Calcul flottant au standard IEEE en Pascal et applications

Raymond Pavec
Université de Bretagne Occidentale
Laboratoire de Mathématiques
Brest, France

Résumé

L'initiation à la programmation en premier cycle des universités scientifiques se fait la plupart du temps en Turbo Pascal [22] sur micro-ordinateur de type PC. En ce qui concerne le calcul en virgule flottante les possibilités de ce langage sont réduites au strict minimum. Nous présentons une bibliothèque de procédures conforme aux spécifications du standard IEEE [1], permettant le controle de l'arrondi et des exceptions, ainsi que quelques applications de cette bibliothèque :

• une bibliothèque de calcul flottant, complexe, rationnel, d'intervalles et de mise en œuvre de la méthode CESTAC [5, 21, 24], utilisable en Pascal,

• une simple calculette programmable permettant d'expérimenter ces différents modes de calculs, et fonctionnant en "mode texte",

• et un logiciel graphique interactif fonctionnant sur micro-ordinateur de type Macintosh destiné à l'expérimentation des méthodes élémentaires d'analyse numérique et à l'illustration graphique de cours de mathématiques.

1 Calcul flottant au standard IEEE

Sur micro-ordinateur de type Macintosh, les fonctionnalités du standard de calcul flottant IEEE sont apportées par la bibliothèque S.A.N.E. (Standard Apple Numeric Environment [2]). La bibliothèque vf_IEEE, apporte des fonctionnalités analogues au langage Turbo-Pascal [22] sur micro-ordinateur de type PC. Cette bibliothèque, écrite partiellement en assembleur (codes F... : FLD, FSTP, FXAM... [7]), utilise directement les opérations "virgule flottante" disponibles sur ces machines.
L'annexe 1 présente la liste des types, fonctions et procédures de la bibliothèque "vf_IEEE".

1.1 Nombres flottants dans le standard IEEE, classes

En plus des nombres binaires flottants normalisés usuels de la forme :

$$\pm 1, a_2 a_3 \dots a_t 2^e,\ E_{min} \le e \le E_{max},\ a_i = 0 \text{ ou } 1$$

le standard IEEE introduit des représentations de ± 0, de $\pm\infty$, de nombres non valides (NaN, Not a Number) et de nombres dénormalisés :

$$\pm 0, a_2 a_3 \dots a_t 2^e,\ \text{avec } e = E_{min}$$

Tous ces nombres se comportent de manière cohérente dans les opérations : par exemple si un opérande est un NaN il en est de même pour le résultat...
Sur micro-ordinateur on dispose de 3 précisions de stockage : les deux formats de base `single` et `double` et un format étendu :

Nom courant	Nom en Pascal	Nom IEEE	t	$[E_{min}, E_{max}]$	Octets
simple	single	single	24	[-126,127]	4
double	double	double	53	[-1022,1023]	8
étendu	extended	double-extended	64	[-16382,16383]	10

Nombres flottants en Pascal

Le type single-extended du standard IEEE n'existe pas en Pascal.
Un nombre flottant rentre dans l'une des six classes de type `NumClass` défini en Pascal par :

```
NumClass = (SNaN, QNaN, Infinite, ZeroNum, NormalNum, DenormalNum);
```

Les fonctions `ClassSingle, ClassDouble` et `ClassExtended` donne la classe d'un nombre donné.

1.2 Rangement en mémoire

Quelle que soit la précision, un nombre flottant comporte trois parties : signe ($s = \pm$), exposant (e), mantisse ($a_1 a_2 \dots a_t$), rangées dans cet ordre en mémoire ([1]). Aussi bien pour l'exposant que pour la mantisse on progresse des bits de poids fort vers les bits de poids faible.
Dans certaines manipulations de données il est utile de pouvoir lire ou modifier directement le dernier bit de la mantisse a_t. Il faut alors tenir compte du fait que dans l'environnement PC dans lequel les machines sont équipés de microprocesseurs Intel, adresses et poids des octets varient dans le même sens (mode little-endian[7]), et en sens contraire pour l'environnement Macintosh (mode big-endian[2]).

Adresses sur PC	$m+p$	$\leftarrow$	$\leftarrow$	$\leftarrow$	m
Nombre flottant	s	e	$a_1 a_2$	$\dots$	a_t
Adresses sur Macintosh	m	$\rightarrow$	$\rightarrow$	$\rightarrow$	$m+p$

Nombre flottant à l'adresse m

Le tableau ci-dessus montre la configuration en mémoire d'un nombre flottant occupant $p = 4, 8$, ou 10 octets, commençant à l'adresse m. Le dernier octet de la mantisse est à l'adresse m sur PC et $m + p - 1$ sur Macintosh.

1.3 Opérations et erreurs d'arrondi

Les opérations élémentaires sont effectuées en pécision étendue, et donnent le meilleur résultat possible compte tenu de la direction d'arrondi en vigueur.

[1] Le premier chiffre a_1, égal à 0 ou 1, n'est effectivement stocké qu'en précision étendue.

Ceci peut être modélisé à l'aide de la fonction d'arrondi fl associant à tout x réel le nombre flottant le plus proche([2]).
Si $\mathrm{fl}(x)$ est de la classe `NormalNum` on a :

$$\mathrm{fl}(x) = x(1+\varepsilon), \quad \frac{1}{1+eps} \leq 1+\varepsilon \leq 1+eps$$

La précision-machine *eps* est égale à 2^{-64} dans le cas de l'arrondi au plus près, et à 2^{-63} dans les autres cas.
Soit $\top$ l'une des opérations usuelles ($\top = +, -, \times, /$), et $\tilde{\top}$ sa réalisation en calcul flottant. Si x et y désignent deux nombres flottants on a par définition : $x\tilde{\top}y = \mathrm{fl}(x\top y)$. Si $x\tilde{\top}y$ est de la classe `NormalNum` on a donc :

$$x\tilde{\top}y = x\top y(1+\varepsilon), \quad \frac{1}{1+eps} \leq 1+\varepsilon \leq 1+eps, \quad eps = 2^{-64}$$

L'arrondi de la précision étendue à la précision simple ou à la précision double, se modélise de la même façon, *eps* étant remplacé par $eps_1 = 2^{-24}$ ou $eps_2 = 2^{-53}$, tant que n'interviennent que des nombres de classe `NormalNum`.

1.4 Exceptions

En Pascal standard une division par zéro, un dépassement de capacité ou une opération invalide se traduit en général par l'arrêt immédiat du programme. La bibliothèque vf_IEEE offre d'autres possibilités. On dispose des constantes :

```
invalid = 1;
denormal = 2;
divzero = 4;
overflow = 8;
underflow = 16;
inexact = 32;
```

et de la procédure `SetHalt` pour fixer les circonstances dans lesquelles on désire obtenir un arrêt du programme. L'argument de cette procédure doit être une somme de ces constantes, ou 0, auquel cas le calcul se déroule toujours sans interruption.
L'initialisation effectuée par la bibliothèque vf_IEEE appelle `SetHalt(0)`, c'est-à-dire que toutes les exceptions sont masquées, conformément aux recommendations du standard IEEE. On notera que dans ces conditions l'apparition d'un NaN ne peut passer inaperçu puisque les opérations faisant intervenir un NaN donnent à leur tour un NaN, ce qui n'est pas le cas pour un Overflow ou une division par 0 [20].
On retrouvera l'environnement standard du Pascal par `SetHalt(divzero + overflow)`.

1.5 Environnement de calcul

L'environnement de calcul comprend la précision (simple, double, étendue) et la direction de l'arrondi (au plus près, par excès, par défaut, vers 0) et s'applique aux opérations usuelles et à la racine carrée.

[2]Dans le cas de l'arrondi au plus près, lorsque x est à égale distance de deux nombres flottants m_1 et m_2, $\mathrm{fl}(x)$ retourne celui dont le dernier chiffre a_t vaut 0.

Les types Pascal utilisés sont :

```
RoundDir = (ToNearest, Upward, Downward, TowardZero);
RoundPre = (ExtPrecision, DblPrecision, RealPrecision);
```

et on dispose des procédures ou fonctions `SetRound, GetRound, SetPrecision, GetPrecision`. Normalement l'arrondi est effectué au plus près et la précision de l'arrondi est la précision étendue.
Les procédures `ProcEntry` et `ProcExit`, utilisées par exemple au début et à la fin d'une procédure, permettent d'installer l'environnement de calcul recommandé par le standard IEEE (exceptions masquées, arrondi au plus près, précision étendue) puis de rétablir l'environnement de calcul d'origine.

1.6 Fonctions élémentaires

1.6.1 Comparaison

La comparaison de deux nombres flottants x et y doit tenir compte des réels non valides. Un flottant non valide ne peut évidemment être comparé à aucun autre nombre flottant. On se trouve toujours dans l'un des quatre cas suivants :

- x et y sont non comparables
- $x < y$
- $x = y$
- $x > y$

Dans vf_IEEE on introduit un type énuméré comportant ces quatre cas :

```
RelOp = (GreaterThan, LessThan, EqualTo, Unordered);
```

et on dispose de la fonction `Relation` :

Function `Relation (x: extended; y: extended): RelOp;`

qui retourne `r` tel que `x r y` soit vrai.
Exemple. Dans l'instruction Pascal :

```
if x > y then
   traitement 1
else
   traitement 2
```

le traitement effectué n'est pas clair dans le cas ou x ou y est un NaN. On écrira plutôt, `r` désignant une variable de type `Relop` :

```
r := Relation(x, y);
if r=GreaterThan then
   traitement 1
else if (r=lessThan)or(r=EqualTo) then
   traitement 2
else
   traitement 3
```

1.6.2 Autres fonctions

Fonctions logb et scalb
`logb(x)` donne l'exposant de l'écriture normalisée en base 2 de son argument. `scalb(x, n)` donne le produit de x par 2^n.
Fonction remainder
`r = XremY(x,y)` donne le reste de la division entière de x par y. `r` est donc tel qu'il existe $k \in \mathcal{Z}$ vérifiant :

$$x = ky + r, 0 \leq r < y$$

Cette fonction ne provoque pas d'erreur d'arrondi. Elle est utilisée pour les calculs de pgcd dans les opérations rationnelles.
Fonctions exp1 et ln1
`exp1(x)` donne une évaluation numériquement stable de $e^x - 1$ et `ln1(x)` de $\ln(1+x)$, ce qui n'est pas le cas au voisinage de 0 lorsqu'on utilise directement les fonctions exp et ln.
Fonction Rint
Cette fonction effectue l'arrondi de son argument à l'entier le plus proche, suivant la direction d'arrondi choisie.
Fonction RandomX
Générateur de nombres au hasard. La formule appliquée est : $7^5 x \bmod (2^{31} - 1)$. Pour utiliser cette fonction il faut initialiser une variable globale `x` à l'aide d'un nombre entier compris entre 1 et $2^{31} - 2$ (par exemple à l'aide de l'horloge), puis faire des appels successifs de `RandomX(x)`.

1.7 Exemple d'utilisation : méthode de bissection

Etant donnée une matrice tridiagonale symétrique, la méthode de bissection permet d'encadrer les valeurs propres par une méthode de dichotomie. La mise en œuvre proposée dans [18] (qui garantit le bon déroulement de l'algorithme et fournit des bornes d'erreur rigoureuses) s'appuie sur le standard IEEE : distinction entre nombres normalisés et dénormalisés, représentation de l'infini, direction de l'arrondi (arrondi par excès pour le calcul de bornes d'erreur rigoureuses).
La matrice tridiagonale symétrique considérée est d'abord mise à l'échelle par multiplication par une puissance de 2 choisie de manière à rendre tous les éléments inférieurs à 1. Les fonctions `logb` et `scalb` permettent de faire facilement cette transformation.
Si on note $(a_i)_{i=1\ldots n}$ les élements diagonaux de la matrice et $(b_i)_{i=2\ldots n}$, les éléments de la sous-diagonale, l'algorithme est basé sur le calcul de la suite :

$$p_k := \begin{cases} a_k - x \quad \text{si} \quad b_k = 0 \quad \text{ou} \quad p_{k-1} = -\infty \\ \\ -\infty \quad \text{si} \quad p_{k-1} = 0 \\ \\ a_k - x - \dfrac{b_k^2}{p_{k-1}} \quad \text{sinon} \end{cases} \qquad \text{pour } k = 1, \ldots, n$$

avec $p_0 = 1, b_1 = 0$ et $x \in [-3, 3]$.
Un dépassement de capacité est évité en remplaçant par 0 les nombres dénormalisés pouvant apparaître dans les calculs. Le plus petit nombre normalisé est $m = 2^{E_{min}}$, $E_{min} = -16382$ et le plus grand $M = 1,11\ldots 1 \times 2^{E_{max}}$, $E_{max} = 16383$. L'expression $a_k - x - b_k^2/p_{k-1}$ n'est alors utilisée que pour $|p_{k-1}| \geq m$ et il est facile de voir que ce calcul donne un résultat inférieur à M en valeur absolue.

Cette modification est bien sur prise en compte dans l'analyse d'erreur : remplacer par 0 un nombre dénormalisé provoque une erreur absolue majorée par $m = 2^{E_{min}}$. Cette erreur absolue s'interprète comme une modification de a_k.
Les autres erreurs d'arrondis se produisant dans le calcul s'interprètent également comme une modification de a_k et de b_k. On montre ainsi que le calcul effectué revient à remplacer la matrice A par une matrice $A+E_x$. Un majorant de la norme infinie de E_x est évalué au cours du calcul. On obtient des bornes rigoureuses pour les valeurs propres (ordonnées) $\lambda_i(A)$ de A en utilisant l'inégalité :

$$|\lambda_i(A) - \lambda_i(A + E_x)| \leq \|E_x\|_\infty$$

2 Application 1 : calcul complexe, rationnel, d'intervalles, stochastique

La bibliothèque vf_CQIS contient des types permettant de définir des nombres de type complexe, rationnel-flottant, intervalle, rationnel-intervalle et stochastique (2 à 5 composantes), et des procédures permettant de manipuler et d'effectuer des opérations sur ces nombres. L'annexe 2 donne la liste des déclarations contenues dans cette bibliothèque.
Les données sont contenues dans des variables Pascal de type `R2single`, `R2double` ou `R2` qui comportent deux composantes flottantes en précision simple, double ou étendue (sauf pour le type stochastique à partir de 3 composantes pour lesquelles le nombre de composantes est de 3, 4 ou 5).

```
R2single = array[1..2] of single;
R2double = array[1..2] of double;
R2 = array[1..2] of extended;
```

En simple ou double précision seuls des flottant valides (0, dénormalisé, normalisé) sont considérés. En précision étendue une donnée comportant une composante infinie ou NaN est considérée comme invalide. Dans les fonctions et procédures susceptibles de générer des résultats invalides, la validité des données et des résultats est controlée par un argument `var valide: boolean`. Aucun traitement n'est effectué lorsque `valide` est `false` en entrée, ou à partir du moment ou valide devient `false`.
Les fortes contraintes de type du langage Pascal dans le passage d'arguments sont contournées par l'utilisation systématique de pointeurs génériques tels que `ptr` qui désigne un pointeur sur un octet, ou `PR2` qui désigne un pointeur sur un élément de `R2`. L'opérateur `@` appliqué à une variable donne un pointeur sur cette variable, et la fonction `Pointer` donne un pointeur sur une adresse mémoire donnée.
Il est ainsi possible d'écrire et d'utiliser sans difficulté des procédures et fonctions opérant sur tous les types données définies ci-dessous.
Chaque type est reconnu par une constante spécifique : `coefR` (nombre flottant ordinaire), `coefC`, `coefQ`, `coefI`, `coefqI`, `coefS2`,. . . , `coefS5`.

2.1 Type complexe et type intervalle (`coefC, coefI`)

Les deux composantes sont respectivement la partie réelle et imaginaire du nombre complexe, ou la borne inférieure et supérieure de l'intervalle. Le type intervalle permet la mise en œuvre du calcul d'intervalles de Moore [13] en utilisant l'arrondi par défaut ou par excès. Les extensions de Kulisch [11] ne sont pas prise en compte.

2.2 Type rationnel-flottant (`coefQ`)

Dans ce type on représente :
• soit un rationnel p/q dont numérateur et dénominateur (> 0) restent dans un intervalle entier $[-max, max]$ dont tous les éléments entiers sont représentables exactement dans la précision considérée,
• soit un nombre flottant, dont le nombre de chiffres t dépend de la précision considérée.

Précision	max (p&q)	t(flottant)	octets
simple	$2^{31} - 1 \approx 2,1\ 10^9$	52	8
double	$2^{63} - 1 \approx 9,2\ 10^{18}$	53	16
étendue	$2^{64} - 1 \approx 1,8\ 10^{19}$	64	20

Caractéristiques du type rationnel-flottant `coefq`

Suivant la précision les deux cas sont distingués soit par la valeur de la deuxième composante ou par la valeur d'un seul bit. La fonction :

Function `Q_Flottant (px: PR2; precision: integer): boolean;`
indique dans quel cas se trouve la donnée pointée par `px`.

2.2.1 Précision étendue

Dans ce cas $max = m = 2^{64} - 1$. Une valeur flottante, en précision étendue, est représentée dans la première composante, la seconde étant mise à zéro. Lorsque la deuxième composante est non nulle le nombre représenté est un rationnel, le numérateur et dénominateur se trouvant respectivement dans la première et la deuxième composante.

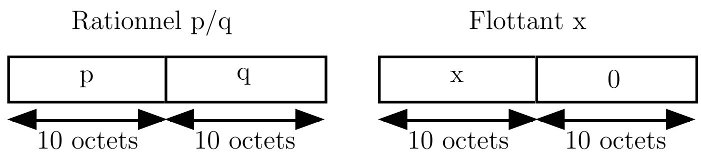


Figure 1 : Rationnel-flottant en précision étendue

Chaque composante occupe 10 octets. p, q et x sont représentés en virgule flottante et précision étendue. La fraction p/q doit être irréductible (voir 2.7.1) et $q > 0$.

2.2.2 Précision double

La représentation est analogue au cas précédent, mais avec $max = m_2 = 2^{63} - 1$. Le numérateur et le dénominateur sont représentés en complément à 2 sur 8 octets (type `comp` du Pascal) et sont convertis en flottant de précision étendue pour les opérations.

2.2.3 Précision simple

La place mémoire totale utilisée est de 8 octets. Un seul bit est utilisé pour reconnaître une représentation rationnelle d'une représentation flottante. La précision de la représentation flottante est donc toute proche de la précision double : 52 bits au lieu de 53.

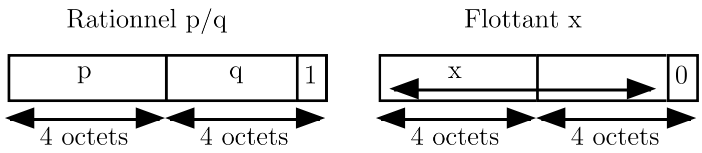


Figure 2 : Rationnel flottant en précision simple

Le bit utilisé pour distinguer les deux états est le dernier bit de la mantisse de x, qui est mis à 0. Le contenu de la mémoire est donc un nombre flottant en double précision directement utilisable.
Dans le cas d'une représentation rationnelle le numérateur occupe 32 bits ($max = m_1 = 2^{31} - 1$) et le dénominateur, qui est positif, en occupe 31. En mémoire, q est donc décalé à gauche de 1 bit, et le dernier bit ainsi libéré est mis à 1.
Le type pascal :

```
type double2 = record
          case integer of
          1 : (p, q : longint);
          2 : (x : double);
      end;
```

permet de manipuler sans difficulté ce type de données.

2.2.4 Opérations élémentaires

Les algorithmes utilisés pour les opérations usuelles sont ceux de Knuth [10]. Ces algorithmes utilisent l'addition, la soustraction et la multiplication d'entiers de l'intervalle $[-m, m]$, qui doivent être réalisées exactement : il suffit pour cela de choisir l'arrondi au plus près et de vérifier que le résultat de chaque opération appartient également à l'intervalle $[-m, m]$. En effet si x et y sont deux entiers appartenant à $[-m, m]$, soit $z = x \top y$, ou $\top$ est $+, -$ ou $*$, et z' le résultat obtenu en virgule flottante avec arrondi au plus près. Comme l'écriture en base 2 de m comporte 64 chiffres tous égaux à 1, le nombre machine suivant m est $m + 1 = 2^{64}$. Si $z' \in [-m, m]$ on a donc $z \in [-m - 1/2, m + 1/2]$, et donc $z \in [-m, m]$ puisque z est entier. Il en résulte $z = z'$.

2.2.5 Comparaisons

Plusieurs cas sont à considérer suivant la nature des deux éléments considérés :

• lorsque les deux nombres à comparer sont des rationnels p_1/q_1 et p_2/q_2 on compare d'abord les valeurs approchées $x_1 = \widetilde{p_1/q_1}$ et $x_2 = \widetilde{p_2/q_2}$. Si $x_1 \neq x_2$ le résultat est donné par la comparaison de x_1 et de x_2. Si $x_1 = x_2$ la comparaison est faite en utilisant le développement en fraction continue de p_1/q_1 et de p_2/q_2.

• dans les autres cas la comparaison est faite en flottant.

2.2.6 Erreurs d'arrondi

Les opérations élémentaires sont effectuées en précision étendue. Une opération donne un résultat flottant si l'un des opérandes est déjà un nombre flottant, ou si le numérateur ou le dénominateur dépasse $m = 2^{64} - 1$. La fonction d'arrondi au plus près fl, vers la précision étendue donne :

$$\mathrm{fl}(x) = x(1+\varepsilon), \quad \frac{1}{1+eps} \leq 1+\varepsilon \leq 1+eps, \quad eps = 2^{-64}$$

L'erreur absolue commise peut être majorée par $|x|eps$ ou $|fl(x)|eps$.
Désignons par $\top$ l'une des opérations élémentaires et par $\tilde{\top}$ sa réalisation en machine. Si x et y sont deux nombres du type `coefq` tels que $x\tilde{\top}y$ donne un résulat flottant normalisé, ce résulat peut s'écrire :

$$x\tilde{\top}y = \mathrm{fl}(\mathrm{fl}(x)\top\mathrm{fl}(y)) = (x(1+\varepsilon_1)\top y(1+\varepsilon_2))(1+\varepsilon_3)$$

avec ε_1 (resp. ε_2) nul si x (resp. y) est déjà un nombre flottant, ou satisfaisant aux inégalités ci-dessus.
Pour une multiplication donnant un résultat flottant on obtient donc :

$$x\tilde{\times}y = x \times y(1+\tau), \quad \frac{1}{(1+eps)^k} \leq 1+\tau \leq (1+eps)^k,$$

avec $k = 1$ si x et y sont déjà des nombres flottants, $k = 2$ si x ou y est rationnel, $k = 3$ si x et y sont rationnels. Au premier ordre l'erreur est donc majorée par $k\ eps\ x \times y$ ou $k\ eps\ x\tilde{\times}y$.
Pour une addition donnant un réultat flottant on peut écrire :

$$x + y = fl(x) + e_1 + fl(y) + e_2 = fl(fl(x) + fl(y)) + e_1 + e_2 + e_3$$

$$x + y = x\tilde{+}y + e_1 + e_2 + e_3$$

On a $e_1 = 0$ si x déjà un nombre flottant, ou $|e_1| \leq |fl(x)|eps$, $e_2 = 0$ si y déjà un nombre flottant, ou $|e_2| \leq |fl(y)|eps$, $|e_3| \leq |x\tilde{+}y|eps$.

Exemple : Considérons le calcul d'un produit scalaire $\sum_1^n x_i y_i$ où x_i et y_i sont des nombres rationnels-flottants en précision étendue. L'erreur totale est la somme des erreurs commises dans la multiplication $p_i = x_i\tilde{\times}y_i$ et dans l'addition de $s_i = s_{i-1}\tilde{+}p_i$. On obtiendra donc une majoration d'erreur au premier ordre en cumulant simultanément la somme des erreurs de multiplication et d'addition. Ces erreurs se majorent à l'aide des valeurs calculées en machine : $|s_{i-1}|, |s_i|, |p_i|$.

2.3 Type rationnel-intervalle (`coefQI`)

Ce type de données combine une représentation rationnelle p/q limitée à un intervalle d'entiers $[-m', m']$, $m' = 2^{63} - 1$, relayée par un intervalle $[x_1, x_2]$. La distinction entre les deux états est faite sur la valeur du dernier bit de la deuxième composante. Lorsque ce bit vaut 1 la valeur représentée est rationnelle, sinon il s'agit d'un intervalle. Ce bit étant le dernier bit de la borne supérieure de l'intervalle, on dispose donc respectivement de 63, 52 ou 23 bits pour cette borne supérieure, au lieu de 64, 53 ou 24 pour la borne inférieure.

Précision	max(p&q)	Précision des bornes	octets
simple	$2^{31}-1$	24, 23	8
double	$2^{63}-1$	53, 52	16
étendue	$2^{63}-1$	64, 63	20

Caractéristiques du type rationnel-intervalle `coefqi`

2.3.1 Précision étendue

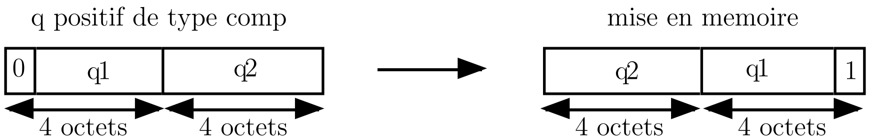
Chaque composante occupe 10 octets. p, q, x_1, x_2 sont des nombres flottants en précision étendue. Pour q et x_2 le dernier bit de la mantisse vaut respectivement 1 et 0. Pour obtenir la valeur de q à considérer il faut donc d'abord ce bit à 0. On accède à ce bit de manière différente suivant les machines (voir 2.1) :

• sur PC équipés d'un microprocesseur Intel, les octets sont rangés des poids faibles vers les poids forts. L'octet de poids faible de q se trouve à la même adresse que q.

• sur Macintosh au contraire les octets sont rangés des poids forts vers les poids faibles : l'octets de poids faible est donc à l'adresse de q augmentée de 9.

2.3.2 Précision double

Le numérateur et le dénominateur p et q de la représentation rationnelle sont codés en complément à 2 sur 64 bits (type `comp` du Pascal). q étant positif, seuls 63 bits sont nécessaires pour q. Le bit de signe (bit de poids fort) est donc disponible. Le bit utilisé pour distinguer la représentation rationnelle de la représentation flottante étant le dernier bit de la mantisse de la représentation flottante, q n'est donc pas stocké tel quel, mais de la manière indiquée par l'illustration ci-dessous :



Codage du dénominateur pour un rationnel de type `coefqi`

Les quatre octets (q_1) de poids fort et les quatre octets de poids faible (q_2) sont échangés, q_1 est décalé de 1 bit à gauche, puis le dernier bit est mis à 1.
La transformation inverse doit être faite pour obtenir la représentation correcte de q dans le type `comp`.
La programmation de ces transformations est la même dans les deux environnements (PC ou Macintosh), grâce à l'utilisation du type Pascal :

```
type comp2 = record
         case integer of
         1 : (q : comp);
         2 : (q1, q2 : longint);
      end;
```

et des fonctions de décalage.

2.3.3 Précision simple

Dans le cas rationnel p et q sont des entiers sur 4 octets de type `longint`. Ce cas est reconnu en mettant le dernier bit de q à 1, après avoir décalé q de 1 bit vers la gauche (voir le type `coefq`).

2.3.4 Conversion en rationnel ou en intervalle

Dans les opérations une telle donnée est convertie en rationnel flottant ou en intervalle suivant le cas. La fonction :

Function `QI_Intervalle (px: PR2; precision: integer): boolean;`
indique dans quel cas se trouve la donnee pointée par `px`. La procédure :

Procedure `QI_Vers_QouI (px: PR2; var coeff: integer);`
effectue la conversion sur place de la donnée pointée par `px` en rationnel ou en intervalle, `coeff` indiquant la conversion effectuée.

2.4 Types stochastiques `(coefS2,..., coefS5)`

Ces types sont destinés à la mise en œuvre synchrone de la méthode CESTAC de Vignes [5, 21, 24]. Le nombre N de composantes flottantes utilisées varie ici de 2 à 5, dans les trois précisions disponibles. Les opérations sont effectuées sur les N composantes, l'arrondi étant effectué au hasard, par défaut ou par excès.
Soit $X = (X_i)_{i=1\ldots N}$ une variable stochastique de N composantes. Si on désigne par $\bar{X}$ la moyenne et σ la variance de X, la loi de Student fournit, sous des hypothèses convenables [4, 5], pour une probabilité p donnée, un intervalle de confiance pour le résultat représenté par X sous la forme :

$$[\bar{X} - d, \bar{X} + d], \quad d = \frac{\sigma t_p}{\sqrt{N}}$$

Pour $p = 0{,}95$ on a :

N	2	3	4	5
$t_p/\sqrt{N}$	8,99	2,49	1,59	1,25

Une estimation du nombre C de chiffres décimaux corrects est alors obtenu en prenant la partie entière du logarithme décimal de $|\bar{X}|/d$.
Ces calculs sont faits par la procédure :

Procedure `MDC (px: PR5; coeff: integer; var resultats: R3);`
retourne dans `resultats` un élément de `R3` dont les composantes sont respectivement la moyenne $\bar{X}$, d et C, à la probabilité $p = 0{,}95$, de la donnée stochastique X pointée par `px`.

2.5 Opérations

Dans les procédures et fonctions ci-dessous, la validité des opérandes et des résultats est en général controlée par une variable booléenne `valide`. Aucun traitement n'est effectué à partir du moment où `valide` devient `false`.
Les opérations sont effectuées par la procédure `Operation` dont la déclaration en Pascal est :

Procedure `Operation(px, py, pz : PR2;`
```
        coeff : integer;
        op : integer;
        precision: integer;
        configuration : integer;
        var valide : boolean);
```

• `px`, `py` sont des pointeurs sur les opérandes, et `pz` un pointeur recevant le résultat,
• `coeff` indique le type considéré : `coefR, coefC, coefQ`, ...
• `op` désigne l'opération à effectuer : `xplusy, xmoinsy, xmuly, xdivy, xpwn, xly, xley, xgy, xgey, xeqy, xneqy, xandy, xory`
• `precision, configuration` : les opérandes et le résultat sont en précision étendue, sauf indication contraire. `precision` indique alors l'autre précision utilisée, et `configuration` est une somme de constantes (`pxReduit, pyReduit, pzReduit`) indiquant les éléments concernés par cette précision.
• `valide` indique en entrée si les données sont valides, et en sortie si l'opération a donné un résultat valide.
Si en entrée `valide` est `false` rien n'est fait, sinon l'exécution de la procédure `Operation` se décompose en trois parties :
• conversion des données en précision étendue, s'il y a lieu,
• réalisation de l'opération en précision étendue,
• conversion du résultat en précision simple ou double s'il y a lieu, et copie du résultat à l'adresse indiquée par `pz` si `valide` est toujours `true`.

La procédure :

Procedure `RacineCarree (px: PR2; coeff: integer; var valide: boolean);`
calcule la racine carrée de la donnée de précision étendue pointée par `px`.
La fonction :

Function `Module (px: PR2; coeff, precision: integer): extended;`
calcule le module de `px`, en considérant l'élémént de plus petit module possible dans le cas d'un intervalle ou d'un type stochastique. Cette fonction retourne donc 0 pour un intervalle contenant l'origine ou une variable stochastique représentant un 0 informatique [23].

2.6 Conversions

Les fonctions et procédures de conversion sont de plusieurs sortes :
• **changement de précision** : conversion de la simple ou double précision vers la précision étendue, et réciproquement, pour tous les types de données :

Procedure `Extension (source: PR2; destination: PR2;`
```
        Coeff: integer; precision: integer);
```
Procedure `Reduction (source: PR2; destination: PR2;`
```
        var valide: boolean; coeff: integer; precision: integer);
```
• **changement de type** :

Procedure `Conversion (source, destination: PR2;`
```
        typeS, typeD: integer);
```
Cette procédure s'applique à des données en précision étendue. `typeS` est le type de `source` et `typeD` est le type de `destination`.

• pour le type rationnel-intervalle, conversion en rationnel ou en intervalle suivant le cas (voir 2.3),

• **conversion en flottant** :

Function `ValeurApprochee (px: PR2;`
`coeff: integer; precision: integer): extended;`

• **conversion en chaine de caractères** :

La fonction :

Function `ConversionEnChaine (donnee: PR2; coeff, precision, chiffres: integer;`
`option: boolean): string;`

s'applique à tous les types et précisions. `chiffres` qui indique le nombre significatifs souhaités, est ignoré pour les rationnels et pour les types stochastiques. Dans ce dernier cas le nombres de chiffres estimés corrects est utilisé. On obtiendra par exemple :

`{11 à 95%}4,539 992 976 3 E-5+\9,73E-16`

qui indique 11 chiffres décimaux corrects, suivi de la valeur moyenne et de la demi-longueur de l'intervalle contenant le résultat. Les chiffres décimaux sont groupés par 3 lorsque `option` est `true`.

2.7 Initialisation

L'initialisation de variables de type flottant ou complexe, se fait directement en affectant des valeurs aux composantes. Dans la plupart des autres cas on pourra utiliser la fonction :

Procedure `Affectation (num, den: comp; destination: PR2;`
`coeff, precision: integer; var valide: boolean);`

Cette procedure affecte à `destination` la fraction `num/den`, dans le type `coeff` et à la `precision` indiquée, et retourne un booléen indiquant la validité de l'affectation. L'affectation est invalide si `den` est nul ou si la réduction à la précision indiquée provoque un dépassement de capacité. Le type `comp` est un type entier sur 64 bits. Le maximum autorisé est $2^{63} - 1 \approx 9,2 \times 10^{18}$.
Les paragraphes suivants indiquent comment faire directement une affectation.

2.7.1 Type rationnel-flottant (`coefQ`)

En précision étendue on affecte directement des valeurs aux deux composantes, en respectant les conventions. Il faut ensuite, dans le cas d'un rationnel (numérateur et dénominateur entiers $\leq 2^{64} - 1$), simplifier la fraction en utilisant la procédure :

Procedure `Simplifier (py: PR2);`

En précision simple ou double, il faut d'abord initialiser une variable en précision étendue, comme ci-dessus, puis utiliser la procédure `Reduction` de la section précédente.
La fonction :

Function `BonEntier (pe: PExt; precision: integer): boolean;`

peut également être utile : elle indique si le nombre flottant pointé par `pe` est un entier ne dépassant pas les bornes permises, suivant la précision.

2.7.2 Type intervalle (`coefI`)

Il est possible d'affecter directement des valeurs aux deux composantes, en s'assurant que la borne inférieure est inférieure ou égale à la borne supérieure.

Si la donnée est une fraction n/d ou une valeur décimale que l'on peut écrire n/d, avec n entier et $d = 10^k$, on pourra utiliser la function `Affectation`. Par exemple pour initialiser une variable `u, (type R2double)` en précision double au plus petit intervalle contenant 5,321456 on utilisera :

```
.. Affectation (5321456, 1000000, @u, coefI, precisionDouble)
```

Si la valeur 5,321456 représente un valeur approchée arrondie au plus près, c'est-à-dire comportant une erreur absolue majorée par $0,5 \times 10^{-6}$ on pourra écrire :

```
u[1] := 5.3214555; u[2] := 5.3214565
```

2.7.3 Type rationnel-intervalle (`coefQI`)

L'affectation directe n'est pas possible. On procédera comme dans le cas précédant en commençant par initialiser une variable de type `coefQ` ou `coefI`, suivant le cas, puis on effectuera la conversion de type, et la réduction de précision s'il y a lieu.

2.7.4 Types stochastiques (`coefS2`,. . . , `coefS5`)

Pour une valeur numérique donnée sous forme décimale, utiliser la fonction `Affectation` ou procéder comme dans le cas rationnel, puis faire la conversion : la fraction sera calculée en choisissant la direction de l'arrondi au hasard, pour chaque composante.
Pour une valeur numérique située dans un intervalle connu, initialiser d'abord une variable de type intervalle `coefI`, puis faire la conversion. Les composantes de la variable stochastique seront calculées au hasard dans l'intervalle prescrit, suivant une loi uniforme.

2.8 Application : résolution de systèmes linéaires

La bibliothèque `syslin` contient des procédures permettant d'effectuer la décomposition LU, avec pivot partiel, d'une matrice carrée A d'ordre n et la résolution des systèmes triangulaires associés à cette décomposition. Tous les types décrits ci-dessus sont considérés, dans les trois précisions de stockage.
Cette bibliothèque est conçue pour travailler sur une seule taille de matrice et de vecteur à la fois : matrice d'ordre n, vecteur de n éléments. En effet, les calculs d'adresse utilisent le nombre d'octets occupés par un élément et par une ligne de la matrice.

2.8.1 Allocation de mémoire

Les éléments de la matrice sont rangés consécutivement, ligne par ligne. On accède au premier élément à l'aide d'un pointeur de type `PR2`. Connaissant la place mémoire occupée par chaque élément il est alors facile de calculer un pointeur sur n'importe quel élément de la matrice.
On dispose des procédures Pascal :

```
Procedure Matrice (var A: PR2; n, coeff, precision: integer;
   var valide: boolean);
Procedure Vecteur (var b: PR2; n, coeff, precision: integer;
   var valide: boolean);
Function A0 (A: PR2; i, j: integer): PR2;
Function A1 (A: PR2; i, j: integer): PR2;
Function Dupliquer (donnee: PR2; elements: integer; var valide: boolean): PR2;
Procedure Dispose (var donnee: PR2; elements : integer);
```

La procédure `Matrice` alloue la mémoire nécessaire au stockage d'une matrice carrée $n \times n$, de type `coeff`, et à la `precision` indiquée. Cette procédure initialise également des variables contenant le nombre d'octets de mémoire occupés par un élément de la matrice et par une ligne de la matrice.
La procédure `Vecteur` alloue un vecteur de n composantes destiné à contenir un second membre. La variable booléenne `valide` indique si l'allocation est possible.
Les fonctions `A0(A, i, j)` et `A1(A, i, j)` donne l'adresse de l'élément (i, j) de la matrice A. La numérotation s'effectue **à partir de 0** pour `A0` et à partir de 1 pour `A1`. L'adresse d'un élément du vecteur b peut s'obtenir par `A0(b, 0, i)` ou `A1(b, 1, i)`.
La fonction `Dupliquer` retourne un pointeur sur une copie du vecteur ou de la matrice pointée par `donnee` et comportant le nombre d'`elements` indiqué. Cette fonction effectue l'allocation de mémoire nécessaire.
La procédure `Dispose` désalloue la place mémoire occupée par `donnee` et attribue à `donnee` la valeur `nil`.

2.8.2 Décomposition LU et résolution

La décomposition LU est effectuée sur place. L'ordre des lignes est noté dans un vecteur d'entiers `Lignes` de type `Indices = array[0..maxn] of integer`, ce qui évite de faire effectivement les permutations. Au début on pose `Lignes[i] := i` pour tout i, et on considère que l'élément de la i^o ligne et de la j^o colonne de la matrice se trouve en `A[Lignes[i], j]`.

Exemple : Pour une matrice d'ordre 5, si l'ordre des lignes est $[4, 1, 5, 2, 3]$, le triangle inférieur strict de $L = (\ell_{ij})$ est le triangle supérieur de $U = (U_{ij})$ sont stockés à la place de la matrice A comme indiqué ci-dessous :

$$\left\{\begin{array}{l} Ligne[1] = 4 \\ Ligne[2] = 1 \\ Ligne[3] = 5 \\ Ligne[4] = 2 \\ Ligne[5] = 3 \end{array}\right. \quad \rightarrow \quad \begin{pmatrix} \ell_{21} & U_{22} & U_{23} & U_{24} & U_{25} \\ \ell_{41} & \ell_{42} & \ell_{43} & U_{44} & U_{45} \\ \ell_{51} & \ell_{52} & \ell_{53} & \ell_{54} & U_{55} \\ U_{11} & U_{12} & U_{13} & U_{14} & U_{15} \\ \ell_{31} & \ell_{32} & U_{33} & U_{34} & U_{35} \end{pmatrix}$$

Si les calculs rencontrés sont corrects (`valide=true`) le triangle inférieur de la matrice L se trouve en $(A_{Lignes[i],j})_{j<i}$, et le triangle supérieur de U en $(A_{Lignes[i],j})_{j \geq i}$.
On dispose des procédures Pascal :

Procedure `DecompositionLU (A: PR2; n: integer; coeff, precision: integer;`
`var Lignes: Indices; pivotPartiel: boolean; var valide: boolean);`
Procedure `Resolution (A, b: PR2; n, coeff, precision: integer;`
`var Lignes: Indices; var valide: boolean);`

La procédure `Résolution` résoud les deux systèmes à matrice triangulaire correspondant à L et U. Le second membre `b` est remplacé par la solution. La matrice `A` doit contenir la factorisation LU produite par la procédure `DécompositionLU`, suivant l'ordre indiqué par `Lignes`.

2.8.3 Quelques résultats

Les procédures ci-dessous ont été appliquées à la matrice de Hilbert :

$$A_n = (\frac{1}{i + j - 1})_{1 \leq i,j \leq n}$$

bien connue pour son mauvais conditionnement pour la résolution des systèmes linéaires.

• dans le type rationnel-flottant `coefq`, en précision simple, la décomposition LU exacte est obtenue pour $n \leq 8$, et pour $n \leq 17$ en précision étendue.

• dans le type intervalle `coefi`, en précision étendue, l'existence de la décomposition LU est obtenue pour $n \leq 9$ (pivots garantis non nuls). Dans le type rationnel-intervalle `coefqi` la décompsition exacte LU est obtenue pour $n \leq 16$, et l'existence démontrée pour $n \leq 21$.

• dans le type stochastique [6] `coefs3`, en précision double, les pivots possèdent au moins 1 chiffre décimal correct pour $n \leq 12$.

3 Application 2 : calculette.

Il s'agit d'un simple programme, fonctionnant en "mode texte", et permettant d'expérimenter les bibliothèques présentées ci-dessus. L'annexe 3 donne un résumé de la syntaxe et du mode d'emploi de ce programme.

Ce programme permet de lire un fichier texte écrit à l'aide d'un éditeur de textes extérieur, contenant des définitions de constantes et de fonctions écrites dans un langage de style Pascal très simplifié. Ce langage permet de manipuler des variables et d'écrire des expressions dans tous les types définis dans la section **2**, en précision étendue.

On peut ensuite calculer des expressions contenant ces constantes et fonctions. Les résultats s'affichent immédiatement sur l'écran et peuvent également enregistrés dans un fichier texte.

3.1 Exemple 1 : calcul stochastique

La fonction fs ci-dessous calcule e^t à l'aide de la série de Taylor $\sum_0^\infty x^k/k!$. L'utilisation de cette série donne de mauvais résultats pour $t < 0$. Pour t=-10 par exemple le résultat de l'ordre de $4,5\ 10^{-5}$ est obtenu à partir d'une somme de termes dont le plus grand en valeur absolue est $2,7557\ 10^3$. On peut donc s'attendre à une erreur absolue d'au moins 10^{-16}, soit une erreur relative de l'ordre de 10^{-11}.

Ce phénomène est mis en évidence en utilisant le type stochastique `s3`, ce qui permet d'obtenir une estimation du nombre de chiffres corrects du résultat. La sommation de la série s'arrête lorsque la somme partielle ne varie plus de manière significative, compte tenu de la précision.

La variable locale v sert à calculer $t^k/k!$. La somme partielle passe de u à $u1 := u + v$ et on s'arrête lorsque $d := u1 - u$ a un nombre de chiffres exacts nul, c'est-à-dire devient un zéro informatique [23]. Le nombre de chiffres significatifs exacts de d est donné par $d.c$.

```
fs(t): s3;   {fonction à valeurs dans s3}
var
   u, u1, d, v, x : s3;   {variables locales de type s3}
   k;   {variable locale de type flottant, par défaut}
begin
   u := 1;{somme partielle initialisée à 1}
   v := 1;{terme général de la série }
   x := t;
   d := 1;
   k := 0;
```

```
    while d.c > 0 do    {tant que d n'est pas un 0 informatique}
      begin
        k := k+1;
        v := v*x/k;{x^k/k!}
        u1 := u+v;{nouvelle somme partielle}
        d := u1-u;
        u := u1;
      end;
    fs:= u ;  {affectation du résultat final à fs}
  end;
```

On notera que dans une affectation telle que :

```
v := v*x/k;
```

`v` étant une variable de type stochastique `s3` le second membre est considéré comme une expression de type stochastique, ce qui veut dire chaque calcul est fait successivement sur les 3 composantes, la direction de l'arrondi étant choisie au hasard, pour chaque composante et chaque opération.

3.2 Calculs

Après lecture du fichier contenant `fs` par la calculette, le calcul de fs se fait en tapant simplement une expression telle que $fs(t)$, où t est une valeur numérique ou une expression flottante.
Résultats :

```
fs(1) = {19 à 95%}2,718 281 828 459 045 235...+\5,39E-19
fs(-5) = {16 à 95%}0,006 737 946 999 085 468...+\5,89E-19
fs(-10) = {11 à 95%}4,539 992 976 3...E-5+\9,73E-16
```

Sur chaque ligne on trouve une estimation à la probabilité $p = 0,95$ du nombre de chiffres corrects, suivi du résultat moyen et de la demi-longueur de l'intervalle contenant le résultat exact.

3.3 Exemple 2 : dichotomie et intervalles

Partant d'un intervalle $[a, b]$ sur lequel une fonction continue f donnée change de signe, la méthode de dichotomie fournit un encadrement d'un zéro de f en remplaçant successivement $[a, b]$ par un intervalle de longueur moitié choisi suivant le signe de $f((a + b)/2)$. La fiabilité de ce signe décroit à mesure qu'on se rapproche d'un zéro. Un calcul d'intervalle permet de dire si le signe est positif, négatif ou incertain.
Considérons par exemple la fonction : $f(x) := 19x^4 - 18x^3 + 4x - 94/100$, qui s'annule sur $[0, 1]$.
Le calcul de $f(x)$ est fait par la fonction :

```
f(x) : i;   {fonction donnant un résultat de type intervalle}
var
  u : i;   {variable locale de type intervalle}
begin
  u := (x, x);   {intervalle réduit à x}
  f:= 19u^4 - 18u^3 + 4u - 94/100;   {calcul d'intervalles}
end;
```

Le signe est obtenu par :

```
signe(x);
   var
      v : i;   {variable locale de type intervalle}
begin
   v := f(x);   {calcul d'intervalles}
   if v.a>0 then
      signe := 1   {f(x) est >0}
   else if v.b<0 then
      signe := -1   {f(x) est <0}
   else
      signe := 0;   {signe incertain}
end;
```

La division par 2 d'un intervalle $[x_1, x_2]$ sur lequel f change de signe est fait à l'aide de la fonction fm ci-dessous, a valeurs dans R^3. Les deux premières composantes contiennent le nouvel intervalle, et la troisième indique ce qui a été fait : si elle vaut 1 la réduction de l'intervalle a échoué.

```
fm(x1, s1, x2) : r3;   {s1 est le signe de x1}
var
   m, sm, n, sn;   {variables locales flottantes}
begin
   m := (x1+x2)/2;
   if (m=x1) or (m=x2) then   {x1 et x2 sont deux nombres consécutifs}
      fm := (x1, x2, 1)
   else
      begin
         sm := signe(m);
         if sm=0 then   {signe du milieu incertain}
            begin
               echec := true;
               n := (x1+m)/2;   {essayer [n, x2]}
               if (x1<n)and(n<x) then
                  begin   {x1 et m ne sont pas consécutifs}
                     sn := signe(n);
                     if s1=sn then   {on considère [n, x2]}
                        begin
                           fm := (n, x2, 0);
                           echec := false;
                        end;
                     if echec then   {essayer [x1, (m+x2)/2]}
                        ...
                  end;
               if echec then
                  fm := (x1,x2, 1)
            end
         else if sm=s1 then
```

```
            fm := (m, x2 , 0)    {x1 est remplacé par m}
         else
            fm := (x1, m, 0) ;   {x2 est remplacé par m}
      end;
   end;
```

Remarque : Lorsque le signe du point milieu m est indéterminé on essaie de poursuivre avec $[n, x_2]$, $n = (x_1 + m)/2$, si le signe de $f(n)$ est le même que celui de $f(x_1)$ ou avec $[x_1, n']$, $n' = (x_2 + m)/2$ si le signe de $f(n')$ est le même que celui de $f(x_2)$. On évite ainsi un arrêt prématuré lorsque l'intervalle de départ est symétrique par rapport à un zéro de f.

L'encadrement d'un zéro a partir d'un intervalle $[x_1, x_2]$ est fait par la fonction :

```
racine(x1, x2) : i;
var
   y : r3;    {variable locale de type r3}
   s1, s2;    {variables locales de type flottant}
begin
   s1 := signe(x1);
   s2 := signe(x2);
   if s1*s2 = (-1) then    {intervalle de départ correct}
      begin
         y := (x1, x2, 0);    {intervalle initial}
         repeat
            y := fm(y.1, s1, y.2);    {itération}
         until y.3=1;
         racine := (y.1, y.2);
      end
   else
      write('intervalle incorrect');    {message d'erreur}
end;
```

Résultat : Le calcul de `racine(0 , 1)` donne :

```
res{i}= 0,533 737 750 139 717 644 5..+\ 3,91E-18
```

3.4 Conservation des résultats en mémoire

Il est également possible de conserver en mémoire plusieurs résultats et de les réutiliser dans les expressions. Il faut pour celà taper :

```
identificateur := expression
```

Par exemple, après lecture du fichier de l'exemple 1 ci-dessus, on peut taper :

```
r1 := fs(1/1000)
```

donne un résultat de type `s3` sous la forme :

```
r1{s3} = {18 a 95%}1,000 100 005 000 166 67..+\2,70 E-19
```

puis :

```
r2 := fs(-1/1000)
```

donne :

```
r2{s3} = {18 a 95%}0,999 900 004 999 833 337..+\9,52 E-20
```

Enfin :

```
(r1-r2)/2
```
donne :
```
res{s3} = {15 a 95%}1,000 000 001 666 67..E-4+\3,89 E-20
```
Ce calcul illustre la perte de précision relative qui se produit dans la soustraction de nombres voisins, et justifie la présence dans la bibliothèque `vf_IEEE` de la fonction `exp1(x)=exp(x)-1` qui permet, par exemple, un calcul numériquement stable de la fonction $\mathrm{sh}(x) = (e^x - e^{-x})/2$.

4 Application 3 : Calculateur graphique interactif.

Il s'agit d'un logiciel [17] fonctionnant uniquement sur les micro-ordinateurs de type Macintosh. Ce logiciel disposant d'une interface graphique reprend et étend les possibilités de programmation de la calculette présentée ci-dessus en y ajoutant :

- un éditeur de texte destiné à la définition de fonctions et de procédures, mettant en évidence la structure des programmes et facilitant la correction des erreurs de syntaxe,
- une fenêtre graphique permettant de construire des figures très élaborées et modifiables interactivement,
- une fenêtre permettant de manipuler des matrices dont les éléments peuvent être de l'un des types décrits ci-dessus, et de mettre en œuvre les méthodes classiques de résolution de systèmes linéaires, de calcul de valeurs propres, ainsi que l'algorithme du simplexe. L'exécution pas à pas de ces algorithmes, sous le contrôle de l'utilisateur fait de ce logiciel un outil pédagogique intéressant [16].

L'annexe 4 présente les principales caractéristiques et fonctionnalités de ce logiciel.

5 Conclusion

Les bibliothèques et logiciels présentés ci-dessus permettent l'expérimentation du standard IEEE de calcul en virgule flottante à trois niveaux différents :

- en Pascal, à l'aide des bibliothèques vf_IEEE et vf_CQIS,
- à l'aide d'un simple logiciel en mode texte permettant de programmer des fonctions et de faire des calculs en utilisant ces fonctions,
- à l'aide d'un logiciel graphique interactif possédant par ailleurs de nombreuses applications dans l'enseignement des mathématiques et de l'analyse numérique [8, 12, 14, 15, 16] .

Pour obtenir une copie des bibliothèques vf_IEEE, vf_CQIS et de la calculette il suffit d'en faire la demande par courier électronique à l'auteur (Raymond.Pavec@univ-brest.fr).

Annexe 1 : Bibliothèque vf_IEEE

Cette annexe énumère les principales déclarations de l'interface de la bibliothèque vf_IEEE [3].

```
Const
   invalid = 1; denormal = 2; divzero = 4;
   overflow = 8; underflow = 16; inexact = 32;

Type
   RelOp = (GreaterThan, LessThan, EqualTo, Unordered);
   NumClass = (SNaN, QNaN, Infinite, ZeroNum, NormalNum, DenormalNum);
   RoundDir = (ToNearest, Upward, DownWard, TowardZero);
   RoundPre = (ExtPrecision, DblPrecision, RealPrecision);
   Environment = word;
Var
   inf, maxRand: extended;   {inf désigne l'infini, maxRand vaut 2^31-1}
```

Procedure `init_vf;`
 `{Initialisation du calcul flottant et installation l'environnement standard IEEE :`
 `exceptions masquées, arrondi au plus près, précision étendue.`
 `Initialisation de inf et maxRand}`

Function `ClassExtended (x: extended): NumClass;   {Retourne la classe de x}`

Function `NextReal (x: single; direction: extended): single;`
Function `NextDouble (x: double direction: extended): double;`
Function `NextExtended (x: Extended; direction: extended): extended;`
 `{Ces fonctions donnent le nombre flottant suivant ou précédant x, suivant la`
 `position de direction par rapport à x}`

`(** Acces a l'environnement **)`
Procedure `SetHalt(e: word);`
 `{Fixe les conditions d'arrêt des programmes. e est une somme de`
 `constantes choisies parmi invalid,..., inexact}`
Procedure `SetRound (r: RoundDir);`
 `{Installe la direction d'arrondi indiquée}`
Function `GetRound: RoundDir;`
 `{Donne la direction d'arrondi en cours}`
Procedure `SetPrecision (p: RoundPre);`
 `{Installe la précision d'arrondi choisie}`
Function `GetPrecision: RoundPre;`
 `{Donne la précision d'arrondi en cours}`
Procedure `ProcEntry (Var e: environment);`
 `{Note l'environnement de calcul en e et installe l'environnement standard IEEE}`
Procedure `ProcExit (e: environment);`
 `{Restaure l'environnement e : exceptions, arrondi, précision}`

`(** Fonctions élémentaires **)`

[3] D'après un projet de maîtrise d'Informatique de Pascal Tang

Function `Relation (x: Extended; y: Extended): RelOp;`
`{Retourne r tel que x r y soit vrai}`
Function `logb (x: extended): extended;`
`{Donne l'exposant de la représentation normalisée de x en base 2}`
Function `scalb (n: integer; x: extended): extended;`
`{Donne le produit de x par 2^n}`
Function `Rint (x: extended): Extended;`
`{Arrondi à l'entier le plus proche, dans la direction de l'arrondi}`
Function `ln1 (x: extended): extended;`
`{Evaluation numériquemment stable de ln(1+x)}`
Function `exp1 (x: extended): extended;`
`{Evaluation numériquemment stable de exp(x)-1}`
Function `XRemY (x: extended; y: extended): extended;`
`{Donne le reste de la division de x par y}`
Function `XpwrI (x: extended; i: integer): extended; {Donne x^i}`
Function `XpwrY (x: extended; y: Extended): extended; {Donne x^y}`
Function `RandomX (Var x: extended): extended;`
`{x étant un entier compris entre 1 et maxRand-1, cette fonction donne le reste de la division de 7^5x par maxRand. x prend également cette valeur}`

Annexe 2 : Bibliothèque vf_CQIS

Const

```
precisionCalcul = 0;   {précision étendue}
precisionSimple = 1; precisionDouble = 2;
  {configuration : nature des entrées et sorties attendues}
pxReduit = 1; pyReduit = 2; pzReduit = 4;
  {Opérations}
xR = 0; xPlusy = 1; xMoinsy = 2; xMuly = 3; xDivy = 4;
xMulB = 5; xMulRC = 6; xMulXY = 7;   {type complexe}
xpwn = 8;   {puissance entière}
xly = 9; xley = 10; xgy = 11; xgey = 12;
xeqy = 13; xneqy = 14; xandy = 15; xory = 16;   {comparaisons}
  {Types de données}
coefR = 1; coefC = 2; coefQ = 3; coefI = 4; coefQI = 5;
coefS2 = 6; coefS3 = 7; coefS4 = 8; coefS5 = 9;
```

Type

```
PExt = ^extended;
R2 = array[1..2] of extended;
PR2 = ^R2;
R2single = array[1..2] of single;
R2double = array[1..2] of double;
PR2single = ^R2single;
PR2double = ^R2double;
  {Des déclarations analogues basées sur R3, R4 et R5 existent également}
```

```
Procedure Operation (px, py,   {pointeurs sur les opérandes}
        pz: PR2;   {pointeur sur le résultat}
        op,   {opération à effectuer : xplusy, xmuly,...}
        coeff,   {type des données : coefR, coefC,...}
        precision: integer;   {précision de certaines données}
        var valide: boolean;   {validité de l'opération}
        configuration: integer);   {somme de constantes indiquant les données dont }
            {la précision est réduite : pxReduit, pyReduit, pzReduit}
Procedure RacineCarree (px: PR2; coeff: integer; var valide: boolean);
  {Calcul sur place de la racine carrée : dans le cas coefC la racine calculée est
   à partie réelle positive ou nulle}
Function Module (px: PR2; coeff, precision: integer): extended;
  {Dans le cas d'un intervalle le minimum du module est considéré}

Procedure Extension (source, destination: PR2;
     Coeff: integer; precision: integer);
  {Conversion de source, de la précision indiquée à la précision étendue}
Procedure Reduction (source, destination: PR2;
     var valide: boolean; coeff: integer; precision: integer);
  {Conversion de source, de la précision étendue à la précision indiquée,
     valide indique la validité}
Procedure Conversion (source, destination: PR2; typeS, typeD: integer);
  {source et destination en précision étendue. Conversion du type typeS en typeD}

Procedure Simplifier (py: PR2);
  {Simplication de la fraction pointée par py}
Function BonEntier (pe: PExt; precision: integer): boolean;
  {Indique si le flottant pointé par pe est entier et se trouve dans les bornes
   pour le type coefq, suivant la précision}
Function Q_Flottant (px: PR2; precision: integer): boolean;
  {Indique si la donnée px de type coefQ représente un nombre flottant}
Function QI_Intervalle (px: PR2; precision: integer): boolean;
  {Indique si la donnée px de type coefQI représente un intervalle}
Procedure QI_Vers_QouI (px: PR2; var coeff: integer);
  {Transforme px de type coefQI en rationnel ou en intervalle, coeff indique
   le type obtenu}
Function ValeurApprochee (px: PR2;
     coeff: integer; precision: integer): extended;
  {Calcul d'une valeur approchée de px. Dans le cas d'un intervalle le milieu
   est considéré}
Procedure MDC (px: PR5; coeff: integer; var resultats: R3);
  {Calcul de la moyenne, de la demi-longueur de l'intervalle et du nombre
   décimaux exacts de px, donnée de type stochastique en précision étendue}
Function ConversionEnChaine (px: PR2; coeff, precision, chiffres: integer;
     option: boolean): string;
  {La donnée pointée par px est convertie en chaine de caractères. Si
     option est true les chiffres décimaux sont groupés par 3}
```

Annexe 3 : Calculette

Cette calculette utilise uniquement la précision étendue. Elle permet d'utiliser les types c, q, i, qi, s2. . . s5, correspondants aux types décrits dans la section 2. Sauf indication explicite de type, des nombres flottants sont considérés.
La calculette permet d'ouvrir un fichier de fonctions, puis de faire des calculs sur ces fonctions. La lecture du fichier se poursuit tant qu'aucune erreur de syntaxe n'est décelée. En cas d'erreur un message indique le numéro de la ligne en cause, et la nature de l'erreur. Les fonctions correctement analysées sont énumérées et immédiatement utilisables. Les résultats des calculs sont directement affichés sur l'écran et peuvent également être notés dans un fichier de type "texte".

1 Syntaxe du fichier de fonctions

Dans les descriptions de syntaxe ci-dessous les éléments entre crochets [...] sont facultatifs

Le fichier peut contenir une liste de constantes et fonctions, séparées par des points- virgules. La syntaxe des constantes et fonctions est :

```
Identificateur[(Arguments)] := Expression flottante;
```

ou

```
Identificateur[(Arguments)] [: Type] ;
[var
   Variables locales;]
Bloc;
```

Dans la deuxième forme le `Bloc` doit contenir une affectation de résultat à l'identidificateur de la fonction, sinon la valeur retournée est invalide (NaN).

1.1 Identificateur

Chaîne d'au plus 10 caractères alphabétiques ou numériques, le premier étant alphabétique. Minuscules et majuscules sont confondues.

1.2 Arguments

Liste d'identificateurs séparés par une virgule. Les arguments sont des nombres flottants. En l'absence d'argument on définit une constante. Les arguments sont utilisables uniquement en lecture : on ne peut pas les modifier.

1.3 Type

Donne le type de valeurs retourné par la fonction. Les mêmes types sont utilisés pour les variables locales. Les types possibles sont :

- r (facultatif) : nombre en virgule flottante, seul admis pour une constante.
- rn : vecteur de n composantes flottantes, $2 \leq n \leq 9$ pour les fonctions, $2 \leq n \leq 218$ pour les variables locales,

- c : complexe,
- q : rationnel-flottant,
- i : intervalle,
- qi : rationnel-intervalle,
- s2…s5 : stochastique.

1.4 Variables locales

Listes d'identificateurs avec indication facultative de type. On accède aux composantes sous la forme : `Identificateur.suffixe`. Les suffixes à utiliser sont :
- types rn : suffixes 1…218, ou indice (variable locale flottante),
- type c : r partie réelle, i partie imaginaire,
- type q : n numérateur, d dénominateur,
- type i : a borne inférieure, b borne supérieure,
- types s2…s5 : composantes 1…5. Les suffixes m, d, c donnent respectivement : la moyenne des composantes, la demi-longueur de l'intervalle contenant le résulat exact, le nombre de chiffres décimaux corrects.

On ne peut pas accéder directement aux composantes d'une variable de type `qi`.
Les mêmes suffixes s'appliquent à l'identificateur de la constante ou de la fonction en cours de définition.

1.5 Bloc

Un bloc comporte :
- soit une seule instruction
- soit une ou plusieurs instructions précédées de `begin` et se terminant par `end`.

1.6 Instructions

1.6.1 Affectation

La syntaxe d'une affectation est :

```
identificateur := expression typée;
```

L'identificateur est une variable locale, la fonction ou une composante. L'expression est du type du premier membre.

1.6.2 Instructions de contrôle

La syntaxe des instructions de contrôle est voisine de celle du Pascal. Les instructions de contrôle sont :
- **if** … **then** … **else**

La syntaxe est :

```
if exprf then
   Bloc 1
[else
   Bloc 2]
```

L'expression flottante `exprf` est évaluée. Si elle donne un résultat

-invalide, le calcul est abandonné,
-différent de zéro (vrai), alors le Bloc 1 est exécuté,
-sinon, si la partie `else` est présente, le Bloc 2 est exécuté.

Exemple : Pour définir une fonction égale à 0 our $t < 0$ et à t pour $t \geq 0$ on peut écrire :

```
f(t);
   begin
      if t<0 then
         f := 0
      else
         f := t;
   end;
```

- **Boucle for**

La syntaxe est :

```
for id = exprf1 to exprf2 [step exp3] do
   Bloc
```

`id` est un identificateur non déclaré comme variable locale, dont la validité s'étend au Bloc suivant l'instruction, et qui ne peut pas être modifié dans ce bloc. Les expression flottantes exprf1, exprf2 et exprf3 (si elle est présente, sinon la valeur 1 est considérée) sont évaluées. Si les résultats sont valides et exprf3≠0 le bloc est exécuté pour :

$$\text{id} = \text{exprf1} + k \times \text{exprf3}, \quad k = 0, 1, \ldots, km$$

km désignant l'entier arrondi au plus près de (exprf2-exprf1)/exprf3. Rien n'est fait si $km < 0$. L'utilisation d'un pas non entier ne posera en général pas de problème. Par exemple pour l'instruction :

```
for k := 0 to 1 step 0,1 do
   Bloc
```

on aura `km=10`.

Exemple : Pour calculer $s = \sum_{k=0}^{n} x^k/k!$ on peut écrire :

```
s := 1; y := 1; m:=1;
for k := 1 to n do
   y := y*x;
   m := m*k;
   s := s + y/m;
end;
```

- **while**

La syntaxe est :

```
while exprf do
   Bloc
```

Le Bloc est exécuté tant que `exprf` donne un résulat différent de 0. Si un résulat invalide est obtenu le calcul est abandonné.

Exemple : Pour compter le nombre de lancer de dés nécessaires pour obtenir pile on peut écrire :

```
k:=1;
while ?<0,5 do
   k := k+1;
```

Le point d'interrogation `?` donne un nombre au hasard compris entre 0 et 1.

- **repeat**

La syntaxe est :

```
repeat
   Bloc
until exprf
```

Le Bloc est exécuté, puis `exprf` est évaluée. Si le résulat est nul le Bloc est à nouveau exécuté, jusqu'à ce `exprf` donne une valeur différente de 0. Si un résultat invalide est obtenu le calcul est abandonné.

Exemple : Pour compter le nombre de lancer de dés nécessaires pour obtenir pile on peut aussi écrire :

```
k:=0;
repeat
   k := k+1;
until ?<0,5;
```

- **case**

La syntaxe est :

```
case exprf of
   liste1 :
      Bloc 1;
[   liste2 :
      Bloc 2;...]
[otherwise
   Bloc 0]
end
```

`exprf` est évaluée, et si elle donne un résultat entier figurant dans une liste le Bloc correspondant est exécuté, sinon le Bloc 0 est exécuté, s'il est présent. Si le résulat est invalide le calcul est abandonné. Une liste est une suite d'entiers séparés par une virgule, ou par deux points. Le nombre de listes est quelconque, mais la différence entre le plus petit et le plus grand entier figurant dans une liste ne doit pas dépasser 1000.

Exemple : `mu` est un paramètre prédéfini. L'instruction ci-dessous permet de choisir une fonction parmi f, g, h,... suivant les valeurs de `mu`.

```
case mu of
   0 : u := f(t);
   1 : u := g(t);
   2 : u := h(t);
otherwise
   begin
      u:=nan(1); write('Ce cas n'est pas prévu !');
   end;
end;
```

1.6.3 Procédures prédéfinies

• **setround**
`setround(k)` modifie la direction de l'arrondi :
 ◦ k=1 : arrondi au plus près (valeur par défaut),
 ◦ k=2 : arrondi par excès,
 ◦ k=3 : arrondi par défaut,
 ◦ k=4 : arrondi vers 0.

• **setprec**
`setprec(k)` modifie la précision de l'arrondi :
 ◦ k=1 : précision simple,
 ◦ k=2 : précision double,
 ◦ k$\neq$ 1 et $\neq$ 2 : précision étendue (valeur par défaut).

Au début d'un calcul les valeurs par défaut sont utilisées.

• **leave**
Cette procédure sans argument n'est utilisable qu'à l'intérieur d'un bloc associé à une boucle for, while ou repeat, et fait sortir de cette boucle.

• **cycle**
Cette procédure sans argument n'est utilisable qu'à l'intérieur d'un bloc associé à une boucle for, while ou repeat, et supprime l'exécution des instructions suivantes du bloc.

• **exit**
Cette procédure sans argument interrompt le calcul en cours

• **write**
`write` permet d'afficher des commentaires ou des résultats intermédiaires en cours de calcul. Cette procédure admet un nombre quelconque d'arguments, de deux types :
 ◦ expression flottante,
 ◦ chaîne de caractères.

Un argument suivi de `: k`, `k` étant un entier occupe au moins `k` caractères. Une chaîne de caractères terminée par un point virgule provoque un retour à la ligne.

Exemple : Le calcul de la fonction `f` ci-dessous en `t=1` :

```
f(t);
begin
   write('Calcul de f;', 't=', t);
   f := ...
end;
```

provoquera l'affichage de :

```
Calcul de f
t=1
```

1.7 Expressions

La syntaxe des expressions est analogue à celle du Pascal. Elles peuvent contenir :

• les 4 opérations : $+ - */$, exponentiation ˆ (avec exposant entier uniquement, sauf pour les expressions flottantes)

• les signes de comparaison : $<, >, =, <=, >=, <>$. les valeurs données par les comparaisons sont 0 ou 1, pour faux et vrai respectivement. Toute valeur différente de 0 représente vrai.

• les arguments de la fonction, les variables locales, un paramètre prédéfini `mu`, égal à 0 par défaut,

• les fonctions usuelles avec quelques restrictions suivant les types. Ainsi pour les types `i` et `s2`...`s5` les seules fonctions disponibles sont `abs, sqr, sqrt`.

Les dérivées première et seconde des fonctions usuelles, ou des fonctions définies précédemment peuvent être utilisées. Pour ces dernières, si la dérivée n'est pas explicitement définie, un calcul approché est effectué. Les formules appliquées sont respectivement :

$$\frac{f(t+h)-f(t-h)}{2h} \quad \text{et} \quad \frac{f(t+h)+f(t-h)-2f(t)}{h^2}$$

avec $h = 2^{e-10}$, e étant l'exposant de la représentation flottante normalisée de t (sauf pour t voisin de 0).

On dispose de fonctions de calcul approché d'intégrales et de résolution d'équations de la forme `f(t)=g(t)` ou `f(t)=c`. La syntaxe de ces fonctions est :

```
gauss(f , k , a , b , np)   {Méthode de Gauss à k points, 1<=k<=50}
trap(f , a , b , np)   {Méthode des trapèzes}
simpson(f , a , b , np)   {Méthode de Simpson}
newtoncotes(f , k , a , b , np)   {Méthode de Newton-Cotes à k points}
eulerml(f , a , b , np)   {Méthode d'Euler-Mac-Laurin}
romberg(f , a , b , np0 , epsilon , test , max)   {Méthode de Romberg}
```

`f` est la fonction à considérer, `k` indique le nombre de points de la formule de base à utiliser, `[a,b]` est l'intervalle de calcul, et `np` le nombre total de points à considérer : la formule de base est répétée `np/k` fois. Dans la méthode de `romberg`, `np0` est le nombre de points à considérer pour la première application de la mathode des trapèzes, `epsilon` une valeur positive, sevant pour le test d'arrêt. Si `test` est différent de 0 le test est fait en valeur relative, `max` est le nombre d'itérations à ne pas dépasser. Si `max` est positif et que le test d'arrêt n'est pas satisfait après `max` itérations le résulat est un NaN. Si `max<0` le dernier itéré calculé est retourné, que le test soit satisfait ou non. Pour `max=-1` par exemple une itération est effectuée.

Ces paramètres ont la même signification dans les fonctions suivantes :

```
dichotomie(f, g , a , b , epsilon , test , max)
regulafalsi(f, g , a , b , epsilon , test , max)
newton1(f, g , x0 , epsilon , test , max)
newton(f, g , x0 , epsilon , test , max)
```

Ces fonctions tentent de résoudre l'équation `f(t)=g(t)` ou `f(t)=0` sur `[a,b]`. `newton1` utilise un calcul approché de la dérivée.

• des sommes et produits : La syntaxe est :

```
sigma(k:=debut, pas, fin, expression)   {Somme}
sigma1(k:=debut, pas, fin, expression)   {Moyenne}
sigma2(k:=debut, pas, fin, expression)   {Variance}
produit(k:=debut, pas, fin, expression)   {Produit}
```

`k` est un identificateur non encore utilisé (et non déclaré). L'expression est calculée pour les valeurs de `k` calculées comme dans une boucle `for` et la somme, la moyenne, la variance ou le produit est calculé. L'expression ne doit pas contenir d'autres sommes ou produits. `sigma1` et `sigma2` ne peuvent pas être utilisées avec des expressions typées.

1.8 Commentaires

Des commentaires peuvent être placés entre accolades $\{\ldots\}$.

2 Calcul d'expressions

Il est possible de calculer une expression de l'un des types :

`r, c, q, i, qi, s2...s5`

2.1 Changement de type

Le choix du type se fait en tapant ! suivi du type choisi (Le type peut aussi être déterminé par la fonction calculée). Après avoir tapé `!q` le programme affiche :
`Exp{q} ->`
et attend la frappe d'une expression rationnelle.

2.2 Calcul d'une expression

Pour calculer une expression, du type affiché par le programme, il suffit de taper cette expression et d'appuyer sur la touche de Retour à la ligne. Le résultat du dernier calcul effectué est conservé dans la variable `res`. Par exemple si après avoir choisi le type rationnel on tape :
`Exp{q} ->` $1/2 - 1/3$
le résultat est affiché sous la forme :
`res{q} = 1 / 6`

2.3 Conservation des résultats en mémoire

Pour conserver un résultat en mémoire, de manière à pouvoir le réutiliser il suffit de faire précéder l'expression d'un identificateur suivi de `:=`. Si l'identificateur est `mu` la valeur calculée est affectée au paramètre prédéfini `mu`.

Annexe 4 : Calculateur graphique interactif

Le développement de ce logiciel est un processus qui s'étend sur une dizaine d'années, à partir de 1987 [15]. Le premier but poursuivi était la création d'un logiciel d'analyse numérique pour l'enseignement, permettant une mise en œuvre rapide des méthodes classiques, et doté d'une interface facilitant l'entrée des données et l'examen des résulats.
Les possibilités graphiques du logiciel ont intéressé le groupe de Formation Recherche constitué au Lycée de Kerichen de Brest en 1990 sur le thème de l'illustration à l'aide de l'ordinateur de cours et d'exercices de mathématiques [9, 12]. La collaboration avec ce groupe a été à l'origine de nouveaux développements, principalement la possibilité de contruire des figures modifiables interactivement, à la manière de Cabri-Géomètre en géométrie élémentaire [3].
L'utilisation du logiciel en première année de DEUG à l'INSA de Toulouse, comme outil d'initiation à la résolution de problèmes sur ordinateur, a également fait l'objet d'une publication [8].

La taille de la dernière version de ce logiciel est de l'ordre de 60 000 lignes de texte source de Pascal. Le code compilé occupe 800K.
Ce logiciel fonctionnant uniquement sur Macintosh existe en deux versions correspondant à l'évolution matérielle de ce type de micro-ordinateur :

• version 68000, pour les micro-ordinateurs équipés d'un microprocesseur de la série Motorola 68000, et construits jusqu'en 1995,

• version PowerPc, pour les micro-ordinateurs équipés d'un micoprocesseur PowerPc, et construits depuis 1993. Sur ce type de machine il n'y a pas de précision étendue : les calculs sont effectués en double précision. Le numérateur et le dénominateur des rationnels sont limités à 53 chiffres en base 2, soit $m = 2^{53} - 1$.

La description de ce logiciel se subdivise en quatre parties correspondant chacune à une fenêtre spécifique du logiciel : programmation, calcul numérique, représentations graphiques, algorithmes élémentaires d'algèbre linéaire.

1 Programmation

Le langage de programmation reprend toutes les possibilités de la calculette, en y ajoutant quelques possibilités supplémentaires.

1.1 Editeur de texte intelligent

L'éditeur de texte est intégré au logiciel. Cet éditeur est intelligent en ce sens qu'il connaît la syntaxe du langage et fait apparaître clairement la structure des blocs : décalages automatiques, variations de style et de couleur. La correction des erreurs se fait pas à pas en réponse aux messages d'erreur.

1.2 Procédures de dessin

Les représentations graphiques de fonctions cartésiennes ou paramétrique, de surfaces $z = f(x, y)$ sont réalisées automatiquement à l'aide des outils du logiciel. Il est également possible d'écrire des commentaires dans les graphes, directement à partir du clavier.

On dispose en outre de procédures de dessin dans le plan ou dans l'espace. Les procédures prédéfinies permettent de tracer des points, des segments de droite, des vecteurs, des figures géométriques simples : cercles, rectangles, ellipses... écrire du texte, des résultats numériques.

Exemple La représentation graphique en "toile d'araignée", de la suite récurrente :

$$u_{n+1} = f(u_n), \quad f(t) = c - t^2/4$$

peut être tracée à partir de la programmation suivante :

```
\c c := 1,5;   {\c indique une constante}
f(t) := c-t^2/4;
\c u0 := 5; \c max := 20; \c epsil := 0,01;   {constantes}

suite : r2;  {procedure de dessin dans le plan r2}
var
   u, v, w, n;  {variables locales de type flottant}
begin
   u := u0;
   moveto(u, 0);  {place la plume en (u,0)}
   mark(6);  {marque numero 6}
   n := 1;
   v := f(u);
   moveto(u, 0);  {place la plume en (u,0)}
   lineto(u, v);  {trace un segment de (u,0) à (u,v)}
   while (abs(u-v)>epsil)&(n<=max) do   {boucle avec test d'arrêt}
      begin
         lineto(v, v);  {trace un segment jusqu'à (v,v)}
         u := v; v := f(v);
         lineto(u, v);  {trace un segment jusqu'à (u,v)}
         n:=n+1;
      end;
   moveto(-6, 3); write(' termes calculés : ', n);
end;
```

Le langage de programmation ne comporte pas de programme principal. L'exécution d'une procédure telle que `suite` se fait à l'aide de la souris, après sélection du graphe destiné à recevoir le dessin, et de la procédure.

1.3 Définition de fonctions à l'aide de méthodes numériques

Des fonctions peuvent être également définies comme :

• **fonction réciproque**, calculée par dichotomie ou *regula falsi*. Le calcul de la fonction f choisie est d'abord fait en des points équidistants (x_i) précisés. Ensuite le calcul approché de $f^{-1}(t)$ pour t donné est fait en appliquant la méthode numérique choisie sur l'intervalle $[x_i, x_{i+1}]$ tel que t apartient à l'intervalle d'extrémités $f(x_i)$ et $f(x_{i+1})$.

• **primitive**, calculée par une méthode de Newton-Cotes. Le calcul de la primitive est fait en utilisant un nombre donné de points équidistants (x_i). Le calcul en x donné est ensuite fait en considérant i tel que $x \in [x_i, x_{i+1}]$ et en appliquant à nouveau la méthode choisie sur $[x_i, x]$.

• **solution de systèmes différentiels** avec conditions initiales, calculée par la méthode d'Euler, de Heun ou de Runge-Kutta d'ordre 4. Une estimation d'erreur et une extrapolation peuvent être effectuées. Comme pour la primitive le calcul est d'abord fait en des points équidistants (x_i). Le calcul en x donné est ensuite fait en considérant i tel que $x \in [x_i, x_{i+1}]$ et en appliquant un pas de la méthode choisie sur $[x_i, x]$.

• fonction d'**interpolation**. Les méthodes d'interpolation considérées sont l'interpolation de Lagrange, les splines cubiques de différents types, les fractions rationnelles.

• fonction d'**approximation** au sens des moindres carrés. Les fonctions de base considérées sont : des polynômes, des B-splines de degré ≤ 9 ou une famille quelconque de fonctions définies à l'aide du langage de programmation et dépendant d'un paramètre μ.

Toutes ces fonctions peuvent être utilisées de la même façon que les fonctions définies à l'aide du langage de programmation. En cas de modification de constantes intervenant dans leur définition, de conditions initiales, ou de points d'interpolation, ces fonctions sont remises à jour.

1.4 Définition de fonctions destinées à calculer des matrices

Ces fonctions, définies à l'aide du langage de programmation, possèdent deux arguments prédéfinis : i et j, indices de ligne et de colonne de la matrice, ou partie sélectionnée de matrice à laquelle on les applique. La sélection est parcourue de la première à la dernière ligne, et sur chaque ligne de la première à la dernière colonne.

Exemple 1 : Pour initialiser la matrice tridiagonale comportant des 2 sur la diagonale, des -1 de chaque coté on peut utiliser la fonction :

```
tridiag;  {fonction sans argument}
if i=j then   {sur la diagonale}
   trigiag := 2
else if abs(i-j)=1 then   {termes voisins de la diagonale}
   trigiag := -1
else
   tridiag := 0;
```

Exemple 2 : Les coefficients a_j du développement limité de $(1+x)^a$ satisfont à la relation de récurrence :

$$a_j = a_{j-1}(a - j + 1)/j, \text{ pour } j \geq 2, \; a_0 = 1, a_1 = a$$

Si on suppose a rationnel, le calcul se fait naturellement dans un tableau rationnel. Après avoir initialisé le premier élément d'une ligne à la valeur a, il suffit d'appliquer la fonction :

```
coeff : q;   {fonction s'appliquant à un tableau rationnel}
coeff := tab(i, j-1) (tab(i,1)-j+1)/j;
```

aux éléments suivants de la ligne considérée. `tab` désigne le tableau utilisé.

Une telle fonction peut aussi modifier directement n'importe quel terme de la matrice, désignée par `tab`, à laquelle est est appliquée et se comporter ainsi comme une procédure.

Exemple 3 : Supposons que l'on veuille faire la décomposition LU sur place d'une matrice $n \times n$, sans permutations de ligne, étape par étape. On va écrire une fonction `unpas` que l'on va appliquer successivement aux termes diagonaux de la matrice et qui va calculer successivement les colonnes de la matrice L, dans le triangle inférieur et faire l'élimination conduisant à la matrice U dans le triangle supérieur..

```
\c n := 10;  {ordre de la matrice}
unpas;
var
   x;
if (i=j)and(tab(i,i)<>0) then   {on vérifie que le pivot est bon}
   begin
      for i1 := i+1 to n do
         begin
            x:= tab(i1, i)/tab(i,i);   {multiplicateur pour la i° ligne...}
            tab(i1,i) := x;   {ce multiplicateur est un élément de L}
               for j1 := i+1 to n do   { Elimination}
                  tab(i1,j1) := tab(i1,j1)-x*tab(i,j1);
         end;
   end;
```

Cette fonction doit être appliquée successivement aux termes diagonaux d'une matrice. La matrice est modifiée par les instructions d'affectation à `tab`. Aucune valeur n'étant affectée à `unpas`, elle ne fait pas d'autre modification de `tab(i,j)`.
Si on applique cette fonction à toute la diagonale la décomposition LU sera obtenue en une seule fois.

2 Calcul numérique

On dispose dans une fenêtre d'édition de texte des possibilités de la calculette décrites dans l'annexe précédente. On dispose également :

- de la méthode de Newton pour les équations équations de la forme $f(x) = 0$,
- de la méthode des approximations successives pour les équations de la forme $f(x) = x$,

f désignant une fonction de $\mathcal{R}^n, (n \leq 9)$ dans lui-même.

3 Représentations graphiques

Il est possible de construire un document graphique d'au plus quatre pages, comportant un ou plusieurs graphes en 2 ou 3 dimensions. Dans chaque graphe, muni de son propre repère, orthonormé ou non, on peut tracer des courbes, exécuter des procédures, écrire du texte. Un graphe peut être agrandi, sur place ou dans autre graphe.

3.1 Figures modifiables interactivement

Les constantes définies à l'aide du langage de programmation peuvent être représentées par des points liés à une droite ou à une courbe : ces points peuvent être déplacés à l'aide de la souris, et les tracés qui en dépendent sont remis à jour immédiatement.
Pour la suite récurrente définie ci-dessus, le premier terme de la suite `u0` peut être représenté par un point sur l'axe des x, et `c`, le sommet de la parabole par un point sur l'axe des y. Si on déplace l'un de ces points à l'aide de la souris, la figure est remise à jour en continu.
Des figures animées peuvent également être obtenues en faisant défiler une suite de courbes.

3.2 Interpolation et approximation

Quelques méthodes d'interpolation et d'approximation peuvent être appliquées à des fonctions ou à des points donnés : interpolation de Lagrange, splines cubiques, fractions rationnelles, courbes de Bézier, B-splines. On définit ainsi des fonctions dans $\mathcal{R}$, $\mathcal{R}^2$ ou $\mathcal{R}^3$.

4 Algorithmes élémentaires d'algèbre linéaire

Le logiciel permet de définir des tableaux de nombres dans tous les types et précisions disponibles, pleins ou de forme bande, symétriques ou non. L'initialisation se fait interactivement à l'aide de valeurs numériques, d'expressions ou de fonctions des indices, pas copier-coller ou duplication. L'utilisation de fonctions des indices permet de faire toute sorte de calculs numériques. Les procédures de dessin permettent d'en obtenir une traduction graphique.
Exemple : Méthode de Relaxation.
Lorsqu'on discrétise (pas $h = 1/(n+1)$) le problème de Poisson sur le carré $[0,1] \times [0,1]$ de frontière Γ :

$$-\Delta u = f, u = 0 \text{ sur } \Gamma$$

on obtient un système d'équations :

$$-u_{i-1,j} - u_{i+1,j} + 4u_{ij} - u_{i,j-1} - u_{i,j+1} = h^2 f_{ij}, i,j = 1 \dots n$$

La méthode de relaxation de paramètre ω consiste à "balayer" le tableau en remplaçant u_{ij} par :

$$u_{ij} + \omega[h^2 f_{ij} + u_{i-1,j} + u_{i+1,j} - 4u_{ij} + u_{i,j-1} + u_{i,j+1}]/4$$

pour $i = 1 \dots n, j = 1 \dots n$, les valeurs de u à la frontière restant nulles.
Pour mettre en œuvre cette méthode on commence par calculer les valeurs $h^2 f_{ij}$ dans un tableau `tab1` carré de $n \times n$ éléments. L'initialisation se fera à l'aide des constantes et fonctions :

```
\c h := 1/(n+1); \c h2 := h^2;
f(x,y) := ...;   {fonction de x et de y}
fij := h2*f(i*h,j*h);   {fonction donnant le second membre}
```

On définit alors un tableau `tab2` de $(n+2) \times (n+2)$ éléments, indexé à partir de 0, initialisé à 0, et on effectue les itérations à l'aide de la fonction `relax` :

```
\c omega := 2/(1+sin(pi/(n+1)));   {paramètre de relaxation optimal}
d := (tab1(i,j)+tab2(i-1,j)+tab2(i+1,j)-4tab2(i,j)+tab2(i,j-1)+tab2(i,j+1))/4;
relax := tab2(i,j) + d*omega/4;
```

Il suffit pour cela de sélectionner le bloc correspondant à $1 \leq i,j \leq n$ et d'appliquer la fonction `relax` pour effectuer une itération.
Il est également possible d'effectuer une représentation graphique en 3 dimensions de la solution approchée correspondant à u, et de suivre son évolution.

4.1 Résolution de systèmes linéaires

La résolution de systèmes s'applique à tous les types : flottant, complexe, rationnel-flottant, intervalle, rationnel-intervalle, stochastiques, et à des systèmes carrés ou rectangulaires $m \times n$, avec $m \geq n$.

4.1.1 Décomposition LU, LL* ou QR

La démarche imposée par le logiciel pour la résolution des systèmes linéaires est celle effectivement utilisée en analyse numérique, et comprend deux étapes :
- décomposition LU, LL^* ou QR, avec ou sans permutation de lignes ou colonnes,
- résolution des deux systèmes ainsi obtenus.

On dispose de plusieurs stratégies de choix de pivot (ou de colonne) :
- décomposition LU : ordre naturel, pivot partiel avec colonnes dans l'ordre ou au hasard, pivot total,
- décomposition LL^* : ordre naturel ou pivotage diagonal,
- décomposition QR : ordre naturel des colonnes, ou dans l'ordre décroissant des normes euclidiennes.

Cette décomposition peut être faite pas à pas : l'utilisateur peut désigner le pivot (ou la colonne) à l'aide de la souris, ou laisser au logiciel le soin de le déterminer, suivant la stratégie sélectionnée.

4.1.2 Stockage des matrices

Si on désigne par $A = (a_{ij})$ la matrice considérée, on obtient dans tous les cas la décomposition d'une matrice PAQ, où P et Q sont des matrices de permutation. Ces permutations sont stockées sous forme de permutations σ_l, σ_c de l'ensemble des indices de lignes ou de colonne. La matrice décomposée est $(a_{\sigma_l(i),\sigma_c(j)})$. Dans le cas de la décomposition LU par exemple les parties utiles des matrices L et U sont stockées à la place de A : les éléments du triangle inférieur de L sont en $(a_{\sigma_l(i),\sigma_c(j)})_{j<i}$ et les éléments du triangle supérieur de U sont en $(a_{\sigma_l(i),\sigma_c(j)})_{j\geq i}$.
L'affichage de la matrice peut être fait tel quel, ou en tenant compte des permutations. L'utilisation de la couleur rend plus facile la distinction entre les éléments de L et U, et montre clairement l'avancement du calcul en cas de progression pas à pas.
Après la décomposition on peut résoudre des systèmes, calculer le conditionnement en norme infinie, calculer et évaluer la précision d'un résidu pour une amélioration itérative éventuelle.

4.1.3 Cas des matrices rectangulaires

Pour une matrice rectangulaire $m \geq n$ de rang n, la décomposition LU permet d'extraire de la matrice un système de Cramer correspondant aux lignes des pivots. On peut alors résoudre ce système et vérifier si les autres équations sont satisfaites.
On obtient :

$$PAQ = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} U$$

On résoud :

$$\begin{pmatrix} L_1 \\ L_2 \end{pmatrix} Ux' = P \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} b_1' \\ b_2' \end{pmatrix}, \quad x = Qx'$$

La solution de $L_1 y = b_1'$ doit vérifier $L_2' y - b_2' = 0$. Le logiciel évalue et affiche le nombre de chiffres corrects de cette différence, en tenant compte des erreurs d'arrondi. La solution peut être considére comme numériquement satisfaisante si ce nombre est 0, auquel cas la différence entre $L_2' y$ et b_2' est de l'ordre des erreurs d'arrondi.
L'utilisation de la décomposition QR permet de résoudre les systèmes $Ax = b$ au sens des moindres carrés et donne $\|Ax - b\|_2$.

4.1.4 Rang approché d'une matrice

La décomposition LL^* avec pivotage diagonal permet de déterminer le rang d'une matrice symétrique semi-définie positive : si à la k^o étape de l'élimination le bloc restant à transformer est de l'ordre des erreurs d'arrondi la matrice peut être considérée comme étant numériquement de rang $k - 1$.
La décomposition LU avec pivot total et la décomposition QR avec pivotage fournissent également le rang d'une matrice rectangulaire.

4.2 Calcul de valeurs propres

Les méthodes considérées sont :
• la méthode de la puissance et de la puissance inverse,
• la méthode de Jacobi pour les matrices symétriques, ou d'Eberlein pour les matrices non symétriques. L'utilisation du type intervalle fournit des bornes rigoureuses pour les valeurs propres [18], et une représentation graphique au moyen de disques de Gershgorin de régions contenant les valeurs propres.
• la méthode de bisection pour les matrices tridiagonales symétriques. Les matrices pleines sont mises sous forme tridiagonale à l'aide de matrices orthogonales élémentaires de Householder. Des bornes rigoureuses sont fournies pour les matrice tridiagonales symétriques [18].
• la méthode QR, pour les matrices de Hessenberg ou tridiagonales symétriques, avec translation d'origine. La mise sous forme de Hessenberg est faite en utilisant les orthogonales élémentaires de Householder.
La mise en œuvre privilégie l'interactivité et le contrôle du déroulement des algorithmes par l'utilisateur. Dans la méthode de Jacobi par exemple, les itérations peuvent être effectuées à la main, par balayage de la matrice, ou de manière totalement automatique. Dans le cas de la méthode QR on peut fixer la translation d'origine à effectuer et suivre l'apparition progressive de la forme triangulaire.

4.3 Algorithme du simplexe

L'algorithme du simplexe [19] est appliqué à la *maximisation* d'une forme linéaire :

$$cx = \sum_{j=1}^{n} c_j x_j$$

sous les contraintes :

$$\begin{cases} x \geq 0 \\ \sum\limits_{j=1}^{n} a_{ij} x_j \top b_i, \ 1 \leq i \leq m \end{cases}$$

où $\top$ désigne $\leq, \geq$ ou $=$.
Ces données doivent être rentrées dans un tableau de forme adaptée à l'application de l'algorithme du simplexe. La forme linéaire c occupe la première ligne, le second membre des contraintes la première colonne, et les coefficients a_{ij} le bloc principal du tableau :

	c_1	c_2	$\cdots$	c_n	
$\top b_1$	a_{11}	a_{12}	$\cdots$	a_{1n}	
$\top b_2$	a_{21}	a_{22}	$\cdots$	a_{2n}	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	
$\top b_m$	a_{m1}	a_{m2}	$\cdots$	a_{mn}	

Les contraintes $x \geq 0$ sont sous-entendues et ne doivent pas figurer dans ce tableau. L'espace libre à droite du tableau sert à l'introduction éventuelle de variables d'écart ou pénalisées.

4.3.1 Base de départ

Le logiciel peut calculer lui-même une base de départ en ajoutant des variables d'écart ou pénalisées.
Ce travail peut aussi être effectué par l'utilisateur. Il faut mettre les contraintes sous forme d'égalités et désigner par une sélection sur la première ligne les variables devant constituer la base. Le logiciel calcule alors le premier tableau du simplexe.
Supposons que les contraintes sont écrites sous la forme Ax=b, et désignons par I l'ensemble des indices choisis comme base de départ. En désignant par A_I la matrice extraite de A comprenant les colonnes dont l'indice appartient à I le logiciel calcule par pivotage (si c'est possible) :

v	$c - c_I A_I^{-1} A$
$A_I^{-1} b$	$A_I^{-1} A$

et indique si la base est primal-réalisable, dual-réalisable, ou ni l'un ni l'autre. v est la valeur de la forme linéaire au sommet correspondant à la base.
Lorsque les coordonnées basiques $(A^{-1}b)_i$ sont ≥ 0 la base est primal- réalisable et on applique l'algorithme primal du simplexe. Le maximum est atteint lorsque $c - c_I A_I^{-1} A \leq 0$.
Lorsque $c - c_I A_I^{-1} A \leq 0$ la base est dual-réalisable, et on peut appliquer l'algorithme dual. Le maximum est atteint lorsque $A^{-1} b \geq 0$.

4.3.2 Changement de base

Si la base est primal ou dual-réalisable le logiciel peut :
- poursuivre lui-même les itérations,
- faire une itération,
- laisser l'utilisateur choisir le changement de base à effectuer.

Le calcul est fait par pivotage à partir du tableau précédent, mais il est toujours possible de recalculer le tableau à partir des données initiales par pivotage.
Lorsqu'une base n'est ni primal ni dual-réalisable on peut faire des changements de base à la main, pour tenter d'atteindre une base permettant d'appliquer l'algorithme primal ou l'algorithme dual.

4.3.3 Utilisation des intervalles et des types stochastiques

L'algorithme fonctionne sur tous les types rationnels, intervalles ou stochastiques.
L'utilisation de rationnels permet une présentation élégante des petits programmes linéaires dont on illustre en général les cours sur l'algorithme du simplexe.
L'utilisation des types stochastiques permet par exemple d'éviter de faire entrer dans la base un élément i tel que $c_i - c_I A_I^{-1} A_i$ positif, mais voisin de 0, et qui représente en fait un zéro informatique.

4.3.4 Tableau de calcul condensé

La matrice $A_I^{-1}A$ contient une matrice identité (colonnes de la base). La représentation sous forme condensée élimine cette matrice identité. L'ordre des colonnes est modifié au cours des changements de base pour faciliter les calculs.

Références

[1] American National Standard Institute / Institute of Electrical and Electronics Engineers : A Standard for Binary Floating-Point Arithmetic. ANSI/IEEE Std. 754-1985, New York, 1985, Std. 854-1987, New York, 1987.

[2] Apple Numerics manual. Addison-Wesley (1988).

[3] Beaulac Y., Bellemain F., Laborde J.M. : Cabri-Géomètre, un logiciel d'aide à l'apprentissage de la géométrie. Cedic-Nathan, Paris (1988).

[4] Bennani M., Brunet M.C., Chatelin F. : De l'utilisation en calcul matriciel de modèles probabilistes pour la simulation des erreurs de calcul. C.R.A.S., Paris, t.307, Série I, p. 847-850, (1988).

[5] Chesneaux J.-M. : Study of the computing accuracy by using probabilistic approach. Contribution to Computer Arithmetic and Self-Validating Numerical Methods. Ed. C. Ulrich, (J.C. Baltzer), 1990, pp. 19-30.

[6] Chesneaux J.-M., Vignes J. : L'algorithme de Gauss en arithmétique stochastique. C.R.A.S., t.316, Série II, p.171-176, (1993).

[7] Crawford J.H., Gelsinger P.P. : Programmation du 80386. Sybex (1988).

[8] Durand R. : The use of Maths&Calculs-io and Mathematica in the teaching of mathematics to first and second year ingeneering students at INSAT, Toulouse. French-Norwegian Symposium on computer aided mathematics (1993)

[9] Groupe de Formation Recherche E.I.O. : Enseignement illustré par ordinateur. IREM de Brest (1992).

[10] Knuth D.E. : The art of programming. Vol. 2. Addison Wesley (1969).

[11] Kulisch U.W., Miranker W.L. : Computer arithmetic in theory and in practice. Academic Press, New York (1981).

[12] MIGNOT A., PAVEC R., ROUÉ G. : Quelques exemples d'illustration d'exercices de mathématiques sur micro-ordinateur. Taol Lagad, IREM de Brest, n^o 68 (1994).

[13] MOORE R.E. : Methods and applications of interval analysis; S.I.A.M. (1979).

[14] PAVEC R. : Illustration de cours de Mathématiques sur micro-ordinateur : un exemple de réalisation. Bulletin de l'E.P.I. n^o 64 (1991).

[15] PAVEC R. : A propos du logiciel Maths&Calculs-io. Taol Lagad, IREM de Brest, n^o 59 (1989), n^o 61 (1990), n^o 63 (1991), n^o 67 (1994).

[16] PAVEC R. : Maths&Calculs-io, a software for teaching, combining graphic representations and numerical methods. Proc. French-Norwegian Symposium on Computer-Aided Mathematics, (1993).

[17] PAVEC R. : Logiciel Maths&Calculs-io, mathématiques et calcul numérique illustrés par ordinateur. Editions JoliCiel, Noisiel, France (1993-96).

[18] PAVEC R. : Some algorithms providing rigourous bounds for the eigenvalues of a matrix. J.U.C.S. n^{o}7 (1995).

[19] PAVEC R. : Un logiciel d' aide à l'enseignement de l'algorithme du simplexe. Colloque du groupe MODE. Brest (1995).

[20] POPOVA E.D. : On a formally correct implementation of IEEE computer arithmetic. J.U.C.S. n^{o}7 (1995).

[21] PICHAT M., VIGNES J. : Ingéniérie du contrôle de la précision des calculs sur ordinateur. Editions Technip (1993).

[22] Turbo-Pascal, compilateur Pascal. Borland International.

[23] VIGNES J. : Zéro mathématique et zéro informatique. C.R.A.S., Paris, t.303, Série I, n^{o}20 (1986).

[24] VIGNES J. : Contrôle et estimation stochastique des arrondis de calcul. AFCET Interfaces, n^{o}54 (1987).