement deux tâches : prendre un événement (éventuellement l'événement "*rien*" = *nothing*) et le traiter. Parmi l'ensemble des événements susceptibles de se produire, il doit obligatoirement en exister un qui est une demande de sortie de l'application; ce qui permet de sortir de la boucle précédente.

(iii) **La destruction** (méthode TApplication.Done).

Cette méthode consiste à terminer l'application proprement. Lorsque la fin d'une application est demandée, les destructeurs de tous les elvis ainsi que celui de l'application sont appelés; ce qui provoque la destruction de tous les objets de l'application. Au cours de cette phase, le gestionnaire d'événements est également désactivé.

3. Les éléments visuels

Avec Turbo Vision, le programmeur ne doit plus (et ne peut plus) effectuer d'écritures directement à l'écran mais il doit fournir à Turbo Vision les informations qu'il veut afficher; celui-ci s'assure alors de placer les informations où il faut au bon moment. La brique de base de toute application Turbo Vision est l'elvis qui est un objet capable de gérer une zone rectangulaire de l'écran; ainsi toute action du programme survenant dans une zone de l'écran sera entièrement prise en charge par l'elvis la recouvrant.

3.a. Les propriétés des elvis.

Tous les elvis sont des descendants de la classe d'objets TView qui contient les méthodes et les données nécessaires à une gestion de base de l'écran. Tous ces elvis doivent impérativement posséder les deux propriétés suivantes :

- Ils doivent être capables de s'afficher eux-mêmes à n'importe quel moment. Cette propriété est très importante car un elvis doit être capable de réapparaître à l'écran lorsqu'un elvis qui le recouvrait disparaît. Par exemple, lorsqu'une boîte de dialogue est refermée, il est nécessaire que l'écran reprenne la forme qu'il possédait avant l'ouverture de cette boîte.

- Ils doivent savoir comment réagir aux événements les concernant et qui leur sont envoyés par le gestionnaire d'événements (cela est explicité dans la section suivante concernant la programmation événementielle). A titre d'exemple, une boîte de dialogue doit être capable de réagir d'elle même lorsque l'utilisateur clique sur son bouton *Ok* à l'aide de la souris.

Chaque elvis possède une méthode particulière nommée Draw qui détermine son aspect lors de son affichage. La particularité de cette méthode est qu'elle doit couvrir entièrement la zone écran associée à l'elvis. En effet si ce n'était pas le cas, des zones d'affichage auraient un contenu indéterminé et des choses bizarres apparaîtraient à l'écran.

Ces méthodes Draw sont constamment appelées par Turbo Vision afin de maintenir à jour les informations affichées à l'écran. Dans une application classique, lorsqu'une fenêtre vient recouvrir une zone de l'écran, il suffit de sauvegarder le contenu de cette zone avant le recouvrement pour pouvoir la restituer plus tard. Par

contre dans une application Turbo Vision c'est tout différent car même si un elvis n'est pas visible actuellement à l'écran, un événement à tout de même pu lui parvenir et en modifier son aspect. En appelant les méthodes Draw à chaque fois qu'un elvis disparaît ou a été déplacé, on est ainsi certain que les informations qui apparaissent à l'écran sont à jour.

3.b. Les groupes.

Alors que les elvis étaient tous des descendants de la classe TView, tous les groupes sont des descendants de la classe TGroup; TGroup étant même un descendant immédiat de TView. TGroup hérite donc de toutes les données et méthodes de TView et fournit des méthodes supplémentaires de gestion de listes d'elvis.

Un groupe est une boîte qui contient et gère un ensemble d'elvis. D'un point de vue technique, un groupe n'est rien d'autre qu'un elvis; un elvis complexe mais un elvis tout de même. Par conséquent, un groupe doit être en mesure d'effectuer les opérations relatives aux elvis; à savoir :

- gérer une zone rectangulaire de l'écran,
- être capable de s'afficher à tout moment,
- prendre en charge les événements le concernant.

La différence entre un elvis et un groupe se situe dans la manière dont ces opérations sont réalisées. Dans le cas d'un elvis, c'est l'elvis lui même qui réalise ces opérations; tandis que dans le cas d'un groupe, celui-ci délègue la majorité de ces opérations à ses elvis composants. A titre d'exemple, lorsqu'un groupe doit être affiché, il l'est par l'intermédiaire de chacune des méthodes Draw de ses elvis.

Afin qu'un elvis fasse partie d'un groupe, il doit y être inséré; cette opération d'insertion étant réalisée dans un certain ordre qui détermine à la fois l'ordre d'affichage des elvis à l'écran et l'ordre dans lequel les événements leur seront transmis. Cet ordre porte le nom d'ordre Z par analogie aux axes associés aux 3 dimensions de l'espace; X et Y représentant la surface de l'écran, Z représentant la profondeur. L'ordre Z représente donc l'ordre dans lequel les différents elvis d'un groupe sont rencontrés en partant de l'utilisateur et en descendant dans les profondeurs de l'écran. Le dernier elvis inséré sera toujours celui qui se trouve le plus proche de l'utilisateur.

Avec Turbo Vision, l'écran n'est plus vu comme étant une surface plane mais comme la superposition de plaques de verre transparentes; chacune représentant un elvis inséré dans le groupe. Ce que l'utilisateur voit est obtenu en superposant chacune de ces plaques dans leur ordre d'insertion dans le groupe. La dernière plaque insérée étant la plus proche de l'utilisateur.

3.c. Les elvis de type modal.

Dans les concepts de Turbo Vision, un mode correspond à une manière d'agir ou d'opérer et chaque elvis peur définir son mode opératoire. Un elvis disposant de son propre mode opératoire est dit *elvis modal*. L'exemple le plus courant d'elvis modal est la boîte de dialogue qui interdit toute action en dehors de sa zone tant qu'elle est activée. Lorsqu'un elvis modal est activé, seul cet elvis et tous les elvis qu'il contient sont susceptibles de répondre aux événements; tous les autre elvis ne sont plus temporairement concernés par ces événements.

Il faut noter que toute application Turbo Vision contient au moins un elvis modal; il s'agit de la représentation de l'objet descendant de TApplication qui est obligatoire dans toute application Turbo Vision. Cette représentation contient au moins les trois elvis suivants : la barre des menus, le bureau et la barre des statuts.

4. La programmation événementielle

Les applications écrites à l'aide de Turbo Vision sont des applications pilotées par événements qui se distinguent des applications classiques par une séparation des structures de contrôle des interactions entre l'utilisateur et le programme d'une part et les procédures et fonctions réalisant les traitements des données d'autre part. A cause de cette séparation, le code des applications Turbo Vision est quelque peu différent de celui des application traditionnelles.

Tout programme interactif doit évidemment travailler en analysant ce que l'utilisateur lui fournit en entrée. Dans le cas d'une application traditionnelle, c'est au programmeur que revient la tâche d'aller collecter les entrées de l'utilisateur, de les analyser et ensuite de décider à quelle fonction ou procédure passer la main; ces fonctions ou procédures disposant aussi éventuellement d'un nouveau cycle de saisie, analyse et routage.

Dans des applications événementielles, le programmeur n'a plus à se soucier de la lecture des entrées et de décider ce qu'il doit en faire. En effet, ces applications

disposent d'un mécanisme de distribution d'événements qui les dispense de la gestion du dialogue utilisateur; c'est Turbo Vision qui s'en charge. L'application attend simplement que le distributeur d'événements fournisse à chacune des procédures les messages auxquels elles sont susceptibles de répondre. Le travail du programmeur consiste alors uniquement à prendre une décision en fonction de ce qui reçu et l'attente ainsi que la réception des entrées utilisateurs ne sont plus à sa charge.

4.a. La nature et le type des événements.

On peut imaginer un événement comme étant un petit paquet d'information décrivant une sollicitation à laquelle l'application doit réagir. Les événements sont des entités atomiques; c'est-à-dire qu'un mot frappé au clavier ne constitue pas un événement unique mais une série d'événements individuels. Parmi les informations véhiculées par un événement on retrouve son type et en fonction de celui-ci le code de la touche frappée ou la position de la souris, ...

Turbo Vision classe les événements en quatre catégories :

1. Les événements relatifs à la souris.

2. Les événements relatifs au clavier.

3. Les événements "message".

4. Les événements neutralisés.

Les deux premières catégories se passent de tout commentaire. Les événements "message" correspondent à ce que l'on appelle des commandes et des événements diffus.

Les commandes sont des événements qui sont généralement ciblés (*i.e.* qu'ils sont destinés à un elvis bien particulier) et qui se traduisent par une action significative du programme. Ces commandes ne sont pas générées directement par l'utilisateur mais par l'intermédiaire d'événements souris ou clavier. Par exemple lorsque l'utilisateur effectue un choix dans un menu, un événement souris ou clavier est généré et transmis à l'objet menu qui génère lui-même un nouvel événement qui est une commande destinée à un autre objet (par exemple à une boîte de dialogue pour lui dire de s'ouvrir).

Les événements diffus sont des événements qui ne connaissent pas a priori leur destinataire et qui sont envoyés à tous les elvis d'un groupe. Par exemple, lorsque l'utilisateur clique sur la flèche d'ascenseur d'une fenêtre, la position de l'ascenseur doit être modifiée, le contenu de la fenêtre également, et peut être d'autres choses encore. Ces événements servent généralement aux communications entre les elvis.

Un événement neutralisé est un événement passé ayant été pris complètement en compte par un objet. Ce type d'événement ne contient plus aucune information utile pour quelque objet que ce soit et il est ignoré.

4.b. La gestion des événements.

Un des grands avantages de la programmation événementielle est que le code n'a pas à se soucier d'où proviennent les événements mais seulement de savoir comment y répondre. Ainsi le concept de programmation événementielle grâce auquel les réceptions et réponses aux événements sont séparées se combine à merveille avec la notion d'objet. Les objets ne sont pas des éléments passifs mais sont en quelque sorte des "acteurs" qui attendent qu'on les sollicite pour réciter leur texte; c'est-à-dire leurs méthodes. Les objets n'ont que faire de la lecture des données de l'utilisateur; leur seule préoccupation est de savoir quand et comment ils doivent agir. *Quand* un objet doit entrer en scène est dicté par les événements qu'il reçoit; *comment* il doit agir est implémenté dans ses méthodes. Tout objet visuel de Turbo Vision ainsi que ses descendants dispose d'une méthode particulière, HandleEvent, qui analyse l'événement qu'il reçoit et lui indique comment se comporter (généralement par l'exécution d'une ou plusieurs de ses méthodes).

Chaque elvis de Turbo Vision sait déjà comment il doit répondre à la plupart des événements qu'il reçoit. Un menu sait comment s'ouvrir, interagir avec l'utilisateur et se refermer. Les fenêtres savent comment s'ouvrir, se fermer, se retailler. Les barres de défilement savent comment répondre à une action sur leurs flèches de défilement, ...

Si un elvis ne se comporte pas comme le programmeur le souhaite et qu'il désire lui affecter des comportements spécifiques à son application, il lui suffit d'en dériver une classe descendante dans laquelle il réécrit la méthode d'analyse des événements. Pour que le nouvel elvis ainsi dérivé réagisse à certains événements différemment de son ancêtre immédiat, il doit tout d'abord intercepter ces événements avant de les transmettre à la méthode HandleEvent de son ancêtre. De

cette manière, le programmeur n'a qu'à écrire ce qui change et pas toute la méthode d'analyse. De la même manière si le nouvel elvis doit posséder des comportements supplémentaires (*i.e.* répondre à de nouveaux événements), il doit tout d'abord appeler la méthode HandleEvent de son ancêtre et ensuite analyser ce qui est nouveau.

Lorsqu'une méthode HandleEvent d'un elvis a complètement pris en charge un événement, elle fait appel à la méthode ClearEvent de son elvis qui a pour effet de neutraliser l'événement; ce qui signifie qu'il a été pris en compte et que s'il est passé à un autre elvis il doit être ignoré.

5. Applications

Nous avons donc réalisé l'interface de notre application à l'aide des objets de Turbo Vision. Les différentes opérations susceptibles d'être réalisées par l'application peuvent être déclenchées par sélection dans les menus ou bien éventuellement pour certaines d'entre elles par des raccourcis clavier (figure VI.1 et VI.2).

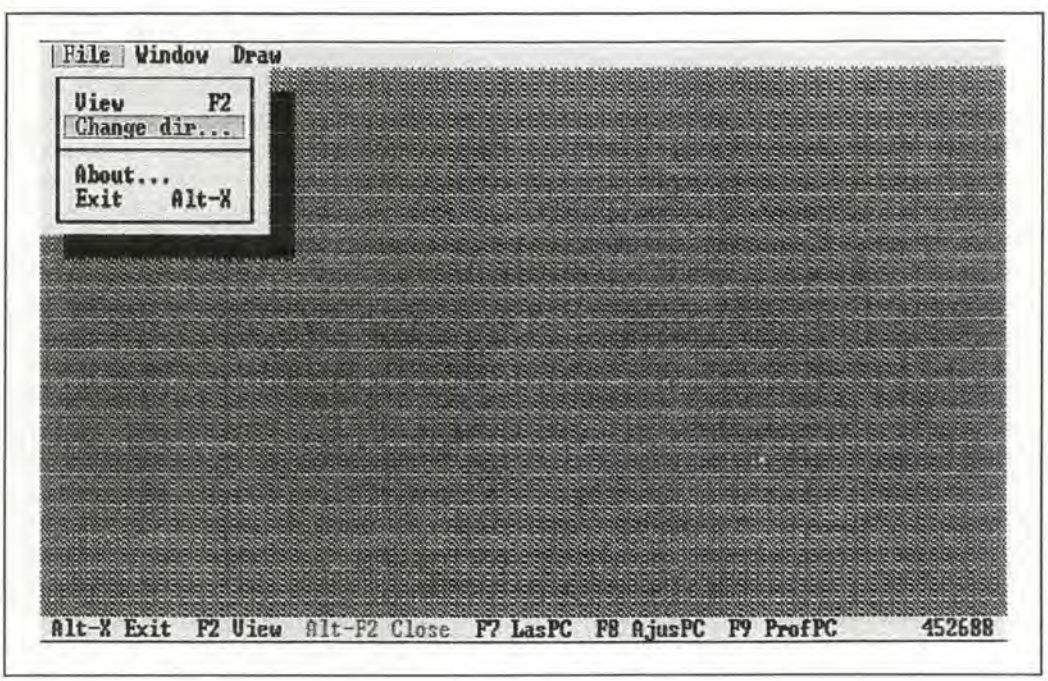


Figure VI.1.

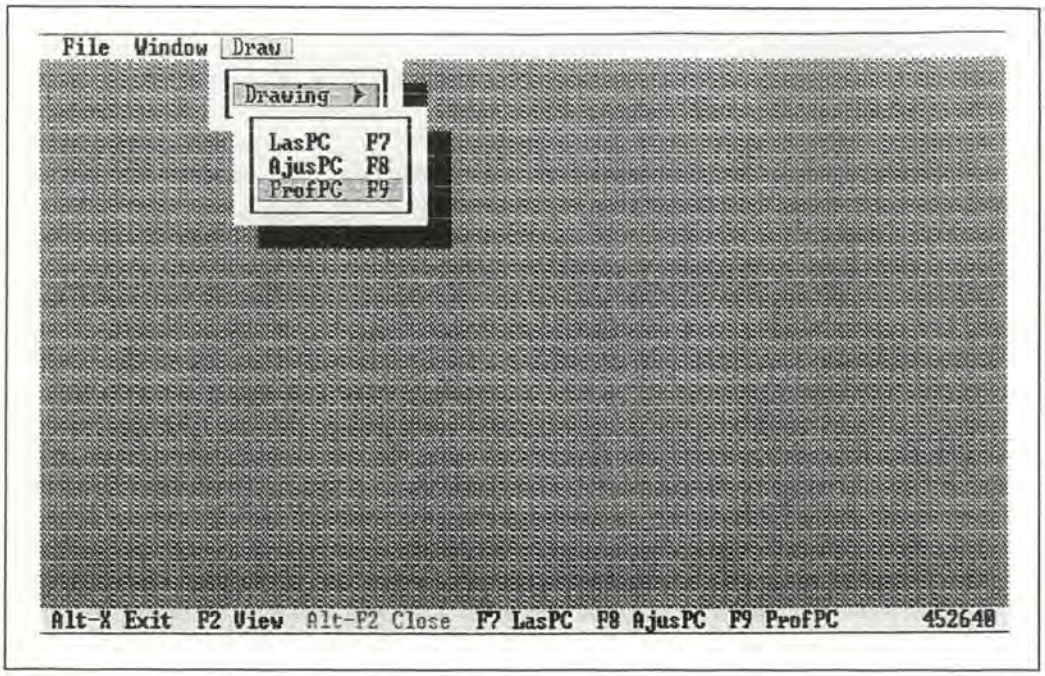


Figure VI.2.

Turbo Vision nous a offerts des objets très utiles tels que des boîtes de dialogue de sélection de fichiers, des boîtes de dialogue pour changer le répertoire courant sans sortir de l'application, ... (figures VI.3 et VI.4).

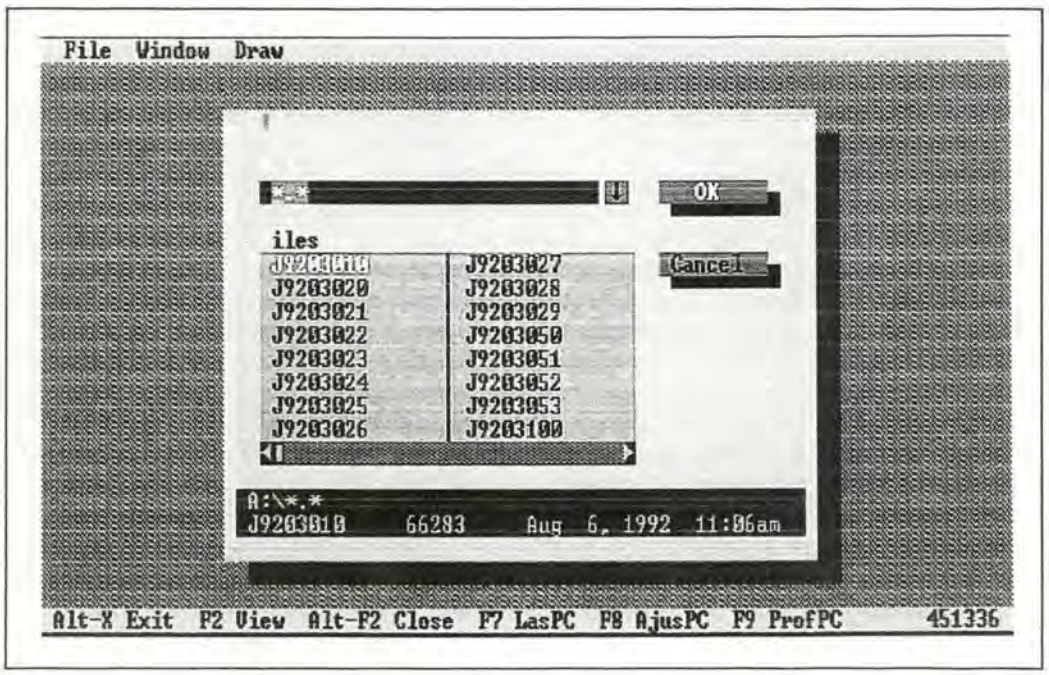


Figure VI.3

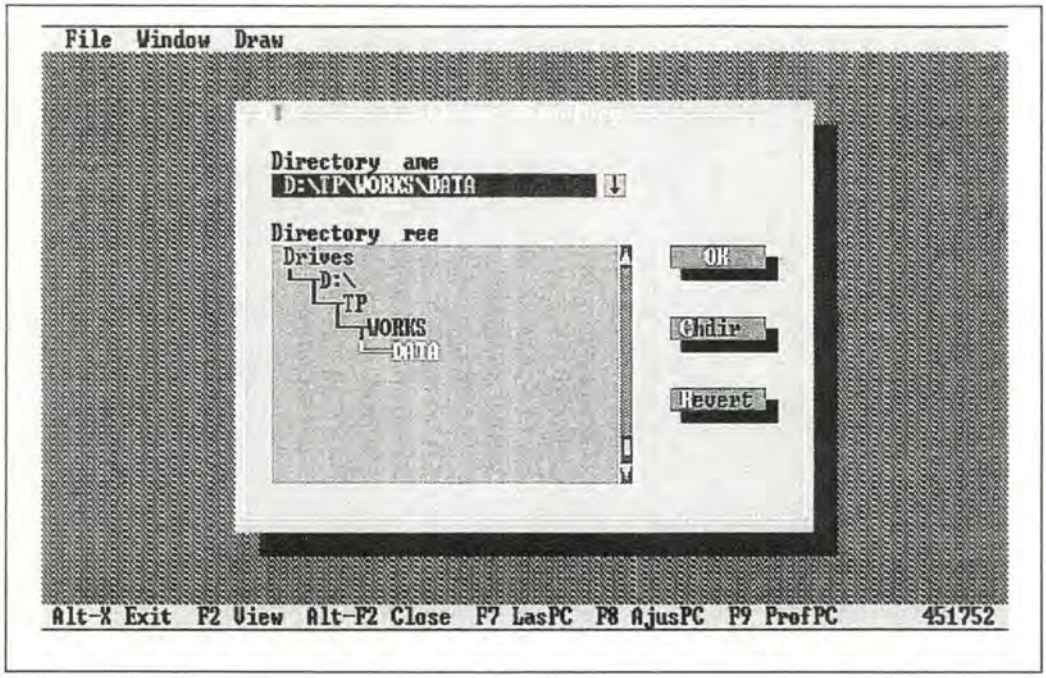


Figure VI.4

Nous avons également implémenté des fenêtres de visualisation du contenu (sous forme textuelle) de fichier. Il est possible d'ouvrir autant de fenêtres qu'on le désire et de les superposer, de modifier leur taille, ...; pour autant que la place mémoire soit suffisante (figure VI.5).

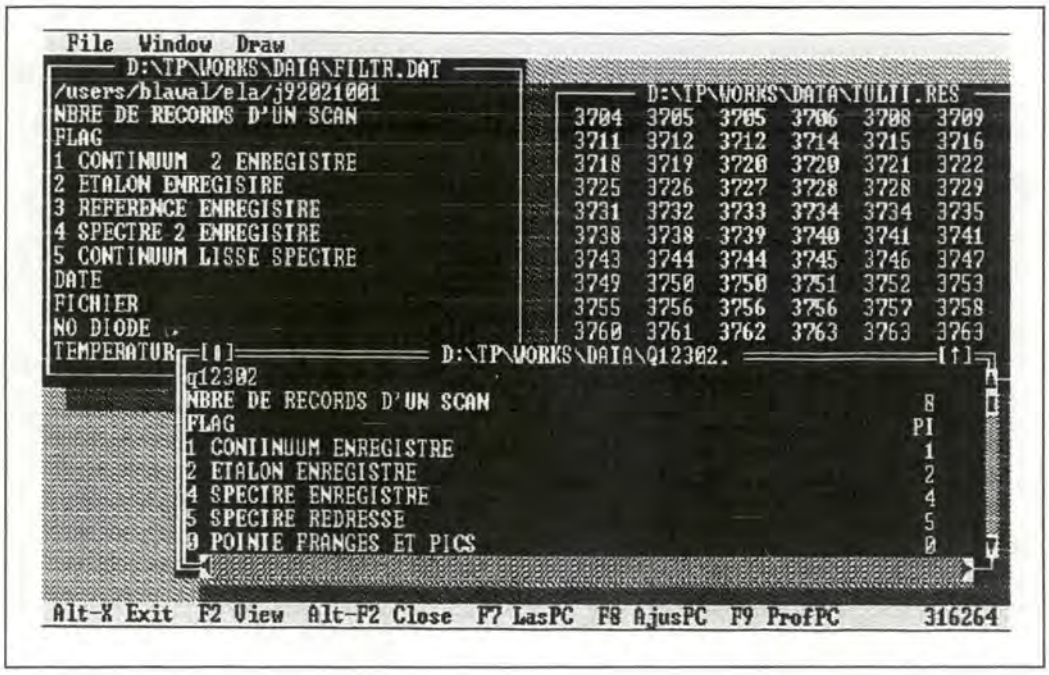


Figure VI.5

Turbo Vision offre aussi la possibilité de renseigner à l'utilisateur un manquement de la ressource mémoire lors d'allocation mémoire. Nous avons également enrichi la routine de gestion standard des erreurs d'entrées/sorties du Turbo Pascal par l'affichage de l'erreur à l'écran sous forme d'un texte dans une boîte de dialogue (figure VI.6); le fait d'intercepter ces erreurs permet à l'application de se poursuivre et non plus de générer un arrêt fatal avec l'erreur sous forme d'un nombre.

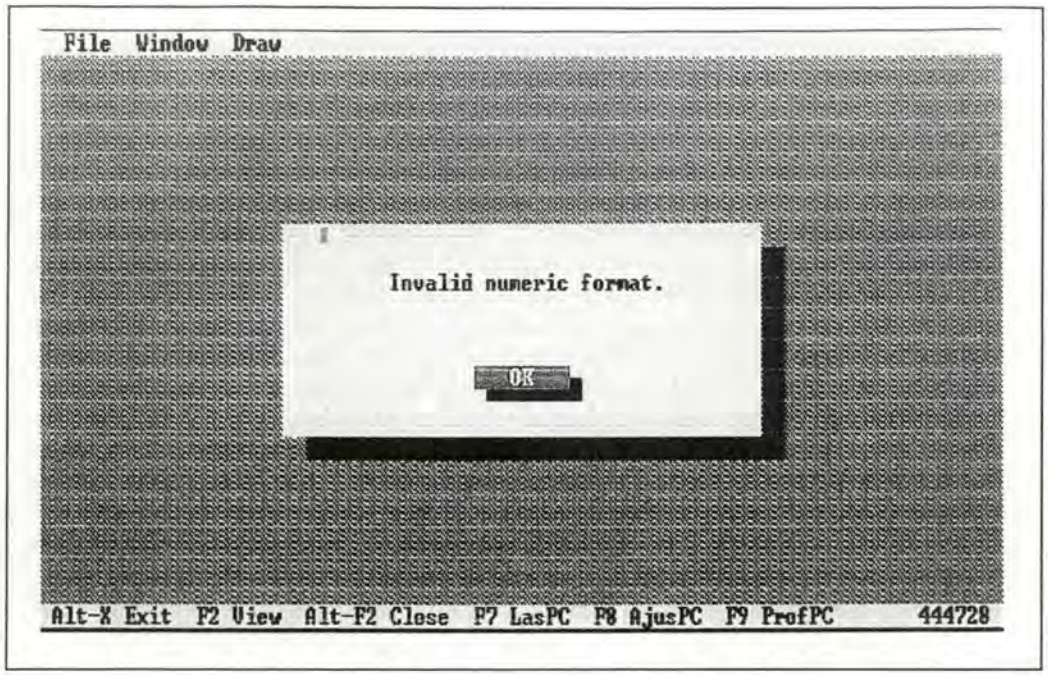


Figure VI.6

Bien que Turbo Vision soit un outil de travail en mode texte, il est tout de même possible d'utiliser son gestionnaire d'événements en mode graphique lorsqu'il s'agit d'intercepter les touches frappées au clavier par l'utilisateur. Etant donné que les actions possibles dans notre application, lorsque ce mode est activé, se limitent justement à des réponses à des frappes de touches (ne correspondant pas à l'affichage de messages), nous avons pu utiliser ce gestionnaire aussi bien en mode texte qu'en mode graphique.

Conclusion

Nous estimons avoir raisonnablement atteint les buts que nous nous étions fixés dans la réalisation de ce travail. Notre premier objectif concernait le développement d'un système d'analyse de spectres d'absorption moléculaire réalisant l'ajustement de profils d'absorption sur des raies isolées ou multiples afin d'en extraire un maximum d'informations.

Pour des raies isolées, le système développé est capable de déterminer parmi un ensemble de modèles de profils celui qui est le mieux adapté et il nous fournit également les valeurs des paramètres ajustables de ce modèle. Appliqué au cas des raies multiples, le système détermine deux paramètres pour chacune des raies ainsi qu'un paramètre global de déplacement en fréquence pour l'ensemble des raies étudiées. Les résultats ainsi obtenus sont plus que satisfaisants et nous ont notamment permis de résoudre le problème crucial de la non-résolution des raies lorsqu'elles se chevauchent. Ces ajustements nous ont en effet permis d'extraire les paramètres des raies individuelles et d'en reconstruire leur profil. Qu'il s'agisse de l'étude de profils sur des raies simples ou multiples, nous avons pu remarquablement reconstituer les spectres d'absorption étudiés et cela avec une excellente précision.

Les ajustements sont réalisés à l'aide de l'algorithme de Levenberg-Marquardt qui est un compromis entre deux méthodes de recherche de l'ajustement optimal au sens des moindres carrés non-linéaires : la méthode du gradient et la méthode de la matrice Hessienne. Appliqué à l'ajustement de profils sur des raies isolées, cet algorithme itératif converge relativement vite. Par contre lorsqu'il s'agit d'étudier des recouvrements de raies, le temps nécessaire à la détermination des valeurs des paramètres devient beaucoup plus long. Le critère de rapidité n'étant pas notre souci majeur, nous n'y avons pas attaché d'importance et nous avons préféré axer notre développement sur l'obtention de résultats corrects endéans des temps calcul raisonnables. Nous avons ainsi par exemple pu étudier des spectres contenant six raies d'absorption et déterminer les valeurs des 13 paramètres associés de manière très satisfaisante.

Comme toutes les méthodes de recherche d'ajustement optimal de modèles non-linéaires, la méthode de Levenberg-Marquardt est fortement sensible aux choix

des valeurs approchées des paramètres. Nous avons ainsi pu constater qu'un choix réfléchi de ces valeurs était nécessaire afin d'obtenir des résultats significatifs.

En ce qui concerne le développement de l'environnement graphique sur PC, nous estimons que les améliorations apportées à l'"ajustement" visuel entre le fond et le spectre permettent une meilleure détermination du profil et par conséquent des paramètres qui lui sont associés. D'autre part, nous estimons que le fait d'y avoir utilisé la programmation orientée objets facilitera sa maintenance future.

Cette application intègre plusieurs traitements ("ajustement", visualisation des spectres, visualisation des résultats des ajustements de profils) et offre un environnement de travail simple, efficace et facile à utiliser. C'est une première étape dans la réalisation d'un atelier logiciel complet de traitement des spectres d'absorption qui devrait inclure dans le futur l'ensemble des traitements réalisés sur PC.

Enfin, en plus des avantages que son utilisation nous a procurés, l'étude de Turbo Vision avec ses méthodes de programmation événementielle nous apporte une manière d'appréhender les problèmes qui nous sera très utile pour le développement d'applications sous Windows.

Bibliographie

1. K. Narahari Rao, "Molecular Spectroscopy : Modern Research (Volume II)", Academic Press, New-York (1976).

2. W.H. Press, B.P. Flannery, S.A. Teukolsky, W.Y. Vetterling, "Numerical Recipes : The Art of Scientific Computing", Cambridge University Press, Cambridge (1992).

3. G. Baldacchini, "LINEFIT : A Digital Acquisition and Fitting Program for Molecular Spectroscopy with Tunable Diode Lasers", Report prepared by Servizio Studi e Documentazione-ENEA, centro (1991).

4. J.J. More, "The Levenberg-Marquardt Algorithm : Implementation and Theory, Numerical Analysis", Lect. Notes Math. 630, 106 (1977).

5. P.R. Bevington, "Data Reduction and Error Analysis for the Physical Sciences", McGraw-Hill Book Company (1969).

6. D. Cope, R. Khoury, R.J. Lovett, "Efficient Calculation of General Voigt Profiles", J. Quant. Spectrosc. Radiat. Transfer, Vol. 39, No. 2, pp. 163-171 (1988).

7. J.-P. Bouanich (Université Orsay), Communication Privée, (Septembre 1991).

8. A.K. Hui, B.H. Armstrong, A.A. Wray, "Rapid Computation of the Voigt and Complex Error Functions", J. Quant. Spectrosc. Radiat. Transfer, Vol. 19, pp. 509-516 (1978).

9. J.H. Pierluissi, P.C. Vanderwood, R.B. Gomez, "Fast Calculational Algorithm for the Voigt Profile", J. Quant. Spectrosc. Radiat. Transfer, Vol. 18, pp. 555-558 (1977).

10. S.R. Drayson, "Rapid Computation of the Voigt Profile", J. Quant. Spectrosc. Radiat. Transfer, Vol. 16, pp. 611-614 (1976).

11. J. Humlicek, "Optimized Computation of the Voigt and Complex Probability Functions", J. Quant. Spectrosc. Radiat. Transfer, Vol. 27, No. 4, pp. 437-444 (1982).

12. P.L. Varghese, R.K. Hanson, "Collisionnal Narrowing Effects on Spectral Line Shapes Measured at High Resolution", App. Opt., Vol. 23, No. 14, pp. 2376-2385 (1984).

13. G. Blanquet, J. Walrand, "Using a Hewlett-Packard Minicomputer for the Processing of Tunable Diode Laser Spectra", Computer Enhanced Spectroscopy, Vol. 2, No. 4, pp. 135-140 (1984).

14. X. Ouyang, P.L. Varghese, "Reliable and Efficient Program for Fitting Galatry and Voigt Profiles to Spectral Data on Multiple Lines", App. Opt., Vol. 28, No. 8, pp. 1538-1545 (1989).

15. L. Rosenmann, Thèse de doctorat, Ecole centrale de Paris (1988).

16. G. Blanquet, J. Walrand, "Procédures d'utilisation du système d'acquisition couplé au spectromètre infrarouge à haute résolution", FUNDP (1982).

17. International Mathematical and Statistical Library, Houston, Texas.

18. R. Rundel, "PeakFit, a Non-Linear Curve-Fitting Software : Technical Guide" (1990).

19. Turbo Pascal, "User's Guide" & "Turbo Vision Guide" (1990).

Annexes

Annexe 1

Cette annexe reprend les différents fichiers sources (en Turbo Pascal 6.0) de l'application sur PC. Le programme principal se trouve dans le fichier Memoire.pas qui utilise le fichier "include" MemUtil.pas et les fichiers "units" MemUnit.pas, TVUtil.pas, Utility.pas et Mouse.pas.

Fichier Programme Memoire.pas

```
{$X+}  { Extended Syntax enable }
{$G+}  { 286-Instructions }
{$M 10000, 0, 655360}
program Memoire;

uses
{ Units Turbo Vision }
  Objects, Drivers, Menus,  Views, Memory, Dialogs, App,
  StdDlg, MsgBox, Gadgets, GraphApp,
{ Units Pascal Standard }
  Dos, Crt, Graph,
{ Units personnelles }
  Utility, TVUtil, MemUnit, Mouse;


{ Définition des constantes de commandes non prédéfinies }
{ auxquelles la méthode HandleEvent doit répondre.       }

const
  cmChangeDir   = 100;   { Changer de répertoire        }
  cmViewFile    = 101;   { Voir le contenu d'un fichier }
  cmAbout       = 102;   { Afficher la version          }
  cmDraw        = 103;   { Choisir l'application        }
  cmLasPC       = 104;   { Dessin des spectres          }
  cmAjusPC      = 105;   { Ajustement du fond           }
  cmProfPC      = 106;   { Visualisation des fits       }

type

{ TMyApp définit l'application. C'est une application }
{ standard pour laquelle on a simplement réécrit le   }
{ gestionnaire d'événements et les barres de menu et  }
{ de status.                                          }

  PMyApp = ^TMyApp;
  TMyApp = object(TStandardApplication)
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure InitStatusLine; virtual;
    procedure InitMenuBar; virtual;
  end;

var
  MyApp : TMyApp;
```

```
{$I MemUtil.pas}

{ TMyApp }

{ TMyApp.HandleEvent définit comment l'application doit répondre aux  }
{ événements. L'application est uniquement sensible aux événements    }
{ issus  d'une sélection dans le menu ou dans la Statusline           }

procedure TMyApp.HandleEvent(var Event: TEvent);

   { ChangeDir ouvre une boite de dialogue qui permet de changer }
   { le répertoire courant.                                      }

   procedure ChangeDir;
   var
     D : PChDirDialog;
   begin
     D := PChDirDialog(ValidView(New(PChDirDialog, Init(0,
          cmChangeDir))));
     if D <> nil then
     begin
       DeskTop^.ExecView(D);
       Dispose(D, Done);
     end;
   end;

   { ViewFile ouvre une boite de dialogue qui permet la sélection    }
   { d'un fichier. Si un fichier a été sélectionn,, son contenu est  }
   { affiché dans une fenêtre et l'utilisateur peut le visualiser en }
   { utilisant les touches de flèches, les touches PgUp et PgDn, ... }
   { ainsi qu'en utilisant les barres de défilements de la fenêtre.  }
   { Plusieurs fenêtres peuvent être ouvertes simultanément.         }

   procedure ViewFile;
   var
     D : PFileDialog;
     F : PFileWindow;
     FileName : PathStr;
   begin
     D := PFileDialog(ValidView(New(PFileDialog, Init('*.*', 'Fichier à
          visualiser','~F~ile name', fdOkButton, 100))));
     if D <> nil then
     begin
       if DeskTop^.ExecView(D) <> cmCancel then
       begin
         D^.GetFileName(FileName);
         F := PFileWindow(ValidView(New(PFileWindow, Init(FileName))));
         if F <> nil then DeskTop^.Insert(F);
       end;
       Dispose(D, Done);
     end;
   end;

   { Tile réorganise l'affichage des différentes fenêtres ouvertes en }
   { leur attribuant à chacune une portion de l'espace de travail.    }

   procedure Tile;
   var
     R : TRect;
   begin
     DeskTop^.GetExtent(R);
     DeskTop^.Tile(R);
   end;
```

```
{ Cascade réorganise l'affichage des différentes fenêtres ouvertes }
{ en les superposant.                                              }

procedure Cascade;
var
  R : TRect;
begin
  DeskTop^.GetExtent(R);
  DeskTop^.Cascade(R);
end;

{ About affiche la version de l'application. }

procedure About;
var
  R : TRect;
begin
  R.Assign(0,0,30,8);
  R.Move((DeskTop^.Size.X-R.B.X) div 2, (DeskTop^.Size.Y-R.B.Y)
         div 2);
  MessageBoxRect(R, #3'SPECTRUM'#13#13#3'Version 1.0', nil,
                 mfInformation)
end;

{ Dessine les spectres enregitrés dans le fichier ainsi que les }
{ pointés ,ventuels. LasPC permet également de désigner un      }
{ intervalle sur le spectre à l'aide de la souris.              }

procedure LasPC;
const
  Color : array[1..5] of Word = (Blue, Red, Green, Yellow, Magenta);
var
  FirstPoint, SecondPoint : String;
  I : Integer;
  E : TEvent;
  Spectrum : PSpectrum;
  SIn, SOut : SystemCoord;
  FileName : PathStr;
  P1, P2 : Coord;
  IsPointe : Boolean;

{ Convers convertit une position sur l'écran en une position }
{ en point fichier.                                          }

procedure Convers(var C : Coord);
var
  X : Real;
begin
  X := (C.X - 127) / SOut.CID.X * SIn.CSG.X ;
  C.X := Trunc(X);
end;

begin
  Spectrum := ReadFile(FileName);
  if Spectrum <> nil then begin
    if RegisterBGIDriver(@EgaVgaDriver) < 0
      then MessageBox(#3'Impossible de charger le driver graphique
                      EGA/VGA.', nil, mfError or mfOkButton)
      else begin
        if GraphicsStart then begin
          { Affichage des carctéristiques du spectre }
          DrawInitialisation(Spectrum^.GetEntete, tyLasPC,
           Spectrum^.GetNofRecord, (Spectrum^.GetFranges <> nil));
          { Définition des systèmes de coordonnées }
```

```
          With SIn do begin
            CSG.X := PRecord(Spectrum^.GetData^.At(0))^.Count;
            CSG.Y := 0; CID.X := 0; CID.Y := 4096;
          end;
          With SOut do begin
            CSG.X := 0; CSG.Y := 0; CID.X := GetMaxX-128;
            CID.Y := GetMaxY-21;
          end;
          IsPointe := False;
          repeat
            ClearEvent(E);
            GetKeyEvent(E);
            if (E.What <> evNothing) then
              if (E.ScanCode > $01) and
                 (E.ScanCode-1 <= Spectrum^.GetNofRecord) then
                              { Dessin d'un spectre }
                 PRecord(Spectrum^.GetData^.At(E.ScanCode-2))^.
                    Draw(SIn, Sout, Color[E.ScanCode-1], True);
              if (E.ScanCode = $0A) then begin
                              { Dessin du pointé }
                if Spectrum^.GetFranges <> nil then
                    Spectrum^.GetFranges^.Draw(SIn, SOut, Red);
                if Spectrum^.GetRaies <> nil then
                    Spectrum^.GetRaies^.Draw(SIn, SOut, Yellow);
              end;
              if (E.ScanCode = $44) then begin
                { Sélection d'un intervalle }
                if InitMouse then begin
                  P1.X := 128; P2.X := GetMaxX-2;
                  P1.Y := 21; P2.Y := GetMaxY-2;
                  SetMouseViewPort(P1, P2);
                  DisplayMouse;
                  Repeat until IsLeftButtonClick(P1);
                  Repeat until IsLeftButtonClick(P2);
                  Convers(P1);
                  Convers(P2);
                  ClearMouse;
                  IsPointe := True;
                end;
              end;
            until (E.What <> evNothing) and (E.ScanCode = $01);
            GraphicsStop;
          end;
          GraphAppDone;
          { Si un intervalle a été sélectionné, on l'affiche }
          if IsPointe then begin
            Str(P1.X, FirstPoint); Str(P2.X, SecondPoint);
            MessageBox(#3'L''intervalle s,lectionn, est :' + #13#13#3 +
                       '[' +FirstPoint + ',' + SecondPoint + ']' ,
                       nil, mfInformation + mfOkButton)
          end;
        end;
      Dispose(Spectrum, Done);
    end;
end;

{ AjusPC permet d'ajuster le fond sur le spectre. Les opérations }
{ disponibles sont affichées à l'écran.                          }

procedure AjusPC;
const
  { Touches actives }
  ActiveKey = [$3B, $3C, $3D, $3E, $44, $47, $48, $49, $4B, $4D, $4F,
               $50, $51];
```

```
  { Touches nécessitant un nouveau tracé du fond }
  RedrawFondOnly = [$47, $48, $4B, $4D, $4F, $50];
  { Touches nécessitant un nouveau tracé du fond et du spectre }
  RedrawAll = [$3B, $3C, $49, $51];
  { Pas du déplacement vertical }
  VStep = 1;
  { Pas du déplacement horizontal }
  HStep = 1;
  { Pas de la rotation }
  RAStep = 0.000001;
var
  V : PInteger;
  V1, V2 : PReal;
  I, Magn : Integer;
  Spectrum : PSpectrum;
  SIn, SOut,
  SInRed, SOutRed,
  SInDiff, SOutDiff : SystemCoord;
  RR, R : PRecord;
  E : TEvent;
  C, D : Coord;
  MagnStr : String;
  FileName : PathStr;
  DiffActive,
  SpectreRed : Boolean;
begin
  R := nil; RR := nil; new(V1); new(V2);
  Spectrum := ReadFile(FileName);
  if Spectrum <> nil then begin
    if (Spectrum^.GetFlag = 'CI') or (Spectrum^.GetFlag = 'CE') or
       (Spectrum^.GetFlag = 'CF') then begin
      if RegisterBGIDriver(@EgaVgaDriver) < 0
       then MessageBox(#3'Impossible de charger le driver graphique
                        EGA/VGA.', nil, mfError or mfOkButton)
       else begin
         if GraphicsStart then begin
           { Préparation de l'écran }
           DrawInitialisation(Spectrum^.GetEntete, tyAjusPC,
              Spectrum^.GetNofRecord, (Spectrum^.GetFranges <> nil));
           { Par défaut, le spectre redressé et la différence }
           { ne sont pas affichés                             }
           SpectreRed := False;
           DiffActive := False;
           { L'amplitude des déplacements est par défaut d'un pas }
           Magn := 1;
           { Affichage de l'amplitude de déplacement }
           Str(Magn, MagnStr);
           SetViewPort(GetMaxX-20, 25, GetMaxX-1, 35, ClipOn);
           ClearViewPort;
           OutTextXY(0,0, MagnStr);
           SetViewPort(1, 21, GetMaxX-1, GetMaxY-101, ClipOn);
           { Définition des systèmes de coordonnées }
           With SIn do begin
             CSG.X := Spectrum^.GetIff; CSG.Y := 0;
             CID.X := Spectrum^.GetIfi; CID.Y := 4096;
           end;
           With SOut do begin
             CSG.X := 0; CSG.Y := 0; CID.X := GetMaxX-1;
             CID.Y := GetMaxY-121;
           end;
           SInRed := SIn;
           SOutRed := SOut;
           SInRed.CID.Y := SInRed.CID.Y + 200;
           With SInDiff do begin
```

```
                  CSG.X := Spectrum^.GetIff; CSG.Y := -200;
                  CID.X := Spectrum^.GetIfi; CID.Y := 200;
                end;
                With SOutDiff do begin
                  CSG.X := 0; CSG.Y := 0; CID.X := GetMaxX-1; CID.Y := 98;
                end;
                { Dessin du spectre et du fond }
                PRecord(Spectrum^.GetData^.At(3))^.Draw(SIn, SOut, Yellow,
                        True);
                PRecord(Spectrum^.GetData^.At(4))^.Draw(SIn, SOut, Red,
                        True);
                repeat
                  { Nettoyage de l'événement }
                  ClearEvent(E);
                  { On va chercher un événement dans la queue }
                  GetKeyEvent(E);
                  if (E.What <> evNothing) then begin
                    if (E.ScanCode in RedrawFondOnly) then
                         PRecord(Spectrum^.GetData^.At(4))^.
                               Draw(SIn, SOut, Red, False);
                    if (E.ScanCode in RedrawAll) and not(SpectreRed) then
                       ClearViewPort;
                    case E.ScanCode of
{ Home }              $47 : PRecord(Spectrum^.GetData^.At(4))^.
                            Rotate(-RAStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ End  }              $4F : PRecord(Spectrum^.GetData^.At(4))^.
                            Rotate(RAStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ Haut }              $48 : PRecord(Spectrum^.GetData^.At(4))^.
                            Shift(VShift, VStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ Bas  }              $50 : PRecord(Spectrum^.GetData^.At(4))^.
                            Shift(VShift, -VStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ Droite }            $4D : PRecord(Spectrum^.GetData^.At(4))^.
                            Shift(HShift, -HStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ Gauche }            $4B : PRecord(Spectrum^.GetData^.At(4))^.
                            Shift(HShift, HStep*Magn, Spectrum^.GetIfi,
                                   Spectrum^.GetIff);
{ F1 = Zoom }         $3B,
{ F2 = UnZoom }       $3C : if not(SpectreRed) then
                              With SIn do begin
                                if E.ScanCode = $3B
                                     then CSG.Y := CSG.Y + 300
                                      else CSG.Y := 0;
                                if E.ScanCode = $3B
                                     then CID.Y := CID.Y - 300
                                      else CID.Y := 4096;
                              end;
{ F3 = Sp. Red. }     $3D : begin
                              SpectreRed := not(SpectreRed);
                              if not(SpectreRed) then begin
                                ClearViewPort;
                                PRecord(Spectrum^.GetData^.At(4))^.
                                     Draw(SIn, SOut, Red, True);
                                PRecord(Spectrum^.GetData^.At(3))^.
                                     Draw(SIn, SOut, Yellow, True);
                              end;
                            end;
{ F4 = Refresh }      $3E : PRecord(Spectrum^.GetData^.At(3))^.
                                     Draw(SIn, SOut, Yellow, True);
{ Pg Up }             $49 : if not(SpectreRed) then
```

```
                        With SIn do begin
                          CSG.Y := CSG.Y + 500;
                          CID.Y := CID.Y + 500;
                        end;
{ Pg Dn }           $51 : if not(SpectreRed) then
                        With SIn do begin
                          CSG.Y := CSG.Y - 500;
                          CID.Y := CID.Y - 500;
                        end;
{ Différence }      $44 : begin
                        DiffActive := not(DiffActive);
                        if not(DiffActive) then begin
                          SetViewPort(1, GetMaxY-98, GetMaxX - 1,
                                      GetMaxY - 1, ClipOn);
                          ClearViewPort;
                          SetViewPort(1, 21, GetMaxX-1, GetMaxY-101,
                                      ClipOn);
                        end;
                      end;
{ +1 sur l'amplitude }
                    $4E : Magn := Magn + 1;
{ -1 sur l'amplitude }
                    $4A : if Magn > 1 then Magn := Magn - 1;
                  end;
                  if (E.ScanCode in RedrawAll)
                     and not(SpectreRed) then begin
                    PRecord(Spectrum^.GetData^.At(4))^.
                            Draw(SIn, SOut, Red, True);
                    PRecord(Spectrum^.GetData^.At(3))^.
                            Draw(SIn, SOut, Yellow, True);
                  end;
                  if (E.ScanCode in RedrawFondOnly) and not(SpectreRed)
                     then PRecord(Spectrum^.GetData^.At(4))^.
                            Draw(SIn, SOut, Red, True);
                  { Si spectre redressé sélectionné on le calcule et }
                  { le dessine                                       }
                  if SpectreRed and ((E.ScanCode in RedrawFondOnly) or
                     (E.ScanCode = $3D)) then begin
                    RR := New(PRecord, Init(PRecord(Spectrum^.GetData^.
                            At(4))^.Count, 0));
                    for I := 0 to PRecord(Spectrum^.GetData^.At(4))^
                                  .Count  - 1 do begin
                      New(V);
                      V1^ := PInteger(PRecord(Spectrum^.GetData^.
                                 At(3))^.At(I))^;
                      V2^ := PInteger(PRecord(Spectrum^.GetData^.
                                 At(4))^.At(I))^;
                      if (V2^ <> 0) then V^ := Trunc(V1^ / V2^ * 4096)
                                    else V^ := 0;
                      RR^.Insert(V);
                    end;
                    ClearViewPort;
                    RR^.Draw(SInRed, SoutRed, Green, True);
                    C.X := 0; C.Y := 200;
                    D.X := Spectrum^.GetNofBloc * 128; D.Y := 200;
                    ModifyCoord(C, SInRed, SOutRed, True);
                    ModifyCoord(D, SInRed, SOutRed, True);
                    Graph.Line(C.X, C.Y, D.X, D.Y);
                    Dispose(RR, Done);
                  end;
                  Str(Magn, MagnStr);
                  SetViewPort(GetMaxX-20, 25, GetMaxX-1, 35, ClipOn);
                  ClearViewPort;
                  OutTextXY(0,0, MagnStr);
```

```
                SetViewPort(1, 21, GetMaxX-1, GetMaxY-101, ClipOn);
                { Si différence active on la calcule et la dessine }
                if DiffActive and not(E.ScanCode in
                    RedrawAll+[$01, $3D, $3E, $4E, $4A]) then begin
                  if R <> nil then Dispose(R, Done);
                  R := New(PRecord, Init(PRecord(Spectrum^.GetData^.
                          At(4))^.Count, 0));
                  for I := 0 to PRecord(Spectrum^.GetData^.At(4))^.
                                        Count  - 1 do begin
                    New(V);
                    V^ := PInteger(PRecord(Spectrum^.GetData^.
                                At(3))^.At(I))^
                        - PInteger(PRecord(Spectrum^.GetData^.
                                At(4))^.At(I))^;
                    R^.Insert(V);
                  end;
                  SetViewPort(1, GetMaxY-98, GetMaxX - 1, GetMaxY - 1,
                              ClipOn);
                  ClearViewPort;
                  C.X := 0; C.Y := 0;
                  D.X := Spectrum^.GetNofBloc * 128; D.Y := 0;
                  ModifyCoord(C, SInDiff, SOutDiff, True);
                  ModifyCoord(D, SInDiff, SOutDiff, True);
                  Graph.Line(C.X, C.Y, D.X, D.Y);
                  R^.Draw(SInDiff, SOutDiff, Red, True);
                  SetViewPort(1, 21, GetMaxX-1, GetMaxY-101, ClipOn);
                end;
              end;
            until (E.What <> evNothing) and (E.ScanCode = $01);
            if R <> nil then Dispose(R, Done);
            GraphicsStop;
          end;
          GraphAppDone;
          if MessageBox(#3'D,sirez-vous sauver les modifications',
              nil, mfYesButton or mfNoButton or mfConfirmation)
              <> cmNo then SaveFile(Spectrum, FileName);
        end;
    end
    else MessageBox(#3'Fichier incompatible avec l''application.
        '#13#3'V,rifier le marqueur.', nil, mfError or mfOkButton);
    Dispose(Spectrum, Done);
  end;
  Dispose(V1); Dispose(V2);
end;

{ ProfPC permet une visualisation des ajustements de profils }
{ Le spectre fitté est superposé au spectre original et leur }
{ différence est affichée.                                   }

procedure ProfPC;
const
  Color : array[1..5] of Word = (Blue, Red, Green, Yellow, Magenta);
var
  I : Integer;
  V : PInteger;
  R : PRecord;
  E : TEvent;
  C, D : Coord;
  Spectrum : PSpectrum;
  SIn, SOut : SystemCoord;
  FileName : PathStr;
begin
  R := nil;
  Spectrum := ReadFile(FileName);
```

```
if Spectrum <> nil then begin
  if (Spectrum^.GetFlag = 'TR') then begin
    if RegisterBGIDriver(@EgaVgaDriver) < 0
      then MessageBox(#3'Impossible de charger le driver graphique
                       EGA/VGA.', nil, mfError or mfOkButton)
      else begin
        if GraphicsStart then begin
          { Préparation de l'écran }
          DrawInitialisation(Spectrum^.GetEntete, tyProfPC,
             Spectrum^.GetNofRecord, (Spectrum^.GetFranges <> nil));
          repeat
            { Nettoyage de l'événement }
            ClearEvent(E);
            { On va chercher un événement dans la queue }
            GetKeyEvent(E);
            if (E.What <> evNothing) then
              if (E.ScanCode > $01) and
                 (E.ScanCode-1 <= Spectrum^.GetNofRecord) then begin
                if R <> nil then Dispose(R, Done);
                R := New(PRecord, Init(PRecord(Spectrum^.GetData^.
                        At(E.ScanCode-2))^.Count, 0));
                for I := 0 to PRecord(Spectrum^.GetData^.
                                At(E.ScanCode-2))^.Count - 1 do begin
                  New(V);
                  V^ := PInteger(PRecord(Spectrum^.GetData^.
                                At(E.ScanCode-2))^.At(I))^
                      - PInteger(PRecord(Spectrum^.GetData^.
                                At(0))^.At(I))^;
                  R^.Insert(V);
                end;
                SetViewPort(1, 21, GetMaxX-1, GetMaxY-101, ClipOn);
                ClearViewPort;
                { Définition des systèmes de coordonnées des }
                { spectres                                   }
                With SIn do begin
                  CSG.X := Spectrum^.GetNofBloc * 128; CSG.Y := 0;
                  CID.X := 0; CID.Y := 4096;
                end;
                With SOut do begin
                  CSG.X := 0; CSG.Y := 0; CID.X := GetMaxX-1;
                  CID.Y := GetMaxY-101;
                end;
                PRecord(Spectrum^.GetData^.At(E.ScanCode-2))^.
                   Draw(SIn, SOut, Color[E.ScanCode - 1], True);
                PRecord(Spectrum^.GetData^.At(0))^.
                   Draw(SIn, SOut, Color[1], True);
                SetViewPort(1, GetMaxY-99, GetMaxX - 1, GetMaxY - 1,
                            ClipOn);
                ClearViewPort;
                { Définition des systèmes de coordonnées dela }
                { différence                                  }
                With SIn do begin
                  CSG.X := Spectrum^.GetNofBloc * 128;
                  CSG.Y := -200; CID.X := 0; CID.Y := 200;
                end;
                With SOut do begin
                  CSG.X := 0; CSG.Y := 0; CID.X := GetMaxX-1;
                  CID.Y := 99;
                end;
                { Affichage de la différence }
                C.X := 0; C.Y := 0;
                D.X := Spectrum^.GetNofBloc * 128; D.Y := 0;
                ModifyCoord(C, SIn, SOut, True);
                ModifyCoord(D, SIn, SOut, True);
```

```
                    Graph.Line(C.X, C.Y, D.X, D.Y);
                    R^.Draw(SIn, SOut, Color[E.ScanCode - 1], True);
                  end;
              until (E.What <> evNothing) and (E.ScanCode = $01);
              if R <> nil then Dispose(R, Done);
              GraphicsStop;
            end;
            GraphAppDone;
          end;
      end
      else MessageBox(#3'Fichier incompatible avec l''application.'
           #13#3'V,rifier le marqueur.', nil, mfError or mfOkButton);
      Dispose(Spectrum, Done);
    end;
  end;


begin
  TApplication.HandleEvent(Event);
  case Event.What of
    evCommand:
      begin
        case Event.Command of
          cmChangeDir : ChangeDir;{ Changer de répertoire }
          cmViewFile : ViewFile;  { Visualiser un fichier }
          cmTile : Tile;          { Réorganiser l'affichage des }
                                  { fenêtres à l'écran          }
          cmCascade : Cascade;    { Afficher les fenêtres les unes au }
                                  { dessus des autres à l'écran       }
          cmAbout : About;        { Afficher la boîte d'information de }
                                  { l'application                      }
          cmLasPC : LasPC;        { Application LASPC  }
          cmAjusPC : AjusPC;      { Application AJUSPC }
          cmProfPC : ProfPC       { Application PROFPC }
        else
          Exit;                   { Quitter l'application }
        end;
        ClearEvent(Event);    { L'événement a été traité, on l'élimine }
      end;
    end;
end;

{ TMyApp.InitStatusLine définit la StatusLine de l'application. }
{ La StatusLine sert à afficher les raccourcis clavier.         }

procedure TMyApp.InitStatusLine;
var
  R : TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('~Alt-X~ Exit', kbAltX, cmQuit,
      NewStatusKey('~F2~ View', kbF2, cmViewFile,
      NewStatusKey('~Alt-F2~ Close', kbAltF2, cmClose,
      NewStatusKey('~F7~ LasPC', kbF7, cmLasPC,
      NewStatusKey('~F8~ AjusPC', kbF8, cmAjusPC,
      NewStatusKey('~F9~ ProfPC', kbF9, cmProfPC,
      nil))))))),
    nil)));
end;

{ TMyApp.InitMenuBar initialise la barre de menu de l'application }
```

```
procedure TMyApp.InitMenuBar;
var
  R : TRect;
begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('~F~ile', hcNoContext, NewMenu(
      NewItem('~V~iew', 'F2', kbF2, cmViewFile, hcNoContext,
      NewItem('~C~hange dir...', '', kbNoKey, cmChangeDir, hcNoContext,
      NewLine(
      NewItem('~A~bout...', '', kbNoKey, cmAbout, hcNoContext,
      NewItem('E~x~it', 'Alt-X', kbAltX, cmQuit, hcNoContext,
      nil))))))),
    NewSubMenu('~W~indow', hcNoContext, NewMenu(
      NewItem('~C~lose', 'Alt-F2', kbAltF2, cmClose, hcNoContext,
      NewItem('~T~ile', '', kbNoKey, cmTile, hcNoContext,
      NewItem('C~a~scade', '', kbNoKey, cmCascade, hcNoContext,
      NewItem('~Z~oom', 'F5', kbF5, cmZoom, hcNoContext,
      nil))))),
    NewSubMenu('~D~raw', hcNoContext, NewMenu(
      NewSubMenu('Drawing', hcNoContext, NewMenu(
        NewItem('~L~asPC', 'F7', kbF7, cmLasPC, hcNoContext,
        NewItem('~A~jusPC', 'F8', kbF8, cmAjusPC, hcNoContext,
        NewItem('~P~rofPC', 'F9', kbF9, cmProfPC, hcNoContext,
        nil)))),
      nil)),
      nil))))));
end;

begin
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
end.
```

Fichier Include MemUtil.pas

```
{ EgaVgaDriver est une procédure de chargement du driver EGA/VGA. }

procedure EgaVgaDriver; External; {$L EgaVgadrv.obj}

{ SmallFontDriver est une procédure de chargement de petites }
{ fonts.                                                     }

procedure SmallFontDriver; External; {$L Litt.obj}

{ SearchInfo recherche le nombre de records, le nombre de }
{ blocs par record et si le pointé a été effectué.        }
{ La recherche est basée sur la valeur du flag.           }

function SearchInfo(Flag : String) : PInfoSpectre;
const
  MaxTable = 6;
type
  InfoTable = array[1..MaxTable] of InfoSpectre;
const
  Info : InfoTable = ((F : 'LI'; N : 4; M : 8; P : False),
                      (F : 'PI'; N : 4; M : 8; P : True),
                      (F : 'AE'; N : 5; M : 8; P : True),
                      (F : 'TR'; N : 5; M : 8; P : False),
                      (F : 'CI'; N : 5; M : 8; P : False),
                      (F : 'CE'; N : 5; M : 8; P : False));
var
  P : PInfoSpectre;
  I : Byte;
begin
  P := nil;
  for I := 1 to MaxTable do
    if Flag = Info[I].F then P := @Info[I];
  if P = nil then MessageBox(#3'Fichier incompatible avec
                  l''application', nil, mfError + mfOkButton);
  SearchInfo := P;
end;

{ ReadFile lit le contenu d'un fichier de spectres. }

function ReadFile(var FileName : PathStr) : PSpectrum;
var
  E : PEntete;
  D : PRecordCollection;
  F : PFrangesCollection;
  R : PRaiesCollection;
  P : PInfoSpectre;
  Q : PInfoRaie;
  Dialog : PFileDialog;
  DataFile : Text;
  StrBuffer : String;
  I, J, Code : Integer;
  Entier : PInteger;
  Reel : PReal;
  Rec : PRecord;

  { ExitReadFile libère les places mèmoire déjà accordées }
  { lors de la lecture du fichier.                        }

  procedure ExitReadFile;
  begin
    if E <> nil then Dispose(E, Done);
```

```
    if D <> nil then Dispose(D, Done);
    if F <> nil then Dispose(F, Done);
    if R <> nil then Dispose(R, Done);
    Dispose(Dialog, Done);
  end;

begin
{$I-}
  E := nil; D := nil; F := nil; R := nil; ReadFile := nil;
  Dialog := PFileDialog(MyApp.ValidView(New(PFileDialog, Init('*.*',
            'Select File', '~F~ile name', fdOkButton, 100))));
  if Dialog <> nil then begin
                              { Une boite FileDialog a pu être ouverte }
    if DeskTop^.ExecView(Dialog) <> cmCancel then begin
                              { Si on ne l'a pas refermée ... }
      Dialog^.GetFileName(FileName); { Saisie du nom de fichier }
      assign(DataFile, FileName);
      reset(DataFile);
      if IOCode <> 0 then
        begin ExitReadFile; Exit end;    { Erreur d'ouverture }

 { Lecture de l'entête }
      E := New(PEntete, Init(30, 1));
      for I := 1 to 3 do begin
        readln(DataFile, StrBuffer);
        if IOCode <> 0 then
          begin ExitReadFile; Exit end;  { Erreur de lecture }
        E^.Insert(NewStr(StrBuffer));
      end;
                                         { Recherche des Info }
      P := SearchInfo(Copy(PString(E^.At(2))^, 61, 2));
      if P = nil then
        begin ExitReadFile; Exit end;  { Fichier incompatible }
      if P^.P then J := 27 + P^.N + 1
              else J := 27 + P^.N;
      for I := 4 to J do begin
        readln(DataFile, StrBuffer);
        if IOCode <> 0 then
          begin ExitReadFile; Exit end;    { Erreur de lecture }
        E^.Insert(NewStr(StrBuffer));
      end;
      readln(DataFile); readln(DataFile);
      Val(Copy(PString(E^.At(1))^, 61, 2), P^.M, Code);

 { Lecture des records }
      D := New(PRecordCollection, Init(1, 1));
      for I := 1 to P^.N do begin
        Rec := New(PRecord, Init(P^.M*128, 0));
        for J := 1 to P^.M*128 do begin
          New(Entier);
          read(DataFile, Entier^);
          Rec^.Insert(Entier);
          if IOCode <> 0 then begin ExitReadFile; Exit end;
        end;
        D^.Insert(Rec);
                                           { Saut de 4 lignes }
        for J := 1 to 4 do readln(DataFile);
      end;

 { Lecture des franges et des raies }
      if P^.P then begin
        F := New(PFrangesCollection, Init(64, 0));
        for J := 1 to 64 do begin
          New(Reel);
```

```
          read(DataFile, Reel^);
          F^.Insert(Reel);
          if IOCode <> 0 then
            begin ExitReadFile; Exit end;  { Erreur de lecture }
        end;
        for J := 1 to 7 do readln(DataFile);
        R := New(PRaiesCollection, Init(1, 1));
        while not eof(DataFile) do begin
          New(Q);
          readln(DataFile, Q^.N, Q^.P, Q^.F, Q^.I, Q^.R);
          R^.Insert(Q);
          if IOCode <> 0 then
            begin ExitReadFile; Exit end;  { Erreur de lecture }
        end;
      end;
      ReadFile := New(PSpectrum, Init(E, D, F, R));
    end;
    Dispose(Dialog, Done);
  end;
{$I+}
end;

{ SaveFile permet d'enregistrer le fichier modifié après }
{ un ajustement.                                         }

procedure SaveFile(S : PSpectrum; FileName : PathStr);
var
  DataFile : Text;
  Dialog : PFileDialog;
  I, J, K, L : Integer;
  Chaine : String;
begin
  Dialog := PFileDialog(MyApp.ValidView(New(PFileDialog, Init(FileName,
          'Save File As', '~F~ile name', fdOkButton, 100))));
  if Dialog <> nil then begin
      if DeskTop^.ExecView(Dialog) <> cmCancel then begin
                                          { Si on veut sauver }
        Dialog^.GetFileName(FileName);    { Saisie du nom     }
        assign(DataFile, FileName);
        rewrite(DataFile);
                                          { Ecriture          }
        for I := 0 to S^.GetEntete^.Count - 1 do
            writeln(DataFile, String(S^.GetEntete^.At(I)^));
        writeln(DataFile);
        for I := 0 to S^.GetData^.Count - 1 do begin
          Str(I + 1, Chaine);
          writeln(DataFile,'Record  ' + Chaine); writeln(DataFile);
          for J := 0 to S^.GetNofBloc - 1 do begin
            for K := 0 to 15 do begin
              for L := 0 to 7 do
                  write(DataFile, PInteger(PRecord(S^.GetData^.At(I))^.
                                  At(J*128 + K*8 + L))^ : 6);
               writeln(DataFile);
            end;
            writeln(Datafile);
          end;
          writeln(DataFile);
        end;
        Close(DataFile);
    end;
    Dispose(Dialog, Done);
  end;
end;
```

```
{ DrawIitialisation dessine les caractéristiques d'un fichier,  }
{ affiche les commandes valides en fonction de l'application et }
{ prépare l'écran pour l'application.                           }

procedure DrawInitialisation(Entete : PEntete; What : DrawType; N : Byte;
P : Boolean);
var
  I : Byte;
  Chaine : String;
begin
  SetColor(White);
  Rectangle(0, 0, GetMaxX, GetMaxY);
  case What of
    tyLasPC  : begin
                 Rectangle(127, 0, GetMaxX, 20);
                 Graph.Line(127, 0, 127, GetMaxY-1);
                 Entete^.Draw;
                 SetViewPort(128, 1, GetMaxX-1, 19, ClipOn);
                 for I := 1 to N-1 do begin
                   Str(I, Chaine);
                   OutTextXY(10+(I-1)*32, 7, Chaine + ' - ');
                 end;
                 Str(N, Chaine);
                 OutTextXY(10+(N-1)*32, 7, Chaine);
                 if P then OutTextXY(N*32-6, 7, '- 9 = Pointe');
                 OutTextXY(N*32+100, 7, 'F10-Select Points');
                 OutTextXY(GetMaxX - 220, 7, 'Esc = Quit');
                 SetViewPort(128, 21, GetMaxX-1, GetMaxY-1, ClipOn);
               end;
    tyAjusPC : begin
                 Rectangle(0, 0, GetMaxX, 20);
                 Graph.Line(0, GetMaxY - 100, GetMaxX -1, GetMaxY - 100);
                 SetViewPort(1, 1, GetMaxX-1, 19, ClipOn);
                 RegisterBGIFont(@SmallFontDriver);
                 SetTextStyle(2, 0, 4);
                 OutTextXY(30, 4, 'F1-Zoom  F2-UnZoom  F3-Sp.Red.  F4-
                           Refresh  F10-Diff.' );
                 Graph.Arc(360, 10, 0, 270, 5);
                 Graph.Line(365, 10, 361, 8);
                 Graph.Line(365, 10, 369, 8);
                 OutTextXY(370, 4, '-End');
                 Graph.arc(410, 10, 90, 360, 5);
                 Graph.Line(415, 10, 411, 12);
                 Graph.Line(415, 10, 419, 12);
                 OutTextXY(420, 4, '-Home');
                 OutTextXY(GetMaxX - 70, 4, 'Esc = Quit');
                 SetTextStyle(0, 0, 1);
                 OutTextXY(460, 7, chr(24) + chr(25) + chr(26) + chr(27)
                           + chr(43) + chr(45));
                 SetViewPort(1, 21, GetMaxX-1, GetMaxY-1, ClipOn);
                 Graph.Line(0, GetMaxY - 120, GetMaxX -1, GetMaxY - 120);
                 SetViewPort(1, 21, GetMaxX-1, GetMaxY-101, ClipOn);
               end;
    tyProfPC : begin
                 Rectangle(0, 0, GetMaxX, 20);
                 Graph.Line(0, GetMaxY - 100, GetMaxX -1, GetMaxY - 100);
                 SetViewPort(1, 1, GetMaxX-1, 19, ClipOn);
                 for I := 1 to N-1 do begin
                   Str(I, Chaine);
                   OutTextXY(10+(I-1)*32, 7, Chaine + ' -');
                 end;
                 Str(N, Chaine);
                 OutTextXY(10+(N-1)*32, 7, Chaine);
                 OutTextXY(GetMaxX - 90, 7, 'Esc = Quit');
```

```
                SetViewPort(1, 21, GetMaxX-1, GetMaxY-1, ClipOn);
              end;
  end;
end;
```

Fichier Unit MemUnit.pas

```
{$X+}  { Extended Syntax enable }
{$G+}  { 286-Instructions }
unit MemUnit;

interface

uses
  Objects, Memory, Views, Drivers, App, MsgBox, { Units Turbo Vision }
  Dos, Graph,
  Utility, TVUtil;

type

{ String2 }
  String2 = String[2];

{ DrawType}
  DrawType = (tyLasPC, tyAjusPC, tyProfPC);

{ ShiftType }
  ShiftType = (HShift, VShift);

{ InfoSpectre contient des informations sur le spectre : le flag, }
{ le nombre de records, le nombre de blocs par records, si les    }
{ raies sont pointées.                                            }

{ InfoSpectre }
  PInfoSpectre = ^InfoSpectre;
  InfoSpectre = Record
    F : String[2];        { Flag     }
    N : Byte;             { NbreRec  }
    M : Byte;             { NbreBloc }
    P : Boolean           { Pointe ? }
  end;

{ InfoRaie contient des informations sur une raie détectée : son  }
{ numéro, sa position (en point fichier), sa fréquence, son       }
{ intensité, le record auquel elle appartient.                    }

{ InfoRaie }
  PInfoRaie = ^TInfoRaie;
  TInfoRaie = record
    N : Byte;             { Num,ro    }
    P : Real;             { Position  }
    F : Real;             { Fréquence }
    I : Integer;          { Intensité }
    R : Integer;          { Record    }
  end;

{ TEntete est une collection de strings, ces strings correspondent }
{ aux informations stockées dans l'entête du fichier de données    }
{ étudié.                                                          }

{ TEntete }
  PEntete = ^TEntete;
  TEntete = object(TLineCollection)
    procedure Draw;
  end;

{ TRecord est une collection d'entier, ces entiers représentent un }
{ spectre.                                                         }
```

```
{ TRecord }
  PRecord = ^TRecord;
  TRecord = object(TIntegerCollection)
    procedure Draw(S1, S2 : SystemCoord; Color : Word; Visible :
                   Boolean);
    procedure Shift(T : ShiftType; DS : Integer; First, Last : Integer);
    procedure Rotate(DA : Real; First, Last : Integer);
  end;

{ TRecordCollection est une collection de records de type TRecord. }

{ TRecordCollection }
  PRecordCollection = ^TRecordCollection;
  TRecordCollection = object(TCollection)
    procedure Extract(R : Byte; var P : PRecord);
    procedure Replace(R : Byte; P : PRecord);
    procedure FreeItem(P : Pointer); virtual;
  end;

{ TFrangesCollection est une collection de réels qui représentent les }
{ franges détectées.                                                   }

{ TFrangesCollection }
  PFrangesCollection = ^TFrangesCollection;
  TFrangesCollection = object(TRealCollection)
    procedure Draw(S1, S2 : SystemCoord; Color : Word);
  end;

{ TRaiesCollection est une collection de raies de type InfoRaie. }

{ TRaiesCollection }
  PRaiesCollection = ^TRaiesCollection;
  TRaiesCollection = object(TCollection)
    procedure Draw(S1, S2 : SystemCoord; Color : Word);
    procedure FreeItem(P: Pointer); virtual;
  end;

{ TSpectrum représente un spectre:il contient un entête, une collection }
{ de records, une collection de franges (facultatif), une collection    }
{ de raies (facultatif).                                                }

{ TSpectrum }
  PSpectrum = ^TSpectrum;
  TSpectrum = object(TObject)
    Entete : PEntete;
    Data   : PRecordCollection;
    Franges : PFrangesCollection;
    Raies : PRaiesCollection;
    constructor Init(E : PEntete; D : PRecordCollection; F :
                     PFrangesCollection; R : PRaiesCollection);
    function GetEntete : PEntete;
    function GetData : PRecordCollection;
    function GetFranges : PFrangesCollection;
    function GetRaies : PRaiesCollection;
    function GetNofRecord : Byte;
    function GetNofBloc : Integer;
    function GetFlag : String2;
    function GetIfi : Integer;
    function GetIff : Integer;
    destructor Done; virtual;
  end;

{ TFileWindow est une fenêtre de visualisation du contenu d'un fichier }
```

```
{ se trouvant sur disque.                                              }

{ TFileWindow }
  PFileWindow = ^TFileWindow;
  TFileWindow = object(TWindow)
    constructor Init(FileName: String);
  end;

{ TFileViewer est un élément visuel qui sait comment afficher le }
{ contenu d'un fichier dans une fenêtre.                         }

{ TFileViewer }
  PFileViewer = ^TFileViewer;
  TFileViewer = object(TScroller)
    FileLines: PCollection;
    IsValid: Boolean;
    constructor Init(Bounds: TRect; AHScrollBar, AVScrollBar: PScrollBar;
                     FileName: PathStr);
    procedure Draw; virtual;
    function Valid(Command: Word): Boolean; virtual;
    destructor Done; Virtual;
  end;

implementation

{ TEntete }

{ Draw affiche les caractéristiques stockées dans ses strings. }

procedure TEntete.Draw;
var
  I : Integer;
begin
  SetColor(White);
  OutTextXY(10, 10, 'Date :');
  OutTextXY(10, 40, 'Fichier :');
  OutTextXY(10, 70, 'Diode :');
  OutTextXY(10, 100, 'Temp. :');
  OutTextXY(10, 130, 'Courant :');
  OutTextXY(10, 160, 'Dial :');
  OutTextXY(10, 190, 'Run :');
  OutTextXY(10, 220, 'Référence :');
  OutTextXY(10, 250, 'Molécule :');
  OutTextXY(10, 280, 'Fréquence :');
  OutTextXY(10, 345, 'P (mbar) :');
  OutTextXY(15, 360, 'Ref.:');
  OutTextXY(15, 375, 'Gaz :');
  OutTextXY(10, 395, 'Temp. :');
  OutTextXY(15, 410, 'Ref.:');
  OutTextXY(15, 425, 'Gaz :');
  OutTextXY(10, 450, 'L (m) :');
  SetColor(Yellow);
  I := 0;
  while Copy(PString(At(I))^, 1, 4) <> 'DATE' do I := I + 1;
  OutTextXY(15, 25, Copy(PString(At(I))^, 54, 9));
  OutTextXY(15, 55, Copy(PString(At(I+1))^, 54, 9));
  OutTextXY(15, 85, Copy(PString(At(I+2))^, 61, 2));
  OutTextXY(15, 115, Copy(PString(At(I+3))^, 57, 6));
  OutTextXY(15, 145, Copy(PString(At(I+4))^, 57, 6));
  OutTextXY(15, 175, Copy(PString(At(I+5))^, 59, 4));
  OutTextXY(15, 205, Copy(PString(At(I+6))^, 59, 4));
  OutTextXY(15, 235, Copy(PString(At(I+7))^, 53, 10));
  OutTextXY(15, 265, Copy(PString(At(I+8))^, 53, 10));
  OutTextXY(15, 295, Copy(PString(At(I+13))^, 53, 10));
```

```
  OutTextXY(15, 310, Copy(PString(At(I+15))^, 53, 10));
  OutTextXY(15, 325, Copy(PString(At(I+17))^, 53, 10));
  OutTextXY(65, 360, Copy(PString(At(I+19))^, 58, 5));
  OutTextXY(65, 375, Copy(PString(At(I+21))^, 58, 5));
  OutTextXY(65, 410, Copy(PString(At(I+20))^, 58, 5));
  OutTextXY(65, 425, Copy(PString(At(I+22))^, 58, 5));
  OutTextXY(70, 450, Copy(PString(At(I+9))^, 57, 6));
  SetColor(White);
end;

{ TRecord }

{ Draw dessine un record de couleur color          }
{      avec S1 pour systŠme de coordonn,es input   }
{           S2 pour systŠme de coordonn,es output  }
{ Si visible = true alors Draw dessine le record   }
{ Si visible = false alors Draw efface le record   }

procedure TRecord.Draw(S1, S2 : SystemCoord; Color : Word; Visible :
Boolean);
var
  C : Coord;
  I : Integer;
  OldColor : Word;
begin
  OldColor := Graph.GetColor;
  if Visible then SetColor(Color)
             else SetColor(GetBkColor);
  C.X := 0;
  C.Y := PInteger(At(0))^;
  ModifyCoord(C, S1, S2, True);
  Graph.MoveTo(C.X, S2.CID.Y - C.Y);
  for I := 1 to Count - 1 do begin
    C.X := I;
    C.Y := PInteger(At(I))^;
    ModifyCoord(C, S1, S2, False);
    Graph.LineTo(C.X, S2.CID.Y - C.Y);
  end;
  SetColor(OldColor);
end;

{ Shift déplace les valeurs d'un record verticalement ou }
{ horizontalement d'une quatité DS.                      }

procedure TRecord.Shift(T : ShiftType; DS : Integer; First, Last :
Integer);
var
  I : Integer;
begin
  Case T of
    VShift : for I := First-1 to Last-1 do
               PInteger(At(I))^ := PInteger(At(I))^ + DS;
    HShift : begin
               if DS > 0 then begin
                 for I := Count - 1 downto DS do
                   PInteger(At(I))^ := PInteger(At(I-DS))^;
                 for I := 0 to DS - 1 do PInteger(At(I))^ := 0;
               end;
               if DS < 0 then begin
                 for I := 0 to Count-1+DS do
                   PInteger(At(I))^ := PInteger(At(I-DS))^;
                 for I := Count+DS to Count-1 do PInteger(At(I))^ := 0;
               end;
             end;
```

```
  end;
end;

{ Rotate effectue une rotation d'un record d'une quantité DA }
{ La rotation est centrée sur le milieu du record           }

procedure TRecord.Rotate(DA : Real; First, Last : Integer);
var
  I, J : Integer;
begin
  J := Count div 2;
  for I := First to Last do
    PInteger(At(I))^ := Trunc(PInteger(At(I))^ * (DA * (I-J+1) + 1));
end;

{ TRecordCollection }

{ Extract extrait le record de rang R de la collection des records. }
{ Il retourne un pointeur P sur ce record.                          }

procedure TRecordCollection.Extract(R : Byte; var P : PRecord);
var
  I : Integer;
  E : PInteger;
  Q : PRecord;
begin
  if (R > 0) and (R <= Count) then begin
    Q := PRecord(At(R-1));
    P := New(PRecord, Init(Q^.Count, 0));
    for I := 0 to Q^.Count do begin
      New(E);
      E^ := PInteger(Q^.At(I))^;
      P^.Insert(E);
    end;
  end;
end;

{ Replace remplace le record de rang R dans la collection par le record }
{ pointé par P.                                                         }

procedure TRecordCollection.Replace(R : Byte; P : PRecord);
var
  I : Integer;
  Q : PRecord;
begin
  if (R > 0) and (R <= Count) then begin
    Q := PRecord(At(R-1));
    for I := 0 to Q^.Count do begin
      PInteger(Q^.At(I))^ := PInteger(P^.At(I))^;
    end;
  end;
end;

{ FreeItem libère la place mémoire accordée à un élément, i.e. un }
{ élément de type TRecord.                                        }

procedure TRecordCollection.FreeItem(P: Pointer);
begin
  Dispose(PRecord(P), Done);
end;

{ TFrangesCollection }

{ Draw dessine les pointés des franges en couleur color          }
```

```
{              avec S1 pour système de coordonnées input        }
{              avec S2 pour système de coordonnées output       }

procedure TFrangesCollection.Draw(S1, S2 : SystemCoord; Color : Word);
var
  OldColor : Word;
  C : Coord;
  I : Byte;
begin
  OldColor := Graph.GetColor;
  SetColor(Color);
  for I := 0 to Count-1 do begin
    C.X := Trunc(PReal(At(I))^);
    C.Y := S1.CID.Y;
    if C.X <> 0 then begin
      ModifyCoord(C, S1, S2, True);
      Graph.Line(C.X, C.Y, C.X, C.Y - 10);
    end;
  end;
  SetColor(OldColor);
end;

{ TRaiesCollection }

{ Draw dessine les pointés des raies en couleur color           }
{              avec S1 pour système de coordonnées input        }
{              avec S2 pour système de coordonnées output       }

procedure TRaiesCollection.Draw(S1, S2 : SystemCoord; Color : Word);
var
  OldColor : Word;
  C : Coord;
  I : Byte;
begin
  OldColor := Graph.GetColor;
  SetColor(Color);
  for I := 0 to Count-1 do begin
    C.X := Trunc(PInfoRaie(At(I))^.P);
    C.Y := S1.CID.Y;
    if C.X <> 0 then begin
      ModifyCoord(C, S1, S2, True);
      Graph.Line(C.X, C.Y, C.X, C.Y - 10);
    end;
  end;
  SetColor(OldColor);
end;

{ FreeItem libère la place mémoire accordée à un élément, i.e. }
{ un élément  de type InfoRaie.                                 }

procedure TRaiesCollection.FreeItem(P: Pointer);
begin
  Dispose(PInfoRaie(P));
end;

{ TSpectrum }

{ Init initialise un spectre avec son entête, sa collection de records, }
{ sa collection de franges et sa collection de raies.                   }

constructor TSpectrum.Init(E : PEntete; D : PRecordCollection; F :
PFrangesCollection; R : PRaiesCollection);
begin
  Entete := E;
```

```
  Data := D;
  Franges := F;
  Raies := R;
end;

{ GetEntete fournit l'entête du spectre. }

function TSpectrum.GetEntete : PEntete;
begin
  GetEntete := Entete;
end;

{ GetData fournit la collection des records du spectre.}

function TSpectrum.GetData : PRecordCollection;
begin
  GetData := Data;
end;

{ GetFranges fournit la collection des franges du spectre.}

function TSpectrum.GetFranges : PFrangesCollection;
begin
  GetFranges := Franges;
end;

{ GetRaies fournit la collection des raies du spectre.}

function TSpectrum.GetRaies : PRaiesCollection;
begin
  GetRaies := Raies;
end;

{ GetNofRecord retourne le nombre de records du spectre. }

function TSpectrum.GetNofRecord : Byte;
begin
  GetNofRecord := Data^.Count;
end;

{ GetNofBloc retourne le nombre de blocs d'un record. }

function TSpectrum.GetNofBloc : Integer;
var
  I, Code : Integer;
begin
  Val(Copy(PString(Entete^.At(1))^, 61, 2), I, Code);
  GetNofBloc := I;
end;

{ GetFlag retourne le flag d'un spectre. }

function TSpectrum.GetFlag : String2;
begin
  GetFlag := Copy(PString(Entete^.At(2))^, 61, 2);
end;

{ GetIfi retourne la valeur de Ifi. }

function TSpectrum.GetIfi : Integer;
var
  I, J, Code : Integer;
begin
  if Franges <> nil then J := 13 + GetNofRecord + 1
```

```
                    else J := 13 + GetNofRecord;
  Val(Copy(PString(Entete^.At(J))^, 59, 4), I, Code);
  GetIfi := I;
end;

{ GetIff retourne la valeur de Iff. }

function TSpectrum.GetIff : Integer;
var
  I, J, Code : Integer;
begin
  if Franges <> nil then J := 14 + GetNofRecord + 1
                    else J := 14 + GetNofRecord;
  Val(Copy(PString(Entete^.At(J))^, 59, 4), I, Code);
  GetIff := I;
end;

{ Done détruit le spectre en restituant les places mémoires accordées. }

destructor TSpectrum.Done;
begin
  dispose(Entete, Done);
  dispose(Data, Done);
  if Franges <> nil then dispose(Franges, Done);
  if Raies <> nil then dispose(Raies, Done);
end;

{ TFileWindow }

{ Init initialise une fenêtre de visualisation d'un fichier. }

constructor TFileWindow.Init(FileName: String);
var
  R : TRect;
  V : PFileViewer;
begin
  DeskTop^.GetExtent(R);
  TWindow.Init(R, FileName, wnNoNumber);
  GetExtent(R);
  R.Grow(-1,-1);
  Options := Options + ofTileable;
  V := New(PFileViewer, Init(R,
    StandardScrollBar(sbHorizontal + sbHandleKeyboard),
    StandardScrollBar(sbVertical + sbHandleKeyBoard), FileName));
  Insert(V);
end;

{ TFileViewer }

{ Init initialise un elvis de visualisation du contenu d'un }
{ fichier dont le nom est pass, comme paramŠtre.           }

constructor TFileViewer.Init(Bounds: TRect; AHScrollBar, AVScrollBar:
PScrollBar; FileName: PathStr);
var
  FileToView: Text;
  Line: String;
  MaxWidth: Integer;
begin
  TScroller.Init(Bounds, AHScrollbar, AVScrollBar);
  GrowMode := gfGrowHiX + gfGrowHiY;
  IsValid := True;
  FileLines := New(PLineCollection, Init(5,5));
  {$I-}
```

```
  Assign(FileToView, FileName);
  Reset(FileToView);
  if IOResult <> 0 then
  begin
    MessageBox(#3'Impossible d''ouvrir '#13#3+Filename+'.', nil, mfError
               + mfOkButton);
    IsValid := False;
  end
  else
  begin
    MaxWidth := 0;
    while not Eof(FileToView) and not LowMemory do
    begin
      Readln(FileToView, Line);
      if Length(Line) > MaxWidth then MaxWidth := Length(Line);
      FileLines^.Insert(NewStr(Line));
    end;
    Close(FileToView);
  end;
  {$I+}
  SetLimit(MaxWidth, FileLines^.Count);
end;

{ Draw dessine le contenu d'un fichier dans une fenêtre. }

procedure TFileViewer.Draw;
var
  B: TDrawBuffer;
  C: Byte;
  I: Integer;
  S: String;
  P: PString;
begin
  C := GetColor(1);
  for I := 0 to Size.Y - 1 do
  begin
    MoveChar(B, ' ', C, Size.X);
    if Delta.Y + I < FileLines^.Count then
    begin
      P := FileLines^.At(Delta.Y + I);
      if P <> nil then S := Copy(P^, Delta.X + 1, Size.X)
      else S := '';
      MoveStr(B, S, C);
    end;
    WriteLine(0, I, Size.X, 1, B);
  end;
end;

{ Valid teste la validité d'une vue après sa création. }

function TFileViewer.Valid(Command: Word): Boolean;
begin
  Valid := IsValid;
end;

{ Done libère la place occupée par le contenu du fichier. }

destructor TFileViewer.Done;
begin
  Dispose(FileLines, Done);
  TScroller.Done;
end;

end.
```

Fichier Unit TVUtil.pas

```
{$X+} { Extended Syntax enable }
unit TVUtil;

{ TVUtil offre des objets simples et utiles utilisables avec Turbo }
{ Vision.                                                           }

interface

uses
  MsgBox, Objects, App, Gadgets;     { Units Turbo Vision utilisées }

type

{ PInteger }
  PInteger = ^Integer;

{ PReal }
  PReal = ^Real;

{ TIntegerCollection est une collection d'entiers. }

{ TIntegerCollection }
  PIntegerCollection = ^TIntegerCollection;
  TIntegerCollection = object(TCollection)
    procedure FreeItem(P: Pointer); virtual;
  end;

{ TRealCollection est une collection de réels. }

{ TRealCollection }
  PRealCollection = ^TRealCollection;
  TRealCollection = object(TCollection)
    procedure FreeItem(P: Pointer); virtual;
  end;

{ TLineCollection est une collection de strings. }

{ TLineCollection }
  PLineCollection = ^TLineCollection;
  TLineCollection = object(TCollection)
    procedure FreeItem(P: Pointer); virtual;
  end;

{ TStandardApplication est un objet de base pour la création          }
{ d'application. Il hérite des propriétés de TApplication et en plus  }
{ permet de visualiser l'espace mémoire disponible sur le tas pendant }
{ l'évolution de l'application. TStandardApplication permet également }
{ de signaler les manquements lors des tentatives d'allocation mémoire. }

{ TStandardApplication }
  PStandardApplication = ^TStandardApplication;
  TStandardApplication = object(TApplication)
    Heap : PHeapView;                  { Pointeur sur la valeur du tas }
    constructor Init;
    procedure Idle; virtual;
    procedure OutOfMemory; virtual;
  end;


{ La fonction IOCode retourne une valeur entière qui est le status de  }
{ la dernière op,ration d'I/O réalisée. IOCode est une version         }
```

```
{ améliorée de IOResult en ce sens qu'elle ouvre une boîte de dialogue }
{ en indiquant le type d'erreur survenue. Si la dernière opération     }
{ d'I/O  s'est déroulée correctement, IOCode vaut 0.                   }

function IOCode : Word;

implementation

function IOCode : Word;
var
  Message : String;
  Code : Word;
begin
  Code := IOResult;
  if Code <> 0 then begin
    case Code of
      2   : Message := 'File not found.';
      3   : Message := 'Path not found.';
      4   : Message := 'Too many open files.';
      5   : Message := 'File access denied.';
      100 : Message := 'Disk read error.';
      101 : Message := 'Disk write error.';
      102 : Message := 'File not assigned.';
      103 : Message := 'File not open.';
      104 : Message := 'File not open for input.';
      105 : Message := 'File not open for output.';
      106 : Message := 'Invalid numeric format.';
    end;
    MessageBox(#3 + Message, nil, mfError + mfOkButton)
  end;
  IOCode := Code;
end;

{ TIntegerCollection }

{ FreeItem libère la place accordée à un ,élément; i.e. un entier. }
procedure TIntegerCollection.FreeItem(P: Pointer);
begin
  Dispose(PInteger(P));
end;

{ TRealCollection }

{ FreeItem libère la place accordée à un élément; i.e. un réel. }
procedure TRealCollection.FreeItem(P: Pointer);
begin
  Dispose(PReal(P));
end;

{ TLineCollection }

{ FreeItem libère la place accordée à un élément; i.e. un string. }
procedure TLineCollection.FreeItem(P: Pointer);
begin
  DisposeStr(P);
end;

{ TStandardApplication }

{ Init initialise une application standard. }
constructor TStandardApplication.Init;
var
  R : TRect;
begin
```

```
  TApplication.Init;
  GetExtent(R);
  Dec(R.B.X);
  R.A.X := R.B.X-9; R.A.Y := R.B.Y-1;
                              { Initialisation de la valeur de Heap^ }
  Heap := New(PHeapView, Init(R));
  Insert(Heap);
end;

{ Idle est appelé par TProgram.GetEvent chaque fois que la queue        }
{ d'événement est vide. L'application peut dès lors réaliser des tâches }
{ de fond en attendant une action de l'utilisateur. Ici, Idle affiche   }
{ l'espace mémoire disponible sur le tas; c'est notament très utile en  }
{ cours de développement pour situer des oublis de libération de        }
{ mémoire.                                                              }
procedure TStandardApplication.Idle;
begin
  TApplication.Idle;
  Heap^.Update;
end;

{ OutOfMemory est appelé par TProgram.ValidView chaque fois qu'une      }
{ atteinte de la réserve mémoire est détectée (la fonction LowMemory    }
{ renvoie True). OutOfMemory avertit l'utilisateur que l'espace mémoire }
{ n'est pas suffisant pour terminer l'opération en affichant le message }
{ "Out of Memory".                                                      }
procedure TStandardApplication.OutOfMemory;
begin
  MessageBox(#3'Out of Memory', nil, mfError + mfOkButton);
end;

end.
```

Fichier Unit Utility.pas

```
unit Utility;

interface

type

  { Coord }

  Coord = record
    X, Y : Integer;
  end;

  { SystemCoord }

  SystemCoord = record
    CSG, CID : Coord;
  end;

{ ModifyCoord modifie les coordonnés d'un point C défini dans un    }
{ système de coordonnées S1 en ses nouvelles coordonnées dans un    }
{ système S2. Si Evaluate est mis … TRUE, la procédure calcule      }
{ les coefficients de la transformation entre les 2 systèmes; ces   }
{ coefficients sont conservés en mémoire et il n'est donc pas       }
{ nécessaire de les recalculer lors de chaque appel à ModifyCoord. }

procedure ModifyCoord(var C : Coord; S1, S2 : SystemCoord; Evaluate :
Boolean);

implementation

var
  UA1, UA2, UB1, UB2 : Real;

procedure ModifyCoord(var C : Coord; S1, S2 : SystemCoord; Evaluate :
Boolean);

begin
  if Evaluate then
  begin
    UA1 := (S2.CID.X - S2.CSG.X) /(S1.CID.X - S1.CSG.X);
    UB1 := (LongInt(S2.CSG.X)*LongInt(S1.CID.X) - LongInt(S2.CID.X) *
           LongInt(S1.CSG.X)) / (S1.CID.X - S1.CSG.X);
    UA2 := (S2.CID.Y - S2.CSG.Y) /( S1.CID.Y - S1.CSG.Y);
    UB2 := (LongInt(S2.CSG.Y)*LongInt(S1.CID.Y) - LongInt(S2.CID.Y) *
           LongInt(S1.CSG.Y)) / (S1.CID.Y - S1.CSG.Y);
  end;
  C.X := Trunc(UA1*C.X + UB1);
  C.Y := Trunc(UA2*C.Y + UB2);
end;

end.
```

Fichier Unit Mouse.pas

```
Unit Mouse;

{ Unit de gestion simple de la souris }

interface

uses Dos, Utility;

{ InitMouse tente d'initialiser le driver de la souris et retourne }
{ une valeur True si l'initialisation s'est parfaitement déroulée, }
{ False sinon.                                                     }

function InitMouse : Boolean;

{ IsLeftButtonClick teste si le bouton gauche de la souris a été   }
{ appuyé. Si c'est le cas, la fonction retourne une valeur True et }
{ la variable C contient la position actuelle du curseur de la     }
{ souris.                                                          }

function IsLeftButtonClick(var C : Coord) : Boolean;

{ DisplayMouse affiche le curseur de la souris. }

procedure DisplayMouse;

{ ClearMouse élimine le curseur de la souris de l'écran. }

procedure ClearMouse;

{ SetMouseViewPort fixe les zones de déplacement horizontale et    }
{ verticale de la souris. C1 contient les positions horizontale    }
{ et verticale minimales de la souris; C2 les positions maximales. }

procedure SetMouseViewPort(C1, C2 : Coord);

implementation

function InitMouse : Boolean;
var
  R : Registers;
begin
  R.AX := $0;
  Intr($33, R);
  if R.AX = $FFFF then InitMouse := True
                  else InitMouse := False;
end;

function IsLeftButtonClick(var C : Coord) : Boolean;
var
  R : Registers;
begin
  IsLeftButtonClick := False;
  R.AX := $3;
  R.BX := 0;
  Intr($33, R);
  if R.BX <> 0 then begin
    write(#7);
    IsLeftButtonClick := True;
    C.X := R.CX;
    C.Y := R.DX;
  end;
```

```
end;

procedure Displaymouse;
var
  R : Registers;
begin
  R.AX := $1;
  Intr($33, R);
end;

procedure Clearmouse;
var
  R : Registers;
begin
  R.AX := $2;
  Intr($33, R);
end;

procedure SetMouseViewPort(C1, C2 : Coord);
var
  R : Registers;
begin
  R.AX := $7;
  R.CX := C1.X;
  R.DX := C2.X;
  Intr($33, R);
  R.AX := $8;
  R.CX := C1.Y;
  R.DX := C2.Y;
  Intr($33, R);
end;

end.
```

Annexe 2

Cette annexe reprend les différents programmes écrits en FORTRAN sur le VAX et utilisant les librairies IMSL; il s'agit des programmes de linéarisation et d'ajustements de profils. Le programme LINR réalise la linéarisation des spectres de raies isolées tandis que le programme LINR2 réalise la linéarisation des spectres à raies multiples. Le programme PROFIL réalise l'ajustement de profils sur des raies isolées et le programme MULTI l'ajustement sur des raies multiples.

Fichier Linr.for

```
************************************************************************
***                  PROGRAMME  LINR
***
***                     30-02-92
***
***   Ce programme linéarise les spectres d'élargissement et
***   d'intensité.
***   Le fichier d'entrée doit posséder la structure suivante :
***
***   Fichier __________ d'entrée :
***   - Record 1 : Fond du Spectre
***   - Record 2 : Franges
***   - Record 3 : Reference
***   - Record 4 : Spectre
***   - Record 5 : Fond de la reference
***   - Record 0 : Franges et Raies détectées sur les records 3 et 4
***
***   Fichier LA_______ de sortie (LI) :
***   - Record 1 : Fond du Spectre = Fond de la reference
***   - Record 2 : Franges
***   - Record 3 : Reference
***   - Record 4 : Spectre
***   - Record 5 : Spectre redresse (inutile ici)
***   - Record 0 : Franges et Raies détectées sur les records 3 et 4
***
************************************************************************

*******************************
***
***   Routine : CLS
***
***   But : Effacer l'écran.
***
***   Appel : Call  cls
```

```
***
************************************
      SUBROUTINE  cls
      print *, char(27), '[2J'
      END

*********************************************************************
***
***   Routine : PRINTYX (LINE, ROW, TEXT)
***
***   But : Imprimer un texte à un endroit précis sur l'écran .
***
***   Arguments : LINE = ordonnée du début du texte (INTEGER)
***               ROW  = abscisse du début du texte (INTEGER)
***               TEXT = texte à imprimer (CHARACTER*..)
***
***   Appel : Call printyx(line, row, text)
***
*********************************************************************
      SUBROUTINE printyx(line, row, text)
      INTEGER  line, row
      CHARACTER*(*) text, lline*2, rrow*2
      WRITE (lline,'(i2.2)') line
      WRITE (rrow,'(i2.2)') row
      PRINT *,char(27), '[', lline, ';', rrow, 'H', text
      END

*********************************************************************
***
***   Routine : LINRSPECTRUM (IN, OUT)
***
***   But : Lineariser un spectre.
***
***   Arguments : IN = Numero du spectre a lineariser; contenu dans Iak
***               OUT = Numero du spectre linearise; contenu dans Kiak
***
***   Appel : Call LinrSpectrum(In, Out)
***
*********************************************************************
      SUBROUTINE LinrSpectrum(In,Out)
      COMMON     Ifi, Iff, Xf, Ntot, Iak, Kiak
      INTEGER    Iak(12288), I, J, Iff, Ifi, NbrePts, In, Out,
     c           OfsetIn, OfsetOut, Kiak(12288), Ntot, Borne
      REAL       Yf(2048), Csval, Break(2048), Cscoef(4,2048),
     c           Xf(2048), X, Pas, Ff

      PARAMETER  (Pas=0.0001)

      EXTERNAL   Csakm, Csval

      OfsetIn = (In - 1) * Ntot
      OfsetOut = (Out - 1) * Ntot
      NbrePts = Iff - Ifi + 1
      DO 10 I=Ifi,Iff
       J = I - Ifi + 1
10    Yf(J) = REAL(Iak(OfsetIn+I))
      CALL Csakm(NbrePts,Xf,Yf,Break,Cscoef)
      Borne = Ntot/2 - 1
      DO 20 I=-Borne,Borne,1
         OfsetOut = OfsetOut + 1
```

```
         Kiak(OfsetOut) = 0
         X = REAL(I) * Pas
         IF(X.LE.Xf(1).OR.X.GE.Xf(NbrePts)) GOTO 20
         Ff = Csval(X,Iff-Ifi,Break,Cscoef)
         Kiak(OfsetOut) = INT(Ff)
20     CONTINUE
       END


*********************************************************************
***
***    Routine : CONVERSION (XFREQ, COEF)
***
***    But : Convertir les positions point en frequence.
***
***    Arguments : IN = Numero du spectre a lineariser; contenu dans Iak
***                OUT = Numero du spectre linearise; contenu dans Kiak
***
***    Appel : Call LinrSpectrum(In, Out)
***
*********************************************************************

       SUBROUTINE Conversion(Xfreq,Coef)
       COMMON      Ifi, Iff, Xf
       REAL        Xf(2048), Xfreq, Coef(4), Xp
       INTEGER     I, J, Ifi, Iff

       DO 10 I=Ifi,Iff
          J=I-Ifi+1
          Xp=REAL(I)                                          ! Position en points
10     Xf(J)=((Coef(4)*Xp+Coef(3))*Xp+Coef(2))*Xp+Coef(1)-Xfreq ! Position en
                                                                  fréquence
       END

*********************************************************************
***
***    PROGRAMME PRINCIPAL
***
*********************************************************************
       Program LINR

       Implicit None

       COMMON        Ifi, Iff, Xf, Ntot, Iak, Kiak
       Character*61 Record(6)
       Character PathFile*30, Date*10, Molecule*10, Buffer*10, File*6,
      c          Reference*6, Temp*6, Courant*6, Dial*4, Run*4,
      c          Flag*2, Diode*2,
      c          Num1, Num2, Num3,
      c          FileName*20, NewName*20
       Integer NbreBloc,Ifi, Iff, NbreRef, I, J, K, L, Iof, Ioff, Iofff,
      c        NbreRaie, NumRaie(128), Intensite(128), NumSpectre(128),
      c        NbreFrange, N, Ios, Ntot, Iak(12288), Kiak(12288),
      c        Continuum, Etalon, Ref, Spectre, FondSp, FondRef
       Real Freq1, Freq2, Freq3, Pref, Pgaz, TempRef, TempGaz,
      c     Position(128), Frequence(128), Frange(64),
      c     Y(64), Coef(4), Sspoly(4), Stat(10), X, Xfreq, Xf(2048),
      c     Yf(2048), Lcell, Pas
```

```
      External   Cls, Printyx, Rcurv, Conversion, LinrSpectrum

* Ouverture et lecture du fichier *
      PRINT 1000,'Nom du fichier à traiter ? '
      Read(*,'(A)') FileName
      Open(20,Iostat=Ios,Err=9000,File=FileName,Access='Sequential',
     c     Status='Old')
      Read(20,Fmt='(a)',Iostat=Ios,Err=9000) PathFile
      Read(20,Fmt='(42x,I20)',Iostat=Ios,Err=9000) NbreBloc
      Read(20,Fmt='(60x,A)',Iostat=Ios,Err=9000) Flag
      If (Flag.NE.'AE'.AND.Flag.NE.'PI') THEN
         Write(*,*) ' Fichier ', FileName, ' incompatible'
         Goto 9000
      EndIf
      Do 1 I=1,6
1             Read(20,Fmt='(A)',Iostat=Ios,Err=9000) Record(I)
      Read(20,Fmt='(52x,A)',Iostat=Ios,Err=9000) Date
      Read(20,Fmt='(56x,A)',Iostat=Ios,Err=9000) File
      Read(20,Fmt='(60x,A)',Iostat=Ios,Err=9000) Diode
      Read(20,Fmt='(56x,A)',Iostat=Ios,Err=9000) Temp
      Read(20,Fmt='(56x,A)',Iostat=Ios,Err=9000) Courant
      Read(20,Fmt='(58x,A)',Iostat=Ios,Err=9000) Dial
      Read(20,Fmt='(58x,A)',Iostat=Ios,Err=9000) Run
      Read(20,Fmt='(56x,A)',Iostat=Ios,Err=9000) Reference
      Read(20,Fmt='(52x,A)',Iostat=Ios,Err=9000) Molecule
      Read(20,Fmt='(56x,F6.2)',Iostat=Ios,Err=9000) Lcell
      Read(20,Fmt='(58x,I4)',Iostat=Ios,Err=9000) Ifi
      Read(20,Fmt='(58x,I4)',Iostat=Ios,Err=9000) Iff
      Read(20,Fmt='(61x,I1)',Iostat=Ios,Err=9000) NbreRef
      Read(20,Fmt='(52x,F8.4)',Iostat=Ios,Err=9000) Freq1
      Read(20,Fmt='(61x,A)',Iostat=Ios,Err=9000) Num1
      Read(20,Fmt='(52x,F8.4)',Iostat=Ios,Err=9000) Freq2
      Read(20,Fmt='(61x,A)',Iostat=Ios,Err=9000) Num2
      Read(20,Fmt='(56x,F6.4)',Iostat=Ios,Err=9000) Freq3
      Read(20,Fmt='(61x,A)',Iostat=Ios,Err=9000) Num3
      Read(20,Fmt='(54x,F8.4)',Iostat=Ios,Err=9000) Pref
      Read(20,Fmt='(58x,F4.2)',Iostat=Ios,Err=9000) TempRef
      Read(20,Fmt='(54x,f8.4)',Iostat=Ios,Err=9000) Pgaz
      Read(20,Fmt='(58x,F4.2)',Iostat=Ios,Err=9000) TempGaz
      Read(20,Fmt='(A)',Iostat=Ios,Err=9000) Buffer
      BackSpace(20)
      Ntot = NbreBloc*128
      IF (Flag.EQ.'AE') THEN
          FondSp = 1
          Etalon = 2
          Ref = 3
          Spectre = 4
          FondRef = 5
      ELSE
          FondSp = 1
          Etalon = 2
          Ref = 3
          Spectre = 4
          FondRef = 1
      ENDIF
      Pas = 0.0001

* Lecture des 5 records *
      Do 3 I=0,4
         Read(20,'(//)')
```

```
        Iof = NbreBloc*128*I
* de nbrebloc chacun (nbrenloc=8,16,32) *
        Do 3 J=0,NbreBloc-1
           Ioff = J*128
           Read(20,'(a)')
* chaque bloc se compose de 16*8 valeurs *
           Do 3 K=0,15
              Iofff = K*8
3             Read(20,'(8I6)')(Iak(Iof+Ioff+Iofff+L),L=1,8)

* Lecture des franges *
     Read(20,'(///)')
     Read(20,'(8(8F8.2/))')Frange

* Lecture des raies (128 valeurs maximum) *
     Read(20,'(/////)')
     Do 4 I=1,127
4       Read(20,'(I5,F10.2,F12.5,I5,I6)',Iostat=Ios,Err=9000,End=5)
    c  NumRaie(I),Position(I),Frequence(I),Intensite(I),NumSpectre(I)
5    NbreRaie = I-1

* DETERMINATION DE LA RELATION POSITION POINT - FREQUENCE *
* LE ZERO DES FREQUENCES COINCIDE AVEC LA PREMIERE FRANGE DETECTEE *
* LA RELATION EST UN POLYNOME DE DEGRE 3 DE LA FORME SUIVANTE *
*    Frequence = C(4)*Pos**3+C(3)*Pos**2+C(2)*Pos+C(1)

     Do 6 I=1,64
        If(Frange(I).EQ.0) GOTO 7                  ! Derniere frange ?
6       Y(I) = REAL(I-1)*0.0097711
7    NbreFrange=I-1
     CALL Rcurv(NbreFrange,Frange,Y,3,Coef,Sspoly,Stat)

* TRAITEMENT D'UNE RAIE DU SPECTRE *

     CALL Cls
     CALL Printyx(3,2,'Liste des raies du spectre')
     CALL Printyx(4,2,'--------------------------')
     DO 8 I=1,NbreRaie
8       IF (NumSpectre(I).EQ.Spectre) WRITE(*,'(I5,F10.2,I5)')
    c                              NumRaie(I),Position(I),Intensite(I)
10   Print *,' Entrer le numero de la raie à traiter'
     READ(*,*)N
     IF(NumSpectre(N).NE.Spectre) GOTO 10
     X = Position(N)                                ! Changement de variable
     XFreq = ((Coef(4)*X+Coef(3))*X+Coef(2))*X+Coef(1) ! Fréquence relative
                              ! de la raie par rapport à la première frange

     CALL Conversion(Xfreq,Coef)
     CALL LinrSpectrum(Spectre,2)

* IDEM SUR LE FOND DU SPECTRE *

     CALL LinrSpectrum(FondSp,1)

* IDEM SUR LA REFERENCE *

     CALL LinrSpectrum(Ref,3)

* TRAITEMENT D'UNE RAIE DE LA REFERENCE *
```

```
      CALL Cls
      CALL Printyx(3,2,'Liste des raies de la référence')
      CALL Printyx(4,2,'------------------------------')
      DO 81 I=1,NbreRaie
81       IF (NumSpectre(I).EQ.Ref) WRITE(*,'(I5,F10.2,I5)')
     c                             NumRaie(I),Position(I),Intensite(I)
101   PRINT *,' Entrer le numero de la raie à traiter'
      READ(*,*)N
      IF(NumSpectre(N).NE.Ref) GOTO 101
      X = Position(N)                                  ! Changement de variable
      XFreq = ((Coef(4)*X+Coef(3))*X+Coef(2))*X+Coef(1) ! Fréquence relative
                                ! de la raie par rapport à la premiŠre frange
      CALL Conversion(Xfreq,Coef)
      CALL LinrSpectrum(Ref,5)

* IDEM SUR LE FOND DE LA REFERENCE *

      CALL LinrSpectrum(FondRef,4)

C************ECRITURE DU FICHIER
      NewName=FileName
      NewName(1:2)='LA'
      OPEN(21,IOSTAT=IOS,ERR=9000,FILE=NewName,access='sequential',
     +     STATUS='NEW')
      WRITE(21,FMT='(A)',IOSTAT=IOS,ERR=9000)PathFile
      WRITE(21,110,IOSTAT=IOS,ERR=9000)NbreBloc
      WRITE(21,111,IOSTAT=IOS,ERR=9000)'LI'
      Ntot=NbreBloc*128
      WRITE(21,112,IOSTAT=IOS,ERR=9000)'1'
      WRITE(21,113,IOSTAT=IOS,ERR=9000)'2'
      WRITE(21,114,IOSTAT=IOS,ERR=9000)'3'
      WRITE(21,115,IOSTAT=IOS,ERR=9000)'4'
      WRITE(21,116,IOSTAT=IOS,ERR=9000)Date
      WRITE(21,117,IOSTAT=IOS,ERR=9000)File
      WRITE(21,118,IOSTAT=IOS,ERR=9000)Diode
      WRITE(21,119,IOSTAT=IOS,ERR=9000)Temp
      WRITE(21,120,IOSTAT=IOS,ERR=9000)Courant
      WRITE(21,121,IOSTAT=IOS,ERR=9000)Dial
      WRITE(21,122,IOSTAT=IOS,ERR=9000)Run
      WRITE(21,123,IOSTAT=IOS,ERR=9000)Reference
      WRITE(21,124,IOSTAT=IOS,ERR=9000)Molecule
      WRITE(21,125,IOSTAT=IOS,ERR=9000)Lcell
      WRITE(21,126,IOSTAT=IOS,ERR=9000)Ifi
      WRITE(21,127,IOSTAT=IOS,ERR=9000)Iff
      WRITE(21,128,IOSTAT=IOS,ERR=9000)NbreRef
      WRITE(21,129,IOSTAT=IOS,ERR=9000)Freq1
      WRITE(21,130,IOSTAT=IOS,ERR=9000)Num1
      WRITE(21,131,IOSTAT=IOS,ERR=9000)Freq2
      WRITE(21,132,IOSTAT=IOS,ERR=9000)Num2
      WRITE(21,133,IOSTAT=IOS,ERR=9000)Freq3
      WRITE(21,134,IOSTAT=IOS,ERR=9000)Num3
      WRITE(21,135,IOSTAT=IOS,ERR=9000)PRef
      WRITE(21,136,IOSTAT=IOS,ERR=9000)TempRef
      WRITE(21,137,IOSTAT=IOS,ERR=9000)PGaz
      WRITE(21,136,IOSTAT=IOS,ERR=9000)TempGaz
      WRITE(21,FMT='(A)',IOSTAT=Ios,ERR=9000)Buffer

*     ECRITURE des 5 records
      DO 16 I=0,4
*          WRITE(21,'(//)')
```

```
          WRITE(21,'(/'' Record'',i2/)')I+1
          if(i.eq.0)backspace(21)
          IOF=NbreBloc*128*I
*     de nbloc chacun (nbloc=8,16,32)
          DO 16 J=0,NbreBloc-1
              IOFF= J*128
              WRITE(21,'(a)')
*     chaque bloc compose de 16*8 valeurs
              DO 16 K=0,15
                  IOFFF=K*8
16                WRITE(21,'(8I6)')(KIAK(IOF+IOFF+IOFFF+L),L=1,8)
      WRITE(21,FMT='(a)',IOSTAT=IOS,ERR=9000)'           '
      GO TO 9999
9000  IF (Ios.EQ.0) THEN
         WRITE(*,*) ' Programme terminé'
      ELSE
         WRITE(6,'(a,i5)')' erreur fichier no : ',ios
      ENDIF
9999  close(20)
      CLOSE(21)

C
110   FORMAT('NBRE DE RECORDS D UN SCAN                    ',I20)
111   FORMAT('FLAG                                         ',A20)
112   FORMAT('1 CONTINUUM LINEARISE                        ',A20)
113   FORMAT('2 SPECTRE LINEARISE                          ',A20)
114   FORMAT('3 REFERENCE LINEARISEE                       ',A20)
115   FORMAT('4 REFERENCE LINEARISEE ET CENTREE            ',A20)
116   FORMAT('DATE                                         ',A20)
117   FORMAT('FICHIER                                      ',A20)
118   FORMAT('NO DIODE                                     ',A20)
119   FORMAT('TEMPERATURE                                  ',A20)
120   FORMAT('COURANT                                      ',A20)
121   FORMAT('DIAL                                         ',A20)
122   FORMAT('RUN(mA)                                      ',A20)
123   FORMAT('REFERENCE                                    ',A20)
124   FORMAT('MOLECULE                                     ',A20)
125   FORMAT('LONGEUR DE LA CELLULE                        ',12X,F8.4)
126   FORMAT('IFI POINT DEPART ETUDE FRANGE                ',I20)
127   FORMAT('IFF POINT FINAL ETUDE FRANGE                 ',I20)
128   FORMAT('NOMBRE DE REFERENCE                          ',I20)
129   FORMAT('FREQUENCE DE LA REFERENCE 1                  ',12X,F8.4)
130   FORMAT('NUMERO DE LA REFERENCE 1 DANS LE FICHIER     ',19X,A)
131   FORMAT('FREQUENCE DE LA REFERENCE 2                  ',12X,F8.4)
132   FORMAT('NUMERO DE LA REFERENCE 2 DANS LE FICHIER     ',19X,A)
133   FORMAT('FREQUENCE DE LA REFERENCE 3                  ',12X,F8.4)
134   FORMAT('NUMERO DE LA REFERENCE 3 DANS LE FICHIER     ',19X,A)
135   FORMAT('PRESSION DU GAZ DE REFERENCE                 ',12X,F8.4)
136   FORMAT('TEMPERATURE DE LA CELLULE                    ',16X,F4.1)
137   FORMAT('PRESSION DU GAZ ETUDIE                       ',12X,F8.4)
1000  FORMAT(' ',A,$)

      END
```

Fichier Linr2.for

```
************************************************************************
***                  PROGRAMME  LINR2
***
***                       30-07-92
***
***   Ce programme linearise les spectres issus de POSIPC
***
***   Fichier d'entree (marqueur PI)
***   - Record 1 : Fond
***   - Record 2 : Franges
***   - Record 3 : Reference
***   - Record 4 : Spectre
***   - Record 5 : Spectre redresse (inutile ici)
***   - Record 0 : Franges et Raies detectees sur les records 3 et 4
***
***   Fichier de sortie (marqueur LI)
***     - Record 1 : Fond linearise
***     - Record 2 : Spectre linearise
***     - Record 3 : Reference linearisee
***     - Record 0 : Raies detectees sur les records 3 et 4
***
************************************************************************
      PROGRAM LINR2

      IMPLICIT NONE

      CHARACTER*62 RECORD(6), BUFFER(35)
      CHARACTER*10 FILENAME
      CHARACTER*2  FLAG
      INTEGER      I, J, K, L, N, IOF, IOFF, IOFFF,IOS,
     c             NBREBLOC, NTOT, IFI, IFF, NBRERAIE, NBREFRANGE,
     c             IAK(4096), KIAK(4096),
     c             NUMRAIE(128), INTENSITE(128), NUMSPECTRE(128),
     c             CENTRE
      REAL         PREF, PGAZ, TEMPREF, TEMPGAZ, PAS,
     c             POSITION(128), FREQUENCE(128), FRANGE(64), X,
     c             XFREQ, XF(2048), YF(2048),
     c             Y(64), COEF(4), SSPOLY(4), STAT(10)

      COMMON       IFI, IFF, XF, NTOT, IAK, KIAK, PAS

      EXTERNAL     RCURV, CONVERSION, LINRSPECTRUM

      PAS = 0.0001

C
C     LECTURE DU FICHIER INPUT
C

      PRINT *,'NOM DU FICHIER A TRAITER ? '
      READ(*,'(A)') FILENAME
      OPEN(20,IOSTAT=IOS,ERR=9000,FILE=FILENAME,ACCESS='SEQUENTIAL',
     c     STATUS='OLD')
      READ(20,FMT='(A)',IOSTAT=IOS,ERR=9000) BUFFER(1)
      READ(20,FMT='(42x,I20)',IOSTAT=IOS,ERR=9000) NBREBLOC
      READ(20,FMT='(60x,A)',IOSTAT=IOS,ERR=9000) FLAG
```

```
      IF (FLAG.NE.'PI') THEN
         WRITE(*,*) ' FICHIER ', FILENAME, ' INCOMPATIBLE'
         GOTO 9000
      ENDIF
      DO 1 I=1,6
1        READ(20,FMT='(A)',IOSTAT=IOS,ERR=9000) RECORD(I)
      DO 2 I=2,11
2        READ(20,FMT='(A)',IOSTAT=IOS,ERR=9000) BUFFER(I)
      READ(20,FMT='(58x,I4)',IOSTAT=IOS,ERR=9000) IFI
      READ(20,FMT='(58x,I4)',IOSTAT=IOS,ERR=9000) IFF
      DO 3 I=12,18
3        READ(20,FMT='(A)',IOSTAT=IOS,ERR=9000) BUFFER(I)
      READ(20,FMT='(54x,F8.4)',IOSTAT=IOS,ERR=9000) PREF
      READ(20,FMT='(57x,F5.2)',IOSTAT=IOS,ERR=9000) TEMPREF
      READ(20,FMT='(54x,F8.4)',IOSTAT=IOS,ERR=9000) PGAZ
      READ(20,FMT='(57x,F5.2)',IOSTAT=IOS,ERR=9000) TEMPGAZ
      READ(20,FMT='(A)',IOSTAT=IOS,ERR=9000) BUFFER(19)
      BACKSPACE(20)
      NTOT = NBREBLOC*128

C
C     LECTURE DES 5 RECORDS DE NBREBLOC (NBREBLOC = 8, 16)
C     CHAQUE BLOC SE COMPOSE DE 16*8 VALEURS
C
      DO 4 I=0,4
         READ(20,'(//)')
         IOF = NBREBLOC*128*I
         Do 4 J=0,NBREBLOC-1
            IOFF = J*128
            READ(20,'(A)')
            Do 4 K=0,15
               IOFFF = K*8
4              READ(20,'(8I6)')(IAK(IOF+IOFF+IOFFF+L),L=1,8)

C
C     LECTURE DES FRANGES
C
      READ(20,'(///)')
      READ(20,'(8(8F8.2/))')FRANGE

C
C     LECTURE DE RAIES (128 VALEURS MAXIMUM)
C
      READ(20,'(/////)')
      Do 5 I=1,127
5        READ(20,'(I5,F10.2,F12.5,I5,I6)',IOSTAT=IOS,ERR=9000,END=6)
     c  NUMRAIE(I),POSITION(I),FREQUENCE(I),INTENSITE(I),NUMSPECTRE(I)
6     NBRERAIE = I-1

      PRINT *,'NOM DU FICHIER RESULTAT ?'
      READ(*,'(A)') FILENAME
      OPEN(21,IOSTAT=IOS,ERR=9000,FILE=FILENAME,ACCESS='SEQUENTIAL',
     c     STATUS='NEW')

C
C     DETERMINATION DE LA RELATION POSITION POINT - FREQUENCE
C     LE ZERO DES FREQUENCES COINCIDE AVEC LA PREMIERE FRANGE DETECTEE
C     LA RELATION EST UN POLYNOME DE DEGRE 3 DE LA FORME SUIVANTE :
C        FREQUENCE = C(4)*POS**3+C(3)*POS**2+C(2)*POS+C(1)
C
```

```
      Do 7 I=1,64
         IF(FRANGE(I).EQ.0) GOTO 8                    ! Derniere frange ?
7        Y(I) = REAL(I-1)*0.0097711
8     NBREFRANGE=I-1
      CALL RCURV(NBREFRANGE,FRANGE,Y,3,COEF,SSPOLY,STAT)

C
C     LINEARISATION DES RECORDS
C     LE ZERO DES FREQUENCE COINCIDE AVEC LE CENTRE DU RECORD
C
      IF (NBREBLOC.EQ.8) THEN
          CENTRE = 512
      ELSE
          CENTRE = 1024
      ENDIF
      X = CENTRE
      XFREQ = ((COEF(4)*X+COEF(3))*X+COEF(2))*X+COEF(1)
      CALL CONVERSION(XFREQ,COEF)
      CALL LINRSPECTRUM(1,1)
      CALL LINRSPECTRUM(4,2)
      CALL LINRSPECTRUM(3,3)
      DO 9 I=1,NBRERAIE
      X = POSITION(I)
        FREQUENCE(I) = ((COEF(4)*X+COEF(3))*X+COEF(2))*X+COEF(1)-XFREQ
9       POSITION(I) = CENTRE + FREQUENCE(I)/PAS

C
C     ECRITURE DU FICHIER RESULTAT
C
      WRITE(21,FMT='(1X,A)',IOSTAT=IOS,ERR=9000) BUFFER(1)
      WRITE(21,110,IOSTAT=IOS,ERR=9000) NBREBLOC
      WRITE(21,111,IOSTAT=IOS,ERR=9000) 'LI'
      WRITE(21,112,IOSTAT=IOS,ERR=9000) '1'
      WRITE(21,113,IOSTAT=IOS,ERR=9000) '2'
      WRITE(21,114,IOSTAT=IOS,ERR=9000) '3'
      WRITE(21,115,IOSTAT=IOS,ERR=9000) ' '
      WRITE(21,116,IOSTAT=IOS,ERR=9000) ' '
      WRITE(21,117,IOSTAT=IOS,ERR=9000) '0'
      DO 10 I=2,11
10    WRITE(21,FMT='(1X,A)',IOSTAT=IOS,ERR=9000) BUFFER(I)
      WRITE(21,118,IOSTAT=IOS,ERR=9000) IFI
      WRITE(21,119,IOSTAT=IOS,ERR=9000) IFF
      DO 11 I=12,18
11    WRITE(21,FMT='(1X,A)',IOSTAT=IOS,ERR=9000) BUFFER(I)
      WRITE(21,120,IOSTAT=IOS,ERR=9000) PREF
      WRITE(21,121,IOSTAT=IOS,ERR=9000) TEMPREF
      WRITE(21,122,IOSTAT=IOS,ERR=9000) PGAZ
      WRITE(21,121,IOSTAT=IOS,ERR=9000) TEMPGAZ
      WRITE(21,FMT='(1X,A)',IOSTAT=IOS,ERR=9000) BUFFER(19)

C
C     ECRITURE DES 3 RECORDS DE NBREBLOC CHACUN
C     CHAQUE BLOC SE COMPOSE DE 16*8 VALEURS
C
      DO 12 I=0,2
          WRITE(21,'(/'' RECORD'',I2/)')I+1
          IF(I.EQ.0) BACKSPACE(21)
          IOF=NBREBLOC*128*I
          DO 12 J=0,NBREBLOC-1
```

```
              IOFF= J*128
              WRITE(21,'(A)')
              DO 12 K=0,15
                 IOFFF=K*8
12               WRITE(21,'(8I6)')(KIAK(IOF+IOFF+IOFFF+L),L=1,8)
      WRITE(21,'(//)')
      WRITE(21,FMT=123,IOSTAT=IOS,ERR=9000)
      WRITE(21,'(/)')
      DO 13 I=1,NBRERAIE
      IF ((POSITION(I).GT.0).AND.(POSITION(I).LE.1024)) WRITE(21,
     c FMT='(I5,F10.2,F12.5,I5,I6)') NUMRAIE(I),POSITION(I),
     c FREQUENCE(I),INTENSITE(I),NUMSPECTRE(I)
13    CONTINUE
      GO TO 9999
9000  IF (IOS.EQ.0) THEN
         WRITE(*,*) ' PROGRAMME TERMINE'
      ELSE
         WRITE(6,'(A,I5)')' ERREUR FICHIER NO : ',IOS
      ENDIF
9999  CLOSE(20)
      CLOSE(21)

C
C     DECLARATION DES FORMATS
C
110   FORMAT(' NBRE DE RECORDS D''UN SCAN                 ',I20)
111   FORMAT(' FLAG                                      ',A20)
112   FORMAT(' 1 CONTINUUM LINEARISE                     ',A20)
113   FORMAT(' 2 SPECTRE LINEARISE                       ',A20)
114   FORMAT(' 3 REFERENCE LINEARISEE                    ',A20)
115   FORMAT(' 4 CONTINUUM REFERENCE CENTREE LINEARISE   ',A20)
116   FORMAT(' 5 REFERENCE LINEARISEE ET CENTREE         ',A20)
117   FORMAT(' 0 POINTE DES PICS                         ',A20)
118   FORMAT(' IFI POINT DEPART ETUDE FRANGE             ',I20)
119   FORMAT(' IFF POINT FINAL ETUDE FRANGE              ',I20)
120   FORMAT(' PRESSION DU GAZ DE REFERENCE              ',12X,F8.4)
121   FORMAT(' TEMPERATURE DE LA CELLULE                 ',15X,F5.2)
122   FORMAT(' PRESSION DU GAZ ETUDIE                    ',12X,F8.4)
123   FORMAT(' #RAIE  POSITION   FREQUENCE  INT  SCAN    ')
      END

*********************************************************************
***
***   Routine : LINRSPECTRUM (IN, OUT)
***
***   But : Lineariser un spectre.
***
***   Arguments : IN = Numero du spectre a lineariser; contenu dans Iak
***               OUT = Numero du spectre linearise; contenu dans Kiak
***
***   Appel : Call LinrSpectrum(In, Out)
***
*********************************************************************
      SUBROUTINE LINRSPECTRUM(IN,OUT)
      COMMON     IFI, IFF, XF, NTOT, IAK, KIAK, PAS
      INTEGER    IAK(4096), I, J, IFF, IFI, NBREPTS, IN, OUT,
     c           OFSETIN, OFSETOUT, KIAK(4096), NTOT, BORNE
      REAL       YF(2048), CSVAL, BREAK(2048), CSCOEF(4,2048),
     c            XF(2048), X, PAS, FF
```

```
      EXTERNAL   CSAKM, CSVAL

      OFSETIN = (IN - 1) * NTOT
      OFSETOUT = (OUT - 1) * NTOT
      NBREPTS = IFF - IFI + 1
      DO 10 I=IFI,IFF
       J = I - IFI + 1
10    YF(J) = REAL(IAK(OFSETIN+I))
      CALL CSAKM(NBREPTS,XF,YF,BREAK,CSCOEF)
      BORNE = NTOT/2 - 1
      DO 20 I=-BORNE,BORNE,1
         OFSETOUT = OFSETOUT + 1
         KIAK(OFSETOUT) = 0
         X = REAL(I) * PAS
         IF(X.LE.XF(1).OR.X.GE.XF(NBREPTS)) GOTO 20
         Ff = CSVAL(X,IFF-IFI,BREAK,CSCOEF)
         KIAK(OFSETOUT) = INT(FF)
20    CONTINUE
      END

*********************************************************************
***
***   Routine : CONVERSION (XFREQ, COEF)
***
***   But : Convertir les positions point en frequence et shifter le
***         zero de xfreq.
***
***   Arguments : XFREQ : Shift en frequence
***               COEF  : COEFficients de la relation Point-Frequence
***
***   Appel : Call Conversion (XFREQ, COEF)
***
*********************************************************************

      SUBROUTINE CONVERSION(XFREQ,COEF)
      COMMON     IFI, IFF, Xf
      REAL       XF(2048), XFREQ, COEF(4), XP
      INTEGER      I, J, IFI, IFF

      DO 10 I=IFI,IFF
         J=I-IFI+1
         XP=REAL(I)                                    ! Position en points
10    XF(J)=((COEF(4)*XP+COEF(3))*XP+COEF(2))*XP+COEF(1)-XFREQ ! Position en
                                                              ! frequence
      END
```

Fichier Profil.for

```
      PROGRAM PROFIL
      IMPLICIT NONE
      CHARACTER*65   BUFFER(35)
      CHARACTER*10   FILENAME, FILENAME2
      CHARACTER*1    REPONSE
      INTEGER        I,IOS,NBREBLOC,NTOT,CENTRE,LPROF,
     c               FONDSP(2048),SPECTRE(2048),REFERENCE(2048),
     c               FONDREF(2048),REFERENCECENTREE(2048),
     c               SPECTRECAL(2048),K,J,IOFF,IOFFF,L,WHAT,
     c               BINF,BSUP
      REAL*8         LCELL,SIGMA0,TEMPREF,TEMPGAZ,PAS,MASSE,CORR,
     c               PROF(2048),FGDOPTH,ERROR,LGR,LGSP,LAPP,LLSP,
     c               FLARGEUR,GDOPPLER,LAMBDA,AQ,XV,YV,SIGMA,FVOIGT,
     c               PROF2(2048),PROFVG(2,2048),X1,X2,DX,A0,AD,EPSILON,
     c               XX,YY,ZZ,WW,TOLERANCE
      LOGICAL        IMPRIME, MANUAL
      COMMON         CENTRE,PAS,PROF,LPROF,BINF,BSUP,GDOPPLER,SIGMA0,
     c               LAMBDA,AQ

C
C   VARIABLES UTILISEES PAR LES ROUTINES IMSL
C
      INTEGER        NMAX,MMAX
      PARAMETER      (NMAX = 3, MMAX = 2048)
      INTEGER        M,N,IPARAM(6),LDFJAC
      REAL*8         XGUESS(NMAX),XSCALE(NMAX),FSCALE(MMAX),RPARAM(7),
     c               X(NMAX),FVEC(MMAX),FJAC(MMAX,NMAX)
C
C   DEFINITION DES CONSTANTES
C
      REAL*8         PI,COEF
      PARAMETER      (PI = 3.141592654,
     c                COEF = 0.4697186)
C
C   DECLARATION DES ROUTINES EXTERNES
C
      EXTERNAL  DUNLSF, GAUSS, LORENTZ, PROFIL, FGDOPTH, FLARGEUR,
     c          VOIGT, VOIGTGENERAL, FVOIGT, INVERSEPROFIL, GALATRY,
     c          VGTRY

      WRITE(*,*) 'NOM DU FICHIER ? '
      READ(*,'(A)') FILENAME
      WRITE(*,*) 'RESULTATS IMPRIMES DANS UN FICHIER ? (Y/N)'
      READ(*,'(A1)') REPONSE
      IF ((REPONSE.EQ.'Y').OR.(REPONSE.EQ.'y')) THEN
          IMPRIME = .TRUE.
          WRITE(*,*) 'NOM DU FICHIER RESULTAT ?'
          READ(*,'(A)') FILENAME2
          OPEN(30,IOSTAT=IOS,ERR=9999,FILE=FILENAME2,ACCESS=
     c    'SEQUENTIAL',STATUS='NEW')
      ENDIF
      OPEN(10,IOSTAT=IOS,ERR=9999,FILE=FILENAME,ACCESS='SEQUENTIAL',
     c     STATUS='OLD')
      FILENAME(1:1) = 'T'
      OPEN(20,IOSTAT=IOS,ERR=9999,FILE=FILENAME,ACCESS='SEQUENTIAL',
```

```
c     STATUS='NEW')
      READ(10,'(A)') BUFFER(1)
      READ(10,'(60X,I2)')NBREBLOC
      NTOT = NBREBLOC * 128
      DO 5 I = 2,15
5     READ(10,'(A)') BUFFER(I)
      READ(10,'(55X,F7.4)') LCELL      ! EN METRES
      DO 6 I = 16,18
6     READ(10,'(A)') BUFFER(I)
      READ(10,'(53X,F9.4)') SIGMA0     ! EN CM-1
      DO 7 I = 19,24
7     READ(10,'(A)') BUFFER(I)
      READ(10,'(58X,F4.1)') TEMPREF    ! EN DEGRES
      READ(10,'(A)') BUFFER(25)
      READ(10,'(58X,F4.1)') TEMPGAZ    ! EN DEGRES
      READ(10,'(A)') BUFFER(26)
      READ(10,'(//)')
      READ(10,*)(FONDSP(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(SPECTRE(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(REFERENCE(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(FONDREF(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(REFERENCECENTREE(I),I=1,NTOT)

      WRITE(20,'(A)') BUFFER(1)
      WRITE(20,'(''NOMBRE DE RECORDS D UN SCAN'',34X,I2)')NBREBLOC
      DO 8 I = 2,15
8     WRITE(20,'(A)') BUFFER(I)
      WRITE(20,'(''LONGUEUR DE LA CELLULE'',34X,F7.4)') LCELL
      DO 9 I = 16,18
9     WRITE(20,'(A)') BUFFER(I)
      WRITE(20,'(''FREQUENCE DE LA REFERENCE 1'',27X,F9.4)') SIGMA0
      DO 10 I = 19,24
10    WRITE(20,'(A)') BUFFER(I)
      WRITE(20,'(''TEMPERATURE DE LA CELLULE'',34X,F4.1)') TEMPREF
      WRITE(20,'(A)') BUFFER(25)
      WRITE(20,'(''TEMPERATURE DE LA CELLULE'',34X,F4.1)') TEMPGAZ
      WRITE(20,'(A)') BUFFER(26)
      WRITE(20,'(//)')

      WRITE(20,'('' RECORD  1'')')
      DO 101 J=0,NBREBLOC-1
              IOFF = J*128
              WRITE(20,'(A)')
         DO 101 K=0,15
                 IOFFF = K*8
101              WRITE(20,'(8I6)')(SPECTRE(IOFF+IOFFF+I),I=1,8)
C
C   CONVERSION DES UNITES
C
      TEMPREF = 273.16 + TEMPREF          ! EN KELVIN
      TEMPGAZ = 273.16 + TEMPGAZ          ! EN KELVIN
      LCELL = LCELL * 100.0               ! EN CM
C
C   INITIALISATION DES VARIABLES
C
      IF (NBREBLOC.EQ.8) THEN
```

```
            CENTRE = 512
          ELSE
            CENTRE = 1024
          ENDIF
          PAS = 0.0001
C
C     LECTURE DES PARAMETRES AU TERMINAL
C
          WRITE(*,*) ' MASSE DE LA MOLECULE (U.M.A) ? '
          READ(*,*) MASSE
          WRITE(*,*)' FACTEUR DE CORRECTION SUR LE FOND ? '
          READ(*,*) CORR
4         WRITE(*,*)' DETERMINATION DES BORNES POUR LE FIT :'
          WRITE(*,*)'    1. MANUELLE'
          WRITE(*,*)'    2. AUTOMATIQUE'
          READ(*,*) WHAT
          IF (WHAT.EQ.1) THEN
             MANUAL = .TRUE.
             WRITE(*,*)' BORNE INFERIEURE ?'
             READ(*,*) BINF
             WRITE(*,*)' BORNE SUPERIEURE ?'
             READ(*,*) BSUP
          ELSE
          IF (WHAT.EQ.2) THEN
             MANUAL = .FALSE.
             WRITE(*,*)' INTENSITE DANS LES AILES ( % par rapport au
         ccentre : 0 - 1) ?'
             READ(*,*) TOLERANCE
          ELSE
             GOTO 4
          ENDIF
          ENDIF
C
C     CORRECTION DU FOND
C
          DO 13 I = 1,NTOT
13        FONDSP(I) = FONDSP(I) * CORR

C
C     FIT D'UNE GAUSSIENNE SUR LA RAIE DE REFERENCE (SPECTRE 4).
C     LA LARGEUR DE CETTE GAUSSIENNE CONTIENT LA LARGEUR DOPPLER
C     DE LA RAIE AINSI QUE LA LARGEUR DE LA FONCTION D'APPAREIL.
C     2 PARAMETRES SONT DETERMINES :
C          - LA LARGEUR DE LA GAUSSIENNE X(1)  (= LGR)
C          - L'INTENSITE AU CENTRE DE LA RAIE X(2)
C
          CALL PROFIL(FONDREF,REFERENCECENTREE,LCELL,CENTRE,PROF,LPROF,
         c MANUAL, TOLERANCE)
          N = 2
          M = LPROF    ! + 1
          LDFJAC = M
          XGUESS(1) = FGDOPTH(SIGMA0,TEMPREF,MASSE)
          XGUESS(2) = PROF(CENTRE)
          DO 50 I=1,N
50        XSCALE(I) = 1.0
          DO 60 I=1,M
60        FSCALE(I) = 1.0
          IPARAM(1) = 0
          CALL DUNLSF(GAUSS,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
         c            RPARAM,X,FVEC,FJAC,LDFJAC)
```

```
      LGR = X(1)
      ERROR = 0.0
      DO 70 I = 1,M
70    ERROR = ERROR + FVEC(I)**2
      ERROR = DSQRT(ERROR)/FLOAT(M)
      PRINT *,X(1), X(2), ERROR
C
C   CALCUL DE LA LARGEUR DE LA FONCTION D'APPAREIL
C
      LAPP = DSQRT(LGR**2-FGDOPTH(SIGMA0,TEMPREF,MASSE)**2)
      PRINT *,LAPP
C
C   FIT D'UNE GAUSSIENNE SUR LA RAIE (SPECTRE 2).
C   2 PARAMETRES SONT DETERMINES :
C       - LA LARGEUR DE LA GAUSSIENNE X(1) (= LGSP)
C       - L'INTENSITE AU CENTRE DE LA RAIE X(2)
C
      CALL PROFIL(FONDSP,SPECTRE,LCELL,CENTRE,PROF,LPROF,
     c MANUAL, TOLERANCE)
      N = 3
      M = LPROF   !+ 1
      LDFJAC = M
      XGUESS(1) = FGDOPTH(SIGMA0,TEMPGAZ,MASSE)
      XGUESS(2) = PROF(CENTRE)
      XGUESS(3) = 0.0
      DO 51 I=1,N
51    XSCALE(I) = 1.0
      DO 61 I=1,M
61    FSCALE(I) = 1.0
      IPARAM(1) = 0
      CALL DUNLSF(GAUSS,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM,X,FVEC,FJAC,LDFJAC)
      LGSP = X(1)
      ERROR = 0.0
      DO 71 I = 1,M
71    ERROR = ERROR + FVEC(I)**2
      ERROR = DSQRT(ERROR)/FLOAT(M)
      PRINT *,X(1), X(2), X(3), ERROR
      IF (IMPRIME.EQ.(.TRUE.)) THEN
         WRITE(30, '(//''Profil Gaussien ''/)')
         WRITE(30, *) 'Largeur : ',X(1)
         WRITE(30, *) 'Intens. : ',X(2)
         WRITE(30, *) 'Shift   : ',X(3)
         WRITE(30, *) 'Error   : ',ERROR
      ENDIF
C
C   CALCUL DU PROFIL GAUSSIEN THEORIQUE A PARTIR DES PARAMETRES
C   DETERMINES PAR LE FIT.
C
      DO 81 I=1,NTOT
      SIGMA = FLOAT(I-NTOT/2) * PAS
      IF (N.EQ.2) THEN
          PROF2(I) = X(2)*COEF*EXP(-LOG(2.0)*(SIGMA/X(1))**2)
      ELSE
          PROF2(I) = X(2)*COEF*EXP(-LOG(2.0)*((SIGMA-X(3))/X(1))**2)
      ENDIF
81    CONTINUE
      CALL INVERSEPROFIL(FONDSP,PROF2,LCELL,CENTRE,NTOT,SPECTRECAL)
      WRITE(20,'(//'' RECORD  2'')')
      DO 102 J=0,NBREBLOC-1
```

```
              IOFF = J*128
              WRITE(20,'(A)')
          DO 102 K=0,15
                 IOFFF = K*8
102              WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C    FIT D'UNE LORENTZIENNE SUR LA RAIE (SPECTRE 2).
C    2 PARAMETRES SONT DETERMINES :
C         - LA LARGEUR DE LA LORENTZIENNE X(1) (= LLSP)
C         - L'INTENSITE AU CENTRE DE LA RAIE X(2)
C
       CALL PROFIL(FONDSP,SPECTRE,LCELL,CENTRE,PROF,LPROF,
      c MANUAL, TOLERANCE)
       N = 2
       M = LPROF
       LDFJAC = M
       XGUESS(1) = FLARGEUR(LPROF,PROF)
       XGUESS(2) = PROF(CENTRE) * (XGUESS(1)**2)
       DO 52 I=1,N
52     XSCALE(I) = 1.0
       DO 62 I=1,M
62     FSCALE(I) = 1.0
       IPARAM(1) = 0
       CALL DUNLSF(LORENTZ,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
      c            RPARAM,X,FVEC,FJAC,LDFJAC)
       LLSP = X(1)
       ERROR = 0.0
       DO 72 I = 1,M
72     ERROR = ERROR + FVEC(I)**2
       ERROR = DSQRT(ERROR)/FLOAT(M)
       PRINT *,X(1), X(2), ERROR
       IF (IMPRIME.EQ.(.TRUE.)) THEN
           WRITE(30, '(//''Profil Lorentzien ''/)')
           WRITE(30, *) 'Largeur : ',X(1)
           WRITE(30, *) 'Intens. : ',X(2)
           WRITE(30, *) 'Error   : ',ERROR
       ENDIF

C
C    CALCUL DU PROFIL LORENTZIEN THEORIQUE A PARTIR DES PARAMETRES
C    DETERMINES PAR LE FIT.
C
       DO 82 I=1,NTOT
       SIGMA = FLOAT(I-NTOT/2) * PAS
82     PROF2(I) = X(2)/(SIGMA**2+X(1)**2)
       CALL INVERSEPROFIL(FONDSP,PROF2,LCELL,CENTRE,NTOT,SPECTRECAL)
       WRITE(20,'(//'' RECORD  3'')')
       DO 103 J=0,NBREBLOC-1
              IOFF = J*128
              WRITE(20,'(A)')
          DO 103 K=0,15
                 IOFFF = K*8
103              WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C    FIT D'UN PROFIL DE VOIGT SUR LA RAIE.
C    2 PARAMETRES SONT DETERMINES :
C         - LA LARGEUR LORENTZ X(1)  (=LLVSP)
C         - L'INTENSITE AU CENTRE DE LA RAIE X(2)
```

```
C   LA LARGEUR DOPPLER EST FIXEE A LA LARGEUR DU FIT DE
C   LA GAUSSIENNE SUR LA REFERENCE.
C
      CALL PROFIL(FONDSP,SPECTRE,LCELL,CENTRE,PROF,LPROF,
     c MANUAL, TOLERANCE)
      N = 2
      M = LPROF   !+ 1
      LDFJAC = M
      XGUESS(1) = LLSP
      XGUESS(2) = PROF(CENTRE)
      GDOPPLER = LGR
      DO 53 I=1,N
53    XSCALE(I) = 1.0
      DO 63 I=1,M
63    FSCALE(I) = 1.0
      IPARAM(1) = 0
      CALL DUNLSF(VOIGT,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM,X,FVEC,FJAC,LDFJAC)
      ERROR = 0.0
      DO 73 I = 1,M
73    ERROR = ERROR + FVEC(I)**2
      ERROR = DSQRT(ERROR)/FLOAT(M)
      PRINT *,X(1), X(2), ERROR
      IF (IMPRIME.EQ.(.TRUE.)) THEN
         WRITE(30, '(//''Profil de Voigt ''/)')
         WRITE(30, *) 'Largeur : ',X(1)
         WRITE(30, *) 'Intens. : ',X(2)
         WRITE(30, *) 'Error   : ',ERROR
      ENDIF

C
C   CALCUL DU PROFIL DE VOIGT THEORIQUE A PARTIR DES PARAMETRES
C   DETERMINES PAR LE FIT.
C
      DO 83 I=1,NTOT
      SIGMA = FLOAT(I-NTOT/2) * PAS
      XV = SQRT(LOG(2.0))*SIGMA/GDOPPLER
      YV = SQRT(LOG(2.0))*X(1)/GDOPPLER
83    PROF2(I) = X(2)*FVOIGT(XV,YV)
      CALL INVERSEPROFIL(FONDSP,PROF2,LCELL,CENTRE,NTOT,SPECTRECAL)
      WRITE(20,'(//'' RECORD  4'')')
      DO 104 J=0,NBREBLOC-1
             IOFF = J*128
             WRITE(20,'(A)')
         DO 104 K=0,15
                IOFFF = K*8
104             WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C   FIT D'UN PROFIL DE VOIGT GENERALISE SUR LA RAIE.
C   3 PARAMETRES SONT DETERMINES :
C        - LA LARGEUR LORENTZ X(1)  (=LLVGSP)
C        - LE SHIFT COLLISIONNEL X(2)
C        - L'INTENSITE AU CENTRE DE LA RAIE X(3)
C   LA LARGEUR DOPPLER EST FIXEE A LA LARGEUR DU FIT DE
C   LA GAUSSIENNE SUR LA REFERENCE.
C
      AQ = 3
      PRINT *, 'M(perturbateur) / M(absorbeur) : '
      READ(*,*) LAMBDA
```

```
      CALL PROFIL(FONDSP,SPECTRE,LCELL,CENTRE,PROF,LPROF,
     c MANUAL, TOLERANCE)
      N = 3
      M = LPROF   !+ 1
      LDFJAC = M
      XGUESS(1) = LLSP
      XGUESS(2) = 0.0
      XGUESS(3) = PROF(CENTRE)
      GDOPPLER = LGR
      DO 54 I=1,N
54    XSCALE(I) = 1.0
      DO 64 I=1,M
64    FSCALE(I) = 1.0
      IPARAM(1) = 0
      CALL DUNLSF(VOIGTGENERAL,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM,X,FVEC,FJAC,LDFJAC)
      ERROR = 0.0
      DO 74 I = 1,M
74    ERROR = ERROR + FVEC(I)**2
      ERROR = DSQRT(ERROR)/FLOAT(M)
      PRINT *,X(1), X(2), X(3), ERROR
      IF (IMPRIME.EQ.(.TRUE.)) THEN
         WRITE(30, '(//''Profil de Voigt generalise ''/)')
         WRITE(30, *) 'Largeur : ',X(1)
         WRITE(30, *) 'Shift   : ',X(2)
         WRITE(30, *) 'Intens. : ',X(3)
         WRITE(30, *) 'Error   : ',ERROR
      ENDIF

C
C   CALCUL DU PROFIL DE VOIGT GENERALISE THEORIQUE A PARTIR DES
C   PARAMETRES DETERMINES PAR LE FIT.
C
      K = NTOT/2
      X1 = (-FLOAT(K)*PAS)/GDOPPLER*SQRT(LOG(2.0))
      X2 = (FLOAT(K-1)*PAS)/GDOPPLER*SQRT(LOG(2.0))
      DX = PAS/GDOPPLER*SQRT(LOG(2.0))
      EPSILON = 1E-8
      A0 = X(1)/GDOPPLER*SQRT(LOG(2.0))
      AD = X(2)/GDOPPLER*SQRT(LOG(2.0))
      CALL VOIL(A0,AD,LAMBDA,AQ,X1,X2,DX,EPSILON,PROFVG)
      DO 84 I=1,NTOT
84    PROF2(I) = X(3) * PROFVG(2,I)
      CALL INVERSEPROFIL(FONDSP,PROF2,LCELL,CENTRE,NTOT,SPECTRECAL)
      WRITE(20,'(//'' RECORD  5'')')
      DO 105 J=0,NBREBLOC-1
            IOFF = J*128
            WRITE(20,'(A)')
         DO 105 K=0,15
               IOFFF = K*8
105            WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C   FIT D'UN PROFIL DE GALATRY SUR LA RAIE.
C   3 PARAMETRES SONT DETERMINES :
C       - LA LARGEUR LORENTZ X(1)  (=LLVGSP)
C       -  X(2)
C       - L'INTENSITE AU CENTRE DE LA RAIE X(3)
C   LA LARGEUR DOPPLER EST FIXEE A LA LARGEUR DU FIT DE
C   LA GAUSSIENNE SUR LA REFERENCE.
```

```
C
      CALL PROFIL(FONDSP,SPECTRE,LCELL,CENTRE,PROF,LPROF,
     c MANUAL, TOLERANCE)
      N = 3
      M = LPROF   ! + 1
      LDFJAC = M
      XGUESS(1) = LLSP
      XGUESS(2) = 0.0
      XGUESS(3) = PROF(CENTRE)
      GDOPPLER = LGR
      DO 55 I=1,N
55    XSCALE(I) = 1.0
      DO 65 I=1,M
65    FSCALE(I) = 1.0
      IPARAM(1) = 0
      CALL DUNLSF(GALATRY,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM,X,FVEC,FJAC,LDFJAC)
      ERROR = 0.0
      DO 75 I = 1,M
75    ERROR = ERROR + FVEC(I)**2
      ERROR = DSQRT(ERROR)/FLOAT(M)
      PRINT *,X(1), X(2), X(3), ERROR
      IF (IMPRIME.EQ.(.TRUE.)) THEN
          WRITE(30, '(//''Profil de Galatry ''/)')
          WRITE(30, *) 'Largeur : ',X(1)
          WRITE(30, *) 'Intens. : ',X(3)
          WRITE(30, *) 'Error   : ',ERROR
      ENDIF

C
C   CALCUL DU PROFIL DE GALATRY THEORIQUE A PARTIR DES PARAMETRES
C   DETERMINES PAR LE FIT.
C
      DO 86 I=1,NTOT
      SIGMA = FLOAT(I-NTOT/2) * PAS
      XX = SIGMA / GDOPPLER*SQRT(LOG(2.0))
      YY = X(1) / GDOPPLER*SQRT(LOG(2.0))
      ZZ = X(2) / (2*PI*GDOPPLER) * SQRT(LOG(2.0))
      CALL VGTRY(XX,YY,ZZ,WW)
86    PROF2(I) = X(3) * WW
      CALL INVERSEPROFIL(FONDSP,PROF2,LCELL,CENTRE,NTOT,SPECTRECAL)
      WRITE(20,'(//'' RECORD  3'')')
      DO 106 J=0,NBREBLOC-1
              IOFF = J*128
              WRITE(20,'(A)')
         DO 106 K=0,15
                 IOFFF = K*8
106              WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)


9999  STOP
      END

C
C   PROCEDURE PROFIL
C
      SUBROUTINE PROFIL(I0,I,L,C,K,G,MANUAL,TOL)
      IMPLICIT NONE
      INTEGER  I0(2048),I(2048),G,C,J,BINF,BSUP,LPROF,CENTRE
      REAL*8   L,K(2048),TOL,PAS,PROF(2048)
```

```
      LOGICAL  MANUAL
      COMMON   CENTRE,PAS,PROF,LPROF,BINF,BSUP
C
      K(C) = LOG(FLOAT(I0(C))/FLOAT(I(C)))/L
      IF (MANUAL.EQ.(.FALSE.)) THEN
         DO 10 J = 1,C-1
         K(C+J) = LOG(FLOAT(I0(C+J))/FLOAT(I(C+J)))/L
         K(C-J) = LOG(FLOAT(I0(C-J))/FLOAT(I(C-J)))/L
10       IF (((I0(C+J)-I(C+J)).LT.TOL*(I0(C)-I(C))).OR.
     c    ((I0(C-J)-I(C-J)).LT.TOL*(I0(C)-I(C)))) GOTO 20
20       G = 2*J+1
         BINF = C-J
         BSUP = C+J
      ELSE
         DO 15 J = BINF, BSUP
15       K(J) = LOG(FLOAT(I0(J))/FLOAT(I(J)))/L
         G = BSUP - BINF + 1
      ENDIF
      END
C
C   PROCEDURE INVERSEPROFIL
C
      SUBROUTINE INVERSEPROFIL(I0,PROF,L,C,G,I)
      IMPLICIT NONE
      INTEGER    I0(2048),I(2048),C,G,J
      REAL*8     PROF(2048),L
C
      DO 10 J=1,G
10    I(J) = FLOAT(I0(J))/EXP(L*PROF(J))
      END

C
C   FONCTION FGDOPTH
C
      REAL*8 FUNCTION FGDOPTH(SIGMA0,TEMPERATURE,MASSE)
      IMPLICIT NONE
      REAL*8   SIGMA0,TEMPERATURE,MASSE
      FGDOPTH = 3.58E-7 * DSQRT(TEMPERATURE/MASSE) * SIGMA0
      END

C
C   FONCTION FLARGEUR
C
      REAL*8 FUNCTION FLARGEUR(M,DATA)
      IMPLICIT NONE
      INTEGER    M,CENTRE,I,SIGMA1,SIGMA2
      REAL*8     DATA(2048),TMP1,TMP2,PAS
      COMMON     CENTRE,PAS

      SIGMA1 = 0.0
      SIGMA2 = 0.0
      M = M / 2
      DO 10 I=1,M
      IF ((DATA(CENTRE-I).GE.DATA(CENTRE)/2.0).OR.(SIGMA1.NE.0.0))
     c    GOTO 20
      SIGMA1 = CENTRE-I
20    IF ((DATA(CENTRE+I).GE.DATA(CENTRE)/2.0).OR.(SIGMA2.NE.0.0))
     c    GOTO 10
      SIGMA2 = CENTRE+I
10    CONTINUE
```

```
      TMP1 = (DATA(CENTRE)/2.0-DATA(SIGMA1))/(DATA(SIGMA1+1)
     c         - DATA(SIGMA1))
      TMP2 = (DATA(CENTRE)/2.0-DATA(SIGMA2))/(DATA(SIGMA2)
     c         - DATA(SIGMA2-1))
      FLARGEUR = ABS(SIGMA1+TMP2-(SIGMA2+TMP1))*PAS/2.0
      END

C
C   PROCEDURE GAUSS
C
      SUBROUTINE GAUSS(M,N,X,F)
      IMPLICIT NONE
      INTEGER     M,N,I,K,LPROF,CENTRE,BINF,BSUP
      REAL*8      PROF(2048),X(N),F(M),SIGMA,PAS
      COMMON      CENTRE,PAS,PROF,LPROF,BINF,BSUP
C
C   DECLARATION DES CONSTANTES
C
      REAL*8      COEF
      PARAMETER   (COEF=0.4697186)
      DO 10 I=BINF,BSUP
      SIGMA = FLOAT(I-CENTRE) * PAS
      IF (N.EQ.2) THEN
           F(I+1-BINF) = PROF(I)-X(2)*COEF*EXP(-LOG(2.0)*
     c                ((SIGMA)/X(1))**2)
      ELSE
           F(I+1-BINF) = PROF(I)-X(2)*COEF*EXP(-LOG(2.0)*
     c                ((SIGMA-X(3))/X(1))**2)
      ENDIF
10    CONTINUE
      END
C
C   PROCEDURE LORENTZ
C
      SUBROUTINE LORENTZ(M,N,X,F)
      IMPLICIT NONE
      INTEGER     M,N,I,K,CENTRE,LPROF,BINF,BSUP
      REAL*8      PROF(2048),X(N),F(M),SIGMA,PAS
      COMMON      CENTRE,PAS,PROF,LPROF,BINF,BSUP
C
      DO 10 I=BINF,BSUP
      SIGMA = FLOAT(I-CENTRE) * PAS
10    F(I+1-BINF) = PROF(I) - X(2)/(SIGMA**2+X(1)**2)
      END
C
C   PROCEDURE VOIGT
C
      SUBROUTINE VOIGT(M,N,X,F)
      IMPLICIT NONE
      INTEGER     M,N,I,LPROF,K,CENTRE,BINF,BSUP
      REAL*8      PROF(2048), X(N), F(M), XV, YV, FVOIGT,GDOPPLER,
     c            SIGMA, PAS, RACINE2
      EXTERNAL FVOIGT
      COMMON CENTRE, PAS, PROF, LPROF, BINF, BSUP, GDOPPLER

      RACINE2 = SQRT(LOG(2.0))
      DO 10 I=BINF,BSUP
      SIGMA = FLOAT(I-CENTRE) * PAS
      XV = RACINE2 * SIGMA / GDOPPLER
      YV = RACINE2 * X(1) / GDOPPLER
```

```
10     F(I+1-BINF) = PROF(I) - X(2) * FVOIGT(XV, YV)
       END
C
C    FONCTION FVOIGT
C
C    L'EVALUATION DE LA FONCTION DE VOIGT EST BASEE SUR L'ARTICLE SUIVANT :
C    A.K. HUI, B.H.ARMSTRONG, A.A.WRAY, J.Q.S.R.T. Vol 19, pp 555-558 (1978)
C
       REAL*8 FUNCTION FVOIGT(XV, YV)
       IMPLICIT NONE
       DIMENSION A(0:6),B(0:6)
       COMPLEX*16 ZV, BUFF
       INTEGER M, N
       REAL*8  XV, YV, A, B

       DATA A/122.607931777104326,214.382388694706425,
      .181.928533092181549,93.155580458138441,30.180142196210589,
      .5.912626209773153,0.564189583562615/
       DATA B/122.607931773875350,352.730625110963558,
      .457.334478783897737,348.703917719495792,170.354001821091472,
      .53.992906912940207,10.479857114260399/

       ZV = DCMPLX(YV,-XV)
       BUFF=((((((A(6)*ZV+A(5))*ZV+A(4))*ZV+A(3))*ZV+A(2))*ZV+A(1))*
      cZV+A(0))
      c/(((((((ZV+B(6))*ZV+B(5))*ZV+B(4))*ZV+B(3))*ZV+B(2))*ZV+B(1))*
      cZV+B(0))
       FVOIGT = REAL(BUFF)
       END

C
C    PROCEDURE VOIGTGENERAL
C
       SUBROUTINE VOIGTGENERAL(M,N,X,F)
       IMPLICIT NONE
       INTEGER    M,N,I,LPROF,CENTRE,K,BINF,BSUP
       REAL*8     PROF(2048), X(N), F(M), XV, YV, VOIL, EPSILON,
      c           GDOPPLER,SIGMA, PAS, SIGMA0, X1, X2, DX, A0, AD, AQ,
      c           LAMBDA, PROFC(2,2048)
       EXTERNAL VOIL
       COMMON CENTRE, PAS, PROF, LPROF, BINF, BSUP, GDOPPLER,
      c SIGMA0,LAMBDA,AQ

       K = LPROF/2
       X1 = (-FLOAT(K)*PAS)/GDOPPLER*SQRT(LOG(2.0))
       X2 = (FLOAT(K)*PAS)/GDOPPLER*SQRT(LOG(2.0))
       DX = PAS/GDOPPLER*SQRT(LOG(2.0))
       EPSILON = 1E-6
       print * , X(1)
       A0 = X(1)/GDOPPLER*SQRT(LOG(2.0))
       AD = X(2)/GDOPPLER*SQRT(LOG(2.0))
       CALL VOIL(A0,AD,LAMBDA,AQ,X1,X2,DX,EPSILON,PROFC)
       DO 10 I=BINF,BSUP
10     F(I+1-BINF) = PROF(I) - X(3) * PROFC(2,I+1-BINF)
       END

C
C    PROCEDURE VOIL
C
C    REFERENCE : D.COPE, R.KHOURY, R.J.LOVETT, J.Q.S.R.T., Vol 39, No 2, pp
```

```
C   509-516 (1978)
C
      SUBROUTINE VOIL(A0,AD,AL,AQ,X1,X2,DX,EPS,PROFIL)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION PROFIL(2,2048),ERAY(2000),HRAY(2,2000)
      NDIMX=2048
      NDIMT=2000
      PI32=5.568327996831708D0
C
C  INITIAL CHECK OF INPUT PARAMETERS AND CALCULATION OF P.
C
C  CAUTION:  NDIMX = DIMENSION OF PROFIL(2,NDIMX)
C            NDIMT = DIMENSION OF ERAY(NDIMT),HRAY(2,NDIMT)
C   IT AMY HAPPEN THAT LARGER ARRAYS ARE REQUIRED (NOTE
C   MESSAGES IN PROGRAM).  BE CAREFUL TO CHANGE BOTH NDIMT
C   AND ERAY,HRAY-DIMENSIONS IN THIS SUBROUTINE VOIL AS
C   WELL AS NDIMX AND PROFIL-DIMENSION IN THIS SUBROUTINE
C   VOIL AND PROFIL-DIMENSION IN SUBROUTINE CNFHLP.
C
C  INPUT PARAMETERS:
C
C   A0   = A-SUB-ZERO (WIDTH PARAMETER)
C   AD   = DELTA-SUB-ZERO (SHIFT PARAMETER)
C   AL   = LAMBDA (MASS RATIO)
C   AQ   = Q (MEASURE OF INTERATOMIC POTENTIAL)
C   X1   = INTITIAL X-VALUE
C   X2   = FINAL X-VALUE
C   DX   = SPACING BETWEEN X-VALUES
C   EPS  = EPSILON (MEASURE OF ABSOLUTE ERROR)
C
C  CONSTRAINTS ON INPUT PARAMETERS:
C
C   A0.GT.0
C   AL.GT.0
C   X2.GT.X1
C   DX.GT.0
C   1.GT.EPS.GT.0
C
      CONTINUE
C  DETERMINATION OF MODEL:
C
C   INPUT AQ.GT.1
C
C         OIL PROFILE OF WARD-COOPER-SMITH WITH AP=(AQ-3)/(AQ-1).
C         THE CASES ARE:
C         AQ.GT.3:  STANDARD OIL PROFILE.
C         AQ.EQ.3:  STANDARD VOIGT PROFILE.
C         3.GT.AQ.GT.1:  NONSTANDARD OIL PROFILE
C
C   INPUT AQ.LE.1
C
C         DEFAULT TO AQ = + INFINITY AND AP = 1, CORRESPONDING
C         TO THE FOWLER-SUNG PROFILE (UNMODIFIED IF AD.EQ.0;
C         MODIFIED IF AD.NE.0).
C
      IF((0.D0.LT.EPS).AND.(EPS.LT.1.D0)) GO TO 4
      WRITE(6,1000) EPS
      return  !STOP
    4 IF((A0.GT.0.D0).AND.(AL.GT.0.D0)) GO TO 8
      WRITE(6,1004) A0,AL
```

```
      return  !STOP
    8 IF((X1.LE.X2).AND.(DX.GT.0.D0)) GO TO 10
      WRITE(6,1008) X1,X2,DX
      return  !STOP
   10 IF(AQ.LE.3.D0) GO TO 20
      AP=(AQ-3.D0)/(AQ-1.D0)
      WRITE(6,1010) AQ,AP
      GO TO 100
   20 IF(AQ.LT.3.D0) GO TO 30
      AP=0.D0
      WRITE(6,1020) AQ,AP
      GO TO 100
   30 IF(AQ.LE.1.D0) GO TO 40
      AP=(AQ-3.D0)/(AQ-1.D0)
      WRITE(6,1030) AQ,AP
      GO TO 100
   40 AP=1.D0
      WRITE(6,1040) AP
C
C  PART 1.  FINDS NUMBER OF X-POINTS AND FILLS PROFIL(1,N) WITH X-VALUES
C
  100 CONTINUE
      NX=(X2-X1)/DX+1.D0+EPS
      IF(NX.LE.NDIMX) GO TO 110
      WRITE(6,1050) NX
      return  !STOP
  110 X=X1
      DO 120 I=1,NX
      PROFIL(1,I)=X
  120 X=X+DX
C
C  PART 2.  FINDS H=T-STEP FOR DISCRETIZATION ERROR ROUGHLY EPS.
C
  200 CONTINUE
      P=DMAX1(AP,0.D0)
      A=(A0*DSQRT(AL))/((1.D0+AL)**(P/2.D0))
      C2=2.D0*DMIN1(5.D0*A,5.D0)
      C4=2.D0*C2
      E=2.D0*DLOG(EPS/A)-1.D0
      IF((2.D0+DLOG(C2)+E).LT.0.D0) GO TO 210
      H=C2
      GO TO 300
  210 HN=C2
  220 HN=HN/2.D0
      IF((C2/HN+DLOG(HN)+E+1.D0).LE.0.D0) GO TO 220
  230 HN1=HN
      HN=(HN1*(C4+HN1*(DLOG(HN1)+E)))/(C2-HN1)
      IF(DABS(1.D0-HN1/HN).GE.0.01D0) GO TO 230
      H=HN
C
C  PART3. FINDS NT=NUMBER OF T-DIVISIONS FOR TRUNCATION ERROR
C         ROUGHLY APS.  ALSO FILLS IN ERAY(N), ESPONENTIAL VALUES.
C
  300 CONTINUE
      EPSCHK=(PI32*A0*EPS)/(((1.D0+AL)**(AP/2.D0))*DSQRT(AL))
      C1=DABS(AP)
      C2=AL/2.D0
      C3=(2.D0*DSQRT(AL)*DMAX1(DABS(X1),DABS(X2)))/3.D0
      N=1
      T=H
```

```
      ERAY(1)=DEXP(-(T*T)/AL)
  310 N=N+1
      IF(N.LE.NDIMT) GO TO 320
      WRITE(6,1060) NDIMT
      return  !STOP
  320 T=T+H
      E=DEXP(-(T*T)/AL)
      ERAY(N)=E
      IF((E*(1./T+C1*(T+C2/T+C3))).GE.EPSCHK) GO TO 310
      NT=N
C
C     WRITE(6,1100) NT,H
C1100 FORMAT(I10,1PE9.1)
C
C  PART 4.  FILLS IN HRAY(2,N) WITH H(T),H'(T)-VALUES.
C
  400 CONTINUE
      CALL CNFLHP(H,NT,AP,EPS,HRAY)
C
C  PART 5.  FILLS IN PROFIL(2,N) WITH OIL-VALUES.
C           NOTE THE USE OF H(U) EVEN, H'(U) ODD.
C
  500 CONTINUE
      RTL=DSQRT(AL)
      CAP=RTL/((1.D0+AL)**(AP/2.D0))
      ACAP=A0*CAP
      DCAP=AD*CAP
      COEF=(ACAP*H)/PI32
C
      DO 510 N=1,NX
      XCAP=PROFIL(1,N)*RTL
      T=0.D0
      SUM=1.D0/((XCAP-DCAP)*(XCAP-DCAP)+(ACAP*ACAP))
C
      DO 520 K=1,NT
      T=T+H
      H1=HRAY(1,K)
      H2=HRAY(2,K)
      H3=ACAP*H1
      H3=H3*H3
      H4=DCAP*H1
      TX=T-XCAP
      C1=-TX*H2+H1
      C2=TX+H4
      C2=C2*C2+H3
      TERM=C1/C2
      TX=T+XCAP
      C1=-TX*H2+H1
      C2=-TX+H4
      C2=C2*C2+H3
      TERM=TERM+C1/C2
      SUM=SUM+ERAY(K)*TERM
  520 CONTINUE
C
  510 PROFIL(2,N)=COEF*SUM
C
      RETURN
 1000 FORMAT(18H EPS OUT OF BOUNDS,1PE12.4)
 1004 FORMAT(20H A0,AL OUT OF BOUNDS, 1P2E12.4)
 1008 FORMAT(23H X1,X2,DX OUT OF BOUNDS, 1P3E12.4)
```

```
 1010 FORMAT(41H WARD-COOPER-SMITH(OIL) PROFILE WITH Q,P=,2F12.6)
 1020 FORMAT(31H STANDARD VOIGT PROFILE OR Q,P=,2F12.6)
 1030 FORMAT(26H NONSTANDARD OIL WITH Q,P=,2F12.6)
 1040 FORMAT(30H FOWLER-SUNG WITH Q,P=INFINITY, F12.6)
 1050 FORMAT(32H INCREASE NDIMX (PROFIL-DIM) TO ,I10)
 1060 FORMAT(1X,'INCREASE NDIMT (ERAY,HRAY-DIM).')
      END
C
      SUBROUTINE CNFLHP(DT,NTDIV,AP,EPS,HRAY)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION HRAY(2,2000)
C
      AP2=AP/2.D0
      C=NTDIV
      EPS1=EPS/C
      EPS2=EPS1*EPS1
      C=-DT*DT
C
      N=0
      BN=1.D0
      SUM=1.D0
      SUMD=0.D0
      IFLAG=0
C
   10 N=N+1
      RN=N
      BN=(C*(RN-1.D0-AP2)*BN)/(RN*(RN+0.5D0))
      SUM=SUM+BN
      SUMD=SUMD+(2.D0*RN*BN)/DT
      IF(DABS(BN).LT.EPS2) IFLAG=IFLAG+1
      IF(DABS(BN).GE.EPS2) IFLAG=0
      IF(IFLAG.LT.3) GO TO 10
C
      HRAY(1,1)=SUM
      HRAY(2,1)=SUMD
C
      IF(NTDIV.GT.1) GO TO 20
      RETURN
C
   20 T0=0.D0
      DO 100 NT=2,NTDIV
      T0=T0+DT
      N=2
      BN2=HRAY(1,NT-1)
      BN1=HRAY(2,NT-1)*DT
      BN=(-DT/T0)*((1.D0+T0*T0)*BN1-AP*T0*DT*BN2)
      SUM=BN+BN1+BN2
      SUMD=2.D0*(BN/DT)+(BN1/DT)
      IFLAG=0
C
   30 N=N+1
      RN=N
      BN3=BN2
      BN2=BN1
      BN1=BN
      BN=(RN-1.D0)*(RN+2.D0*T0*T0)*BN1+4.D0*T0*(RN-2.D0-AP2)*DT*BN2
      BN=-(BN+2.D0*(RN-3.D0-AP)*DT*DT*BN3)*(DT/(T0*RN*(RN-1.D0)))
      CHK=(RN*BN)/DT
      SUM=SUM+BN
      SUMD=SUMD+CHK
```

```
      IF(DABS(CHK).LT.EPS1) IFLAG=IFLAG+1
      IF(DABS(CHK).GE.EPS1) IFLAG=0
      IF(IFLAG.LT.6) GO TO 30
C
      HRAY(1,NT)=SUM
      HRAY(2,NT)=SUMD
C
  100 CONTINUE
C
      RETURN
      END

C
C   PROCEDURE GALATRY
C
      SUBROUTINE GALATRY(M,N,X,F)
      IMPLICIT NONE
      INTEGER    M,N,K,I,CENTRE,LPROF,BINF,BSUP
      REAL*8     X(N),F(M),PROF(2048),SIGMA,GDOPPLER,XX,YY,ZZ,WW,PAS
      COMMON     CENTRE,PAS,PROF,LPROF,BINF,BSUP,GDOPPLER
C
C   DEFINITION DES CONSTANTES
C
      REAL*8     PI
      PARAMETER (PI = 3.141592654)
C
      EXTERNAL   VGTRY
C
      K = LPROF/2
      DO 10 I=BINF,BSUP
      SIGMA = FLOAT(I-CENTRE) * PAS
      XX = SIGMA / GDOPPLER*SQRT(LOG(2.0))
      YY = X(1) / GDOPPLER*SQRT(LOG(2.0))
      ZZ = X(2) / (2*PI*GDOPPLER) * SQRT(LOG(2.0))
      CALL VGTRY(XX,YY,ZZ,WW)
10    F(I+1-BINF) = PROF(I) - X(3) * WW
      END

      SUBROUTINE VGTRY(X,Y,Z,G)
C**********************************************************************
C
C     THIS ROUTINE COMPUTES THE STANDARDIZED GALATRY FUNCTION
C     G(X',Y,Z)
C     WHERE:
C     X'(X) IS THE DISTANCE FROM THE SHIFTED LINE CENTER
C     X' = X-S
C     Y(Y)  IS THE COLLISIONAL BROADENING PARAMETER
C     Z(Z)  IS THE COLLISIONAL NARROWING PARAMETER
C     X',Y,Z ARE ALL NORMALIZED BY THE 1/E DOPPLER HALFWIDTH
C     Y CORRESPONDS THE THE VOIGT A COEFFICIENT AND
C       G(X,Y,0) = V(X,A)
C
C     THE VOIGT FUNCTION IS THE REAL PART OF THE COMPLEX
C     PROBABILITY FUNCTION COMPUTED BY THE ROUTINE CPF
C
C     PHILIP L. VARGHESE, APP. OPTICS, Vol 23, No 14, pp 2376-2385 (1984)
C
C**********************************************************************
      COMPLEX I,Q,W,DW(8),A(75),AO,ALPHA
      DIMENSION C(8),FACT(8),DEL(8)
```

```
      DATA I,ZOLD,RTPI/(0.,1.),0.,1.772454/
      DATA FACT/1.,2.,6.,24.,120.,720.,5040.,40320./
      DATA N2MAX,N3MAX/75,75/
      IF(Z.GT.0.) GO TO 5
C*****************************************************************
C            J
C     IF Z = 0, COMPUTE VOIGT FUNCTION (REAL(CPF))
C
      CALL CPF(X,Y,G,WI)
      RETURN
C
C*****************************************************************
C
C     BRANCH TO REGIONS I-III OF Y,Z PLANE DEPENDING ON INPUTS
C
5     CONTINUE
      IF(Z.GT.5) GO TO 200
      IF(Z.LE.0.1) GO TO 90
      IF(Y.GE.(4.*Z**0.868)) GO TO 300
      GO TO 200
90    IF((Y.GE.0.5).OR.(Z.GT.0.04).OR.(X.GT.2.)) GO TO 300
C
C*****************************************************************
C
C     REGION I - (0<Y<0.5, 0<Z<0.04, X<2) EXPANSION ABOUT VOIGT
C     FUNCTION
C
C     COMPUTE THE COEFFICIENTS OF THE ASYMPTOTIC EXPANSION OF
C                        N=N                          N=N
C     EXP(-1/(2*Z*Z)*SIGMA (-Z*T)**N/N!  = 1 + SIGMA C(N)*T**N
C                        N=3                          N=3
C
C     IF THE VALUE OF Z IS UNCHANGED SINCE THE LAST CALL THE
C     COEFFICIENTS ARE NOT RECOMPUTED
C
100   CONTINUE
      IF(Z.EQ.ZOLD) GO TO 15
      ZOLD=Z
      DO 10 J=3,8
10    C(J)=-(-Z)**(J-2)/(2.*FACT(J))
      C(6)=C(6)+C(3)*C(3)/2.
      C(7)=C(7)+C(3)*C(4)
      C(8)=C(8)+C(3)*C(5)+C(4)*C(4)/2
15    CALL CPF(X,Y,WR,WI)
      Q=CMPLX(X,Y)
      W=CMPLX(WR,WI)
C
C     COMPUTE DERIVATIVES (DW) OF THE COMPLEX PROBABILITY FUNCTION
C
      DW(1)=2*(I/RTPI-Q*W)
      DW(2)=-2*(Q*DW(1)+W)
      DO 20 J=3,8
20    DW(J)=-2*(Q*DW(J-1)+(J-1)*DW(J-2))
C
C     COMPUTE THE CORRECTION TO THE VOIGT PROFILE.  IF THE
C     ASYMPTOTIC EXPANSION DIVERGES THEN TERMINATE.
C
      DEL(3) = REAL(C(3)*DW(3)/(I**3))
      DELT = DEL(3)
      DO 30 J = 4,8
```

```
      DEL(J) = REAL(C(J)*DW(J)/I**J)
      IF(ABS(DEL(J)).GE.ABS(DEL(J-1))) GO TO 35
      DELT=DELT+DEL(J)
30    CONTINUE
35    G = WR + DELT
      RETURN
C             *
C*************************************************************************
C             *
C     REGION II - A & B    (0<Y<4Z**0.868, 0.1<Z<5) & (ALL Y, Z>5)    *
C             *
C     THE NUMBER OF TERMS IN THE SUM WAS DETERMINED EMPIRICALLY.      *
C             *
200   CONTINUE
      N2=4+(1.+3.*EXP(-1.1*Y))/Z**1.05
      IF(N2.GT.N2MAX) N2 = N2MAX
      Q = CMPLX(Y,-X)
      DELTA=0.5/(Z*Z)
      ALPHA=DELTA*(1.+2*Z*Q)
      W=1./ALPHA
      A(1)=W*DELTA/(ALPHA+1)
      W=W+A(1)
      DO 210 J=2,N2
      A(J)=A(J-1)*DELTA/(ALPHA+J)
      W=W+A(J)
210   CONTINUE
      G=REAL(W)/(RTPI*Z)
      RETURN
C
C*************************************************************************
C
C     REGION III - (Y<1, 0.4<Z<0.1) & (Y>4Z**.868, 0.1<Z<5).
C
C     THE NUMBER OF TERMS IN THE CONTINUED FRICTION WAS DETERMINED
C     EMPIRICALLY
C
300   CONTINUE
      N3=2+37.*EXP(-0.6*Y)
      IF(N3.GT.N3MAX) N3=N3MAX
      Q=CMPLX(Y,-X)
      A(N3)=0.5*N3/(N3*Z+Q)
      JN3 = N3-1
      DO 310 NJ = 1,JN3
      J = JN3 - (NJ-1)
      A(J)=0.5*J/(J*Z+Q+A(J+1))
310   CONTINUE
      A(1)=1./(Q+A(1))
      G=REAL(A(1))/RTPI
      RETURN
C
C*************************************************************************
      END
C
C
       SUBROUTINE CPF(X,Y,WR,WI)
C*************************************************************************
C
C     THIS ROUTINE COMPUTES THE REAL  (WR) AND IMAGINARY (WI) PARTS OF
C     THE COMPLEX PROBABILITY FUNCTION W(Z),
C
```

```
C     W(Z) = EXP(-Z*Z) * ERFC(-I*Z),
C
C     WHERE ERFC IS THE COMPLEMENTARY ERROR FUNCTION.  THE COMPUTATION
C     IS VALID FOR THE UPPER HALF PLANE OF Z = X + IY , IE Y > 0.
C
C     MAXIMUM RELATIVE ERROR FOR WR IS < 2.E-6 AND FOR WI IS < 5.E-6.
C
C     SUBROUTINE ADAPTED FROM:
C     J. HUMLICEK, J. QUANT. SPECTROSC. RADIAT. TRANSFER 21, 309 (1979)
C
C     PHILIP L. VARGHESE        <820315.0112>
C
C     SUBROUTINE MODIFIED TO ACCEPT LARGE POSITIVE OR NEGATIVE X VALUES.
C     THIS CHANGE (CHANGING EXP(X*X) TO DEXP(X*X)) WILL CAUSE A WARNING
C     WHEN COMPILED, BUT WILL EXECUTE CORRECTLY.
C
C     DAVID CLINE  6/6/86
C
C*********************************************************************
      DIMENSION T(6), C(6), S(6)
      DATA T/0.314240376E00,0.947788391E00,0.159768264E01,
     * 0.227950708E01,0.302063703E01,0.38897249E01/
      DATA C/0.101172805E01,-0.75197147E00,0.12557727E-01,
     * 0.100220082E-01,-0.242068135E-03,0.500848061E-06/
      DATA S/0.1393237E01,0.231152406E00,-0.155351466E00,
     * 0.621836624E-02,0.919082986E-04,-0.627525958E-06/
      WR=0.
      WI=0.
      Y1=Y+1.5
      Y2=Y1*Y1
C*********************************************************************
C
C     BRANCH TO REGION I OR II DEPENDING ON VALUES OF X AND Y.
C
C*********************************************************************
      IF(Y.GT.0.85.OR.ABS(X).LT.(18.1*Y+1.65)) GO TO 20
C*********************************************************************
C
C     CALCULATIONS FOR REGION II
C
      IF (ABS(X).LT.12.) WR=EXP(-X*X)
      Y3=Y+3.
      DO 10 I=1,6
      R=X-T(I)
      R2=R*R
      D=1./(R2+Y2)
      D1=Y1*D
      D2=R*D
      WR=WR+Y*(C(I)*(R*D2-1.5*D1)+S(I)*Y3*D2)/(R2+2.25)
      R=X+T(I)
      R2=R*R
      D=1./(R2+Y2)
      D3=Y1*D
      D4=R*D
      WR=WR+Y*(C(I)*(R*D4-1.5*D3)-S(I)*Y3*D4)/(R2+2.25)
10    WI=WI+C(I)*(D2+D4)+S(I)*(D1-D3)
      RETURN
C
C     END OF CALCULATIONS FOR REGION II
C
```

```
C*************************************************************************
C
C      CALCULATIONS FOR REGION I
C
20     DO 30 I=1,6
       R=X-T(I)
       D=1./(R*R+Y2)
       D1=Y1*D
       D2=R*D
       R=X+T(I)
       D=1./(R*R+Y2)
       D3=Y1*D
       D4=R*D
       WR=WR+C(I)*(D1+D3)-S(I)*(D2-D4)
30     WI=WI+C(I)*(D2+D4)+S(I)*(D1-D3)
       RETURN
C
C      END OF CALCULATIONS FOR REGION I
C
C*************************************************************************
       END
```

Fichier Multi.for

```
      PROGRAM MULTI
      IMPLICIT NONE
      CHARACTER*10    FILENAME, FILENAME2
      CHARACTER*1     REPONSE
      INTEGER         I,IOS,NBREBLOC,NTOT,CENTRE,LPROF,
     c                FONDSP(2048),SPECTRE(2048),REFERENCE(2048),
     c                SPECTRECAL(2048),K,J,IOFF,IOFFF,L,WHAT,
     c                BINF,BSUP, NBRERAIE, NBRERAIEREF,
     C                MAX, POSMAX,
     c                NUMRAIE(128), INTENSITE(128), NUMSPECTRE(128)
      REAL*8          LCELL,TEMPREF,TEMPGAZ,PAS,MASSE,CORR,
     c                PROF(2048),FGDOPTH,ERROR,LGR,LGSP,LAPP,LLSP,
     c                FLARGEUR,LAMBDA,AQ,XV,YV,SIGMA,FVOIGT,
     c                PROF2(2048),PROFVG(2,2048),X1,X2,DX,A0,AD,EPSILON,
     c                XX,YY,ZZ,WW,TOL,FLDOPPLER, SIGMA0(10), POSREF(10),
     c                POSITION(128), FREQUENCE(128), GDOPPLER(10),
     c                SIGMAREL(10)
      LOGICAL         IMPRIME, MANUAL

      COMMON          /XX/CENTRE, PAS, LCELL, TOL, MANUAL
      COMMON          /YY/BINF, BSUP, NTOT
      COMMON          /ZZ/PROF, GDOPPLER, SIGMAREL, NBRERAIEREF

      EXTERNAL        PROFIL, FLDOPPLER, INVERSEPROFIL

C
C   VARIABLES UTILISEES PAR LES ROUTINES IMSL
C
      INTEGER         NMAX,MMAX
      PARAMETER       (NMAX = 20, MMAX = 2048)
      INTEGER         M,N,IPARAM(6),LDFJAC
      REAL*8          XGUESS(NMAX),XSCALE(NMAX),FSCALE(MMAX),RPARAM(7),
     c                X(NMAX),FVEC(MMAX),FJAC(MMAX,NMAX)
      EXTERNAL        DUNLSF, GAUSSMULTI, VOIGTMULTI, FVOIGT
C
C   DEFINITION DES CONSTANTES
C
      REAL*8          PI,COEF
      PARAMETER       (PI = 3.141592654,
     c                 COEF = 0.4697186)

      WRITE(*,*) 'NOM DU FICHIER ? '
      READ(*,'(A)') FILENAME
      WRITE(*,*) 'RESULTATS IMPRIMES DANS UN FICHIER ? (Y/N)'
      READ(*,'(A1)') REPONSE
      IF ((REPONSE.EQ.'Y').OR.(REPONSE.EQ.'y')) THEN
          IMPRIME = .TRUE.
          WRITE(*,*) 'NOM DU FICHIER RESULTAT ?'
          READ(*,'(A)') FILENAME2
          OPEN(30,IOSTAT=IOS,ERR=9999,FILE=FILENAME2,ACCESS=
     c    'SEQUENTIAL',STATUS='NEW')
      ENDIF
      OPEN(10,IOSTAT=IOS,ERR=9999,FILE=FILENAME,ACCESS='SEQUENTIAL',
     c     STATUS='OLD')
      FILENAME(1:1) = 'T'
      OPEN(20,IOSTAT=IOS,ERR=9999,FILE=FILENAME,ACCESS='SEQUENTIAL',
```

```
c     STATUS='NEW')
      READ(10,*)
      READ(10,'(61X,I2)')NBREBLOC
      NTOT = NBREBLOC * 128
      DO 5 I = 1,16
5     READ(10,*)
      READ(10,'(56X,F7.4,//////////)') LCELL          ! EN METRES
      READ(10,'(58X,F5.2,/)') TEMPREF                 ! EN DEGRES
      READ(10,'(58X,F5.2,////)') TEMPGAZ              ! EN DEGRES
      READ(10,*)(FONDSP(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(SPECTRE(I),I=1,NTOT)
      READ(10,'(///)')
      READ(10,*)(REFERENCE(I),I=1,NTOT)
      READ(10,'(/////)')
      DO 7 I=1,127
7     READ(10,'(I5,F10.2,F12.5,I5,I6)',IOSTAT=IOS,ERR=9999,END=6)
     cNUMRAIE(I), POSITION(I), FREQUENCE(I), INTENSITE(I), NUMSPECTRE(I)
6     NBRERAIE = I-1

      WRITE(20,'('' RECORD  1'')')
      DO 101 J=0,NBREBLOC-1
              IOFF = J*128
              WRITE(20,'(A)')
         DO 101 K=0,15
                 IOFFF = K*8
101              WRITE(20,'(8I6)')(SPECTRE(IOFF+IOFFF+I),I=1,8)
C
C   CONVERSION DES UNITES
C
      TEMPREF = 273.16 + TEMPREF          ! EN KELVIN
      TEMPGAZ = 273.16 + TEMPGAZ          ! EN KELVIN
      LCELL = LCELL * 100.0               ! EN CM
C
C   INITIALISATION DES VARIABLES
C
      IF (NBREBLOC.EQ.8) THEN
           CENTRE = 512
      ELSE
           CENTRE = 1024
      ENDIF
      PAS = 0.0001
C
C   LECTURE DES PARAMETRES AU TERMINAL
C
      WRITE(*,*)' FACTEUR DE CORRECTION SUR LE FOND ? '
      READ(*,*) CORR
4     WRITE(*,*)' DETERMINATION DES BORNES POUR LE FIT :'
      WRITE(*,*)'    1. MANUELLE'
      WRITE(*,*)'    2. AUTOMATIQUE'
      READ(*,*) WHAT
      IF (WHAT.EQ.1) THEN
         MANUAL = .TRUE.
         WRITE(*,*)' BORNE INFERIEURE ?'
         READ(*,*) BINF
         WRITE(*,*)' BORNE SUPERIEURE ?'
         READ(*,*) BSUP
      ELSE
      IF (WHAT.EQ.2) THEN
         MANUAL = .FALSE.
```

```
         WRITE(*,*)' INTENSITE DANS LES AILES ( % par rapport au
     ccentre : 0 - 1) ?'
         READ(*,*) TOL
       ELSE
         GOTO 4
       ENDIF
       ENDIF
C
C    CORRECTION DU FOND
C
       DO 133 I = 1,NTOT
133    FONDSP(I) = FONDSP(I) * CORR

C
C      RECHERCHE DES RAIES DOPPLER
C
       N = 0
       POSMAX = 1
       MAX = INTENSITE(1)
       DO 109 I=1,NBRERAIE
       IF ((NUMSPECTRE(I).EQ.3).AND.(POSITION(I).GT.BINF).AND.
      c      (POSITION(I).LT.BSUP).AND.(INTENSITE(I).GT.0)) THEN
         N = N+1
         POSREF(N) = POSITION(I)
         SIGMA0(N) = FREQUENCE(I)
         IF (INTENSITE(I).GT.MAX) THEN
           POSMAX = N
           MAX = INTENSITE(I)
         ENDIF
       ENDIF
109    CONTINUE
       NBRERAIEREF = N

C
C      CALCUL DES SIGMA0 RELATIFS DES RAIES DOPPLER
C
       DO 111 I=1,NBRERAIEREF
          SIGMAREL(I) = SIGMA0(POSMAX)-SIGMA0(I)
111    CONTINUE


C
C      FIT D'UNE SOMME DE GAUSSIENNE SUR LA REFERENCE
C
       CALL PROFIL(FONDSP, REFERENCE, PROF, LPROF)
       N = 1
       DO 10 I=1,NBRERAIEREF
         J = INT(POSREF(I))
         XGUESS(N) = SIGMA0(I)
         XGUESS(N+1) = PROF(J)
         XGUESS(N+2) = FLDOPPLER(J)
         print *,xguess(N),xguess(N+1),xguess(N+2)
         write(30,*)xguess(N),xguess(N+1),xguess(N+2)
         N = N+3
10     CONTINUE
       N = N-1
       M = LPROF
       LDFJAC = M
       DO 50 I=1,N
50     XSCALE(I) = 1.0
```

```
      DO 60 I=1,M
60    FSCALE(I) = 1.0
      IPARAM(1) = 0
      CALL DUNLSF(GAUSSMULTI,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM, X, FVEC, FJAC, LDFJAC)
      DO 65 I=1,N/3
65    GDOPPLER(I) = X((I-1)*3+3)
      print *
      write(30,*)
      do 70 I=1,N,3
      write(30,*)x(I),x(I+1),x(I+2)
70    print *,x(I),x(I+1),x(I+2)
      error = 0.0
      do 71 i=1,M
71    error = error + fvec(i)**2
      error = dsqrt(error)/float(m)
      print *,error
      write(30,*)error

C
C     CALCUL DU PROFIL SUR BASE DES PARAMETRES FITTES
C
      DO 145 I=1,NTOT
        SIGMA = FLOAT(I-NTOT/2) * PAS
        PROF(I) = 0
        DO 145 J=1,N,3
           PROF(I) = PROF(I) + X(J+1)*EXP(-LOG(2.0)*
     c               ((SIGMA-X(J))/X(J+2))**2)
145   CONTINUE
      CALL INVERSEPROFIL(FONDSP,SPECTRECAL)
      WRITE(20,'(//'' RECORD  2'')')
      DO 146 J=0,NBREBLOC-1
             IOFF = J*128
             WRITE(20,'(A)')
         DO 146 K=0,15
                IOFFF = K*8
146             WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C     FIT D'UNE SOMME DE VOIGT SUR LE SPECTRE
C
      CALL PROFIL(FONDSP, SPECTRE, PROF, LPROF)
      XGUESS(1) = SIGMA0(POSMAX)
      N = 2
      DO 11 I=1,NBRERAIEREF
        J = INT(POSREF(I))
        XGUESS(N) = 0.001
        XGUESS(N+1) = PROF(J)
        print *,xguess(N),xguess(N+1)
        write(30,*)xguess(N),xguess(N+1)
        N = N+2
11    CONTINUE
      N = N-1
      M = LPROF
      LDFJAC = M
      DO 51 I=1,N
51    XSCALE(I) = 1.0
      DO 61 I=1,M
61    FSCALE(I) = 1.0
      IPARAM(1) = 0
```

```
      CALL DUNLSF(VOIGTMULTI,M,N,XGUESS,XSCALE,FSCALE,IPARAM,
     c            RPARAM, X, FVEC, FJAC, LDFJAC)
      print *,x(1)
      do 72 I=2,N-1,2
      write(30,*)x(I),x(I+1)
72    print *,x(I),x(I+1)
      error = 0.0
      do 73 i=1,M
73    error = error + fvec(i)**2
      error = dsqrt(error)/float(m)
      write(30,*)error
      print *,error

C
C     CALCUL DU PROFIL SUR BASE DES PARAMETRES FITTES
C
      DO 143 I=1,NTOT
        SIGMA = FLOAT(I-NTOT/2) * PAS
        PROF(I) = 0
        DO 143 J=1,NBRERAIEREF
           XV = SQRT(LOG(2.0))*(SIGMA-(X(1)-SIGMAREL(J)))/GDOPPLER(J)
           YV = SQRT(LOG(2.0)) * X(J*2) / GDOPPLER(J)
           PROF(I) = PROF(I) + X(J*2+1)*FVOIGT(XV,YV)
143   CONTINUE
      CALL INVERSEPROFIL(FONDSP,SPECTRECAL)
      WRITE(20,'(//'' RECORD  3'')')
      DO 144 J=0,NBREBLOC-1
            IOFF = J*128
            WRITE(20,'(A)')
       DO 144 K=0,15
               IOFFF = K*8
144            WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)

C
C     CALCUL DU PROFIL DE CHACUNE DES RAIES
C

      DO 246 J=1,NBRERAIEREF
        DO 245 I=1,NTOT
           SIGMA = FLOAT(I-NTOT/2) * PAS
           PROF(I) = 0
           XV = SQRT(LOG(2.0))*(SIGMA-(X(1)-SIGMAREL(J)))/GDOPPLER(J)
           YV = SQRT(LOG(2.0)) * X(J*2) / GDOPPLER(J)
           PROF(I) = PROF(I) + X(J*2+1)*FVOIGT(XV,YV)
245     CONTINUE
        CALL INVERSEPROFIL(FONDSP,SPECTRECAL)
        WRITE(20,'(//'' RECORD  '',I2)')3+J
        DO 247 L=0,NBREBLOC-1
              IOFF = L*128
              WRITE(20,'(A)')
         DO 247 K=0,15
                 IOFFF = K*8
247              WRITE(20,'(8I6)')(SPECTRECAL(IOFF+IOFFF+I),I=1,8)
246   CONTINUE



9999  STOP
      END
```

```
C
C     PROCEDURE PROFIL
C
      SUBROUTINE PROFIL(I0,I,K,G)

      IMPLICIT NONE
      INTEGER  I0(2048), I(2048), G, C, J, BINF, BSUP, CENTRE
      REAL*8   LCELL ,K(2048), TOL, PAS
      LOGICAL  MANUAL
      COMMON   /XX/CENTRE, PAS, LCELL, TOL, MANUAL
      COMMON   /YY/BINF, BSUP
C
      C = CENTRE
      IF (MANUAL.EQ.(.FALSE.)) THEN
         K(C) = LOG(FLOAT(I0(C))/FLOAT(I(C)))/LCELL
         DO 10 J = 1,C-1
         K(C+J) = LOG(FLOAT(I0(C+J))/FLOAT(I(C+J)))/LCELL
         K(C-J) = LOG(FLOAT(I0(C-J))/FLOAT(I(C-J)))/LCELL
10       IF (((I0(C+J)-I(C+J)).LT.TOL*(I0(C)-I(C))).OR.
     c    ((I0(C-J)-I(C-J)).LT.TOL*(I0(C)-I(C)))) GOTO 20
20       G = 2*J+1
         BINF = C-J
         BSUP = C+J
      ELSE
         DO 15 J = BINF, BSUP
15       K(J) = LOG(FLOAT(I0(J))/FLOAT(I(J)))/LCELL
         G = BSUP - BINF + 1
      ENDIF
      END

C
C   PROCEDURE INVERSEPROFIL
C
      SUBROUTINE INVERSEPROFIL(I0,I)
      IMPLICIT NONE
      INTEGER    I0(2048),I(2048),J,BINF,BSUP,NTOT,CENTRE
      REAL*8     PROF(2048), LCELL, PAS

      COMMON          /XX/CENTRE, PAS, LCELL
      COMMON          /YY/BINF, BSUP, NTOT
      COMMON          /ZZ/PROF
C
      DO 10 J=1,NTOT
c10    I(J) = FLOAT(I0(J))/EXP(LCELL*PROF(J))
10    I(J) = FLOAT(I0(J))/EXP(PROF(J)*LCELL)
      END


C
C     PROCEDURE FLDOPPLER
C
      REAL*8 FUNCTION FLDOPPLER(WHERE)
      IMPLICIT NONE
      INTEGER    CENTRE, I, SIGMA1, SIGMA2, WHERE
      REAL*8     PROF(2048), TMP1, TMP2, PAS
      COMMON     /XX/CENTRE, PAS
      COMMON     /ZZ/PROF

      SIGMA1 = 0.0
      SIGMA2 = 0.0
```

```
      DO 10 I=1,CENTRE/10
      IF ((PROF(WHERE-I).GE.PROF(WHERE)/2.0).OR.(SIGMA1.NE.0.0))
     c    GOTO 20
      SIGMA1 = WHERE-I
20    IF ((PROF(WHERE+I).GE.PROF(WHERE)/2.0).OR.(SIGMA2.NE.0.0))
     c    GOTO 10
      SIGMA2 = WHERE+I
10    CONTINUE
      TMP1 = (PROF(WHERE)/2.0-PROF(SIGMA1))/(PROF(SIGMA1+1)
     c        - PROF(SIGMA1))
      TMP2 = (PROF(WHERE)/2.0-PROF(SIGMA2))/(PROF(SIGMA2)
     c        - PROF(SIGMA2-1))
      FLDOPPLER = ABS(SIGMA1+TMP2-(SIGMA2+TMP1))*PAS/2.0
      END

C
C     PROCEDURE GAUSSMULTI
C
      SUBROUTINE GAUSSMULTI(M,N,X,F)
      IMPLICIT NONE
      INTEGER  I, J, M, N, CENTRE, BINF, BSUP
      REAL*8   PROF(2048), PAS, SIGMA, X(N), F(M)

      COMMON          /XX/CENTRE, PAS
      COMMON          /YY/BINF, BSUP
      COMMON          /ZZ/PROF

      REAL*8    COEF
      PARAMETER (COEF = 0.4697186)

      DO 10 I=BINF,BSUP
        SIGMA = FLOAT(I-CENTRE)*PAS
        F(I+1-BINF) = 0
        DO 20 J=1,N,3
           F(I+1-BINF) = F(I+1-BINF)+X(J+1)*EXP(-LOG(2.0)*
     c                   ((SIGMA-X(J))/X(J+2))**2)
20      CONTINUE
        F(I+1-BINF) = PROF(I) - F(I+1-BINF)
10    CONTINUE
      END

C
C     PROCEDURE VOIGTMULTI
C
      SUBROUTINE VOIGTMULTI(M,N,X,F)
      IMPLICIT NONE
      INTEGER    M,N,I,J,LPROF,K,CENTRE,BINF,BSUP,NBRERAIEREF
      REAL*8     PROF(2048), X(N), F(M), XV, YV, FVOIGT,
     c           SIGMA, PAS, RACINE2, GDOPPLER(10), SIGMAREL(10)
      EXTERNAL FVOIGT
      COMMON          /XX/CENTRE, PAS
      COMMON          /YY/BINF, BSUP
      COMMON          /ZZ/PROF, GDOPPLER,SIGMAREL,NBRERAIEREF

      RACINE2 = SQRT(LOG(2.0))
      DO 10 I=BINF,BSUP
        SIGMA = FLOAT(I-CENTRE) * PAS
        F(I+1-BINF) = 0
        DO 20 J=1,NBRERAIEREF
           XV = RACINE2 * (SIGMA-(X(1)-SIGMAREL(J))) / GDOPPLER(J)
```

```
            YV = RACINE2 * X(J*2) / GDOPPLER(J)
            F(I+1-BINF) = F(I+1-BINF) + X(J*2+1)*FVOIGT(XV,YV)
20        CONTINUE
          F(I+1-BINF) = PROF(I) - F(I+1-BINF)
10      CONTINUE
c        print *,x(1),x(2),x(3),x(4),x(5),x(6)
        END

C
C     FONCTION FVOIGT
C
        REAL*8 FUNCTION FVOIGT(XV, YV)
        IMPLICIT NONE
        DIMENSION A(0:6),B(0:6)
        COMPLEX*16 ZV, BUFF
        INTEGER M, N
        REAL*8  XV, YV, A, B

        DATA A/122.607931777104326,214.382388694706425,
       .181.928533092181549,93.155580458138441,30.180142196210589,
       .5.912626209773153,0.564189583562615/
        DATA B/122.607931773875350,352.730625110963558,
       .457.334478783897737,348.703917719495792,170.354001821091472,
       .53.992906912940207,10.479857114260399/

        ZV = DCMPLX(YV,-XV)
        BUFF=((((((A(6)*ZV+A(5))*ZV+A(4))*ZV+A(3))*ZV+A(2))*ZV+A(1))*
       cZV+A(0))
       c/((((((ZV+B(6))*ZV+B(5))*ZV+B(4))*ZV+B(3))*ZV+B(2))*ZV+B(1))*
       cZV+B(0))
        FVOIGT = REAL(BUFF)
        END
```