Les fonctions interpretees.Manuel d'utilisation ,de programmation,de reference

Patrick Laug

INRIA

CENTRE DE ROCQUENCOURT

Rapports Techniques

N° 38

LES FONCTIONS INTERPRETÉES

MANUEL D'UTILISATION, DE PROGRAMMATION, DE RÉFÉRENCE

Patrick LAUG

Juin 1984

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

LES FONCTIONS INTERPRETEES

Manuel d'utilisation,
de programmation,
de référence.

Juin 1984

Patrick Laug

Sommaire

Résumé

Le logiciel présenté ici permet à un programme d'appel d'analyser et d'interpréter des fonctions durant l'exécution. Cet interpréteur, conçu et réalisé à l'INRIA, fait partie de la bibliothèque scientifique Modulef.

Abstract

The software presented here allows a calling program to analyse and interpret functions during execution. This interpreter, designed and realised at INRIA, is a part of the Modulef scientific library.

1. Introduction

Un programme numérique est rarement autonome. A l'exécution, il demande une intervention de la part de l'utilisateur : lecture de données sur cartes, sur console, ou sur un support magnétique, ou encore appel de fonctions que l'utilisateur doit définir. Par exemple, le module de maillage APNOPO [Modulef 104] demande l'expression analytique de chaque frontière courbe (de la forme x**2 + y**2 - 1 pour le cercle unité). Il existe aussi des programmes interactifs qui tracent le graphe d'une fonction quelconque à définir.

Pour réaliser ce type de programmes en Fortran, une solution est d'utiliser le concept de FUNCTION existant dans ce langage :

- Le programme appelle une fonction ayant un nom conventionnel, par exemple UTILIS :

```
R = UTILIS(X,Y,Z)
```

- L'utilisateur écrit cette fonction en Fortran :

```
FUNCTION UTILIS(X,Y,Z)
UTILIS = ...
END
```

Cette solution, qui semble naturelle en Fortran, présente cependant plusieurs inconvénients :

- Chaque fois que l'utilisateur désire lancer le programme avec une autre fonction (traitement d'un nouveau problème ou légère modification), il lance la compilation de la nouvelle fonction, ce qui implique une perte de temps homme et machine. En outre, sur la plupart des systèmes, il active également l'édition de liens du programme complet, ce qui est particulièrement coûteux en temps U.C. et en nombre d'entrées/sorties.

- Il est impossible pour l'utilisateur de changer la fonction en cours d'exécution. Par exemple, un programme de tracé ne peut visualiser qu'un nombre de graphes prédéfini avant l'appel.

Pour pallier ces inconvénients, nous proposons ici une autre solution, où les fonctions-utilisateurs sont fournies dans les données mêmes du programme. Leur définition est analysée et stockée en mémoire sous forme codée. Ensuite, chaque fonction est exécutée à la demande, autant de fois que nécessaire, en interprétant les instructions mémorisées.

Certes, le temps d'exécution est alors légèrement augmenté, du fait de l'interprétation des instructions. En revanche, tous les inconvénients décrits précédemment disparaissent : la gestion des fichiers de fonctions Fortran est supprimée (sources et objets), la phase d'édition de liens est faite une fois pour toutes, et la souplesse d'utilisation est accrue.

Ce rapport est divisé en trois parties, qui peuvent être étudiées indépendamment selon les motivations du lecteur : le manuel d'utilisation (quelles données l'utilisateur doit-il taper ?), le manuel de programmation (comment écrire un programme qui appelle l'interpréteur ?), et le manuel de référence (comment fonctionne l'interpréteur ?).

2. Manuel d'utilisation

La définition des fonctions est fournie dans les données, en format libre [Modulef 44], sur une ou plusieurs lignes, en colonnes 1 à 72. La forme générale est la suivante :

```
fonction (paramètre1, paramètre2, ...)  = expression ;
```

Exemple :

```
F(X,Y) = 3.14 * COS(X+Y)
       + X / (1 - (X + 36*Y)) ;
```

Les lecteurs connaissant le langage Fortran peuvent se contenter d'étudier le paragraphe 2.1 ci-dessous. Il est demandé aux autres lecteurs de se reporter directement au paragraphe 2.2, dans lequel la syntaxe et la sémantique des expressions sont expliquées en détail.

2.1 Comparaison avec Fortran

La syntaxe est voisine de celle des "statement functions" Fortran. Hormis les quelques incompatibilités décrites plus bas, toute expression Fortran de type réel simple précision, parenthésée ou non, est acceptée avec la même sémantique. Les conventions concernant la priorité des opérateurs et l'ordre d'évaluation sont identiques. Par exemple,

A/B*C est évalué en divisant d'abord A par B, puis en multipliant le résultat par C, c'est-à-dire comme (A/B)*C

A+B*C est évalué comme A+(B*C)

Insistons cependant sur les différences suivantes :

- La "zone instruction" commence en colonne 1 et non en colonne 7.

- Si la définition d'une fonction nécessite l'emploi de plusieurs lignes, on n'emploie pas de "caractère suite" comme en Fortran. Ecrire simplement ligne après ligne, comme en Pascal, PL/1, ADA, ... Bien noter que chaque définition de fonction se termine par un point-virgule.

- Tous les opérandes sont considérés comme des objets de type REAL (réel simple précision) : les paramètres (quelle que soit la première lettre de l'identificateur) et les constantes numériques (36 et 36. sont équivalents). Ainsi, 1/2 retourne 0.5 et non 0.

- L'exponentiation suit la règle générale d'évaluation de gauche à droite : A**B**C est évalué comme (A**B)**C, contrairement à Fortran !

- Les noms des fonctions intrinsèques sont réservés, et ne peuvent donc pas être employés pour identifier un paramètre. La liste des identificateurs réservés est donnée dans ce tableau :

ABS	ACOS	ASIN	ATAN	COS	COSH	DBLE	EXP	FIN	INT	LOG
LOG10	MAX	MIN	MOD	REAL	SIN	SINH	SQRT	TAN	TANH	

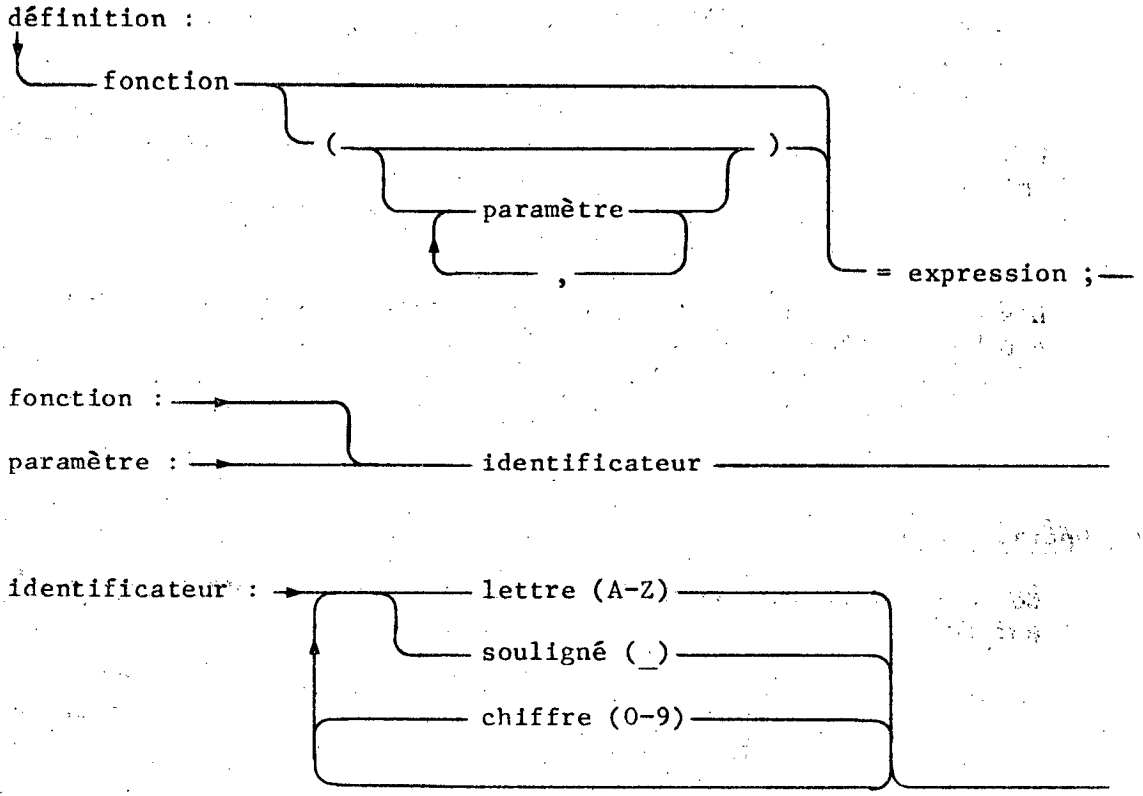
- Un certain nombre d'extensions sont autorisées, comme par exemple l'emploi d'identificateurs de plus de 6 caractères. Le lecteur désirant utiliser toutes ces possibilités est prié de se reporter au paragraphe suivant.

2.2 Description détaillée

Rappelons la forme générale :

fonction (paramètrel paramètre2, ...) = expression ;

Cette syntaxe est symbolisée par le diagramme suivant :



Un "identificateur" est une suite non vide de lettres (A-Z), de soulignés (_), et de chiffres (0-9). Le premier caractère est obligatoirement une lettre ou un souligné. La longueur d'un identificateur est limitée uniquement par celle des lignes d'entrée, soit 72 caractères. L'analyseur syntaxique vérifie que tout identificateur est différent des mots réservés énumérés plus haut :

```
ABS   ACOS  ASIN  ATAN  COS   COSH  DBLE  EXP   FIN   INT   LOG
LOG10 MAX   MIN   MOD   REAL  SIN   SINH  SQRT  TAN   TANH
```

"fonction" est l'identificateur de la fonction à définir, considérée comme réelle simple précision.

"paramètrel, paramètre2, ..." sont les identificateurs des paramètres formels pouvant apparaître dans l'expression, ces paramètres étant considérés comme des variables réelles simple précision.

"expression" est une construction formée d'opérandes, d'opérateurs, et de parenthèses, retournant une valeur réelle simple précision. Les opérandes sont des paramètres ou des constantes numériques, et les opérateurs sont des symboles "infixés" (placés entre les deux opérandes qu'ils concernent), ou "préfixés" (placés devant le ou les opérandes qu'ils concernent). Les priorités de ces opérateurs sont indiquées ci-dessous.

Un point-virgule termine la définition de chaque fonction.

Opérateurs infixés

Ce sont les opérateurs à placer entre les deux opérandes sur lesquels ils portent. On leur associe une priorité, qui détermine l'ordre d'évaluation de l'expression :

Opérateur	Symbole	Priorité
addition, soustraction	+ -	2
multiplication, division	* /	3
exponentiation	**	5

Les opérateurs adjacents de priorité plus forte sont les premiers à être pris en compte :

A+B*C est évalué en multipliant d'abord B par C, puis en ajoutant le résultat à A, c'est-à-dire comme A+(B*C)

Les opérateurs infixés adjacents de même priorité sont tous considérés de gauche à droite, y compris l'exponentiation :

A/B*C est évalué comme (A/B)*C

A**B**C est évalué comme (A**B)**C

Opérateurs préfixés

Ce sont les opérateurs à placer devant le ou les opérandes sur lesquels ils portent :

Opérateur	Symbole	Nombre d'opérandes	Priorité
plus unaire, moins unaire	+ -	1	4
conversion en entier, réel s.p., d.p.	INT REAL DBLE	1	6
valeur absolue	ABS	1	6
racine carrée	SQRT	1	6
exponentielle	EXP	1	6
logarithme népérien, logarithme base 10	LOG LOG10	1	6
sinus, cosinus, tangente	SIN COS TAN	1	6
arc sinus, arc cosinus, arc tangente	ASIN ACOS ATAN	1	6
sinus hyperbolique, cos. hyp., tg. hyp.	SINH COSH TANH	1	6
reste	MOD	2	6
minimum, maximum	MIN MAX	2	6

Par définition des opérateurs préfixés, on pourra écrire :

-A ABS A MOD A B

La forme : opérateur(opérandel,opérande2,...), qui visualise mieux l'ordre d'évaluation, est aussi autorisée :

ABS(A) MOD(A*B,C)

MAX(MIN A B, COS C) ou MAX(MIN(A,B), COS(C)) calcule le maximum entre le résultat de MIN(A,B) et le résultat de COS(C).

D'après les priorités des opérateurs infixés et préfixés,

-A+B est évalué comme (-A)+B

-A**B est évalué comme -(A**B)

Des opérateurs préfixés adjacents de même priorité sont considérés de droite à gauche :

SIN COS A est évalué comme SIN(COS A)

Variables

Les variables ne peuvent être désignées que par des identificateurs de paramètres formels (ici "paramètre1" et "paramètre2"). Le type réel simple précision leur est associé.

Constantes numériques

Les constantes entières ou réelles simple précision sont acceptées :

```
1234   5.6   7.89E12
```

Les constantes peuvent être signées. Ainsi, +6.7 et -8.9 sont considérées comme des constantes en elles-mêmes, ayant leurs propres représentations internes, et ne demandent pas à l'exécution l'appel des opérateurs unaires plus et moins.

Optimisation

Lorsque l'utilisateur demande une élévation au carré, l'opérateur d'exponentiation n'est pas appelé : en fait, le résultat de l'expression qui précède **2 est multiplié par lui-même, ce qui optimise généralement le temps d'exécution et la précision du calcul. Ainsi, ne pas hésiter à taper X**2 plutôt que X*X !

3. Manuel de programmation

Le but de ce paragraphe est de montrer la manière d'écrire un programme numérique qui utilise les fonctions interprétées.

3.1 Exemple

But : imprimer les valeurs d'une fonction F(X) pour X = 0.0, 0.1, ..., 1.0 :

Programme Fortran

```
      DIMENSION M(150)
 1000 FORMAT (' F(',F3.1,') = ',F10.8)
C
C INITIALISATIONS
      CALL INITI (M,150,0,0)
      CALL FONINI(M,100,ICODE)
C
C LECTURE DE LA DEFINITION DE F(X)
      WRITE (IINFO('I'),*) 'FONCTION ?'
      CALL FONDES(M,IFON,ICODE)
C
C INTERPRETATION DE F(X) POUR X = 0.0, 0.1, ..., 1.0
      WRITE (IINFO('I'),*) ' '
      DO 100 X = 0.0, 1.0, 0.1
         CALL FON1RR(M,IFON,X,R,ICODE)
         WRITE (IINFO('I'),1000) X,R
  100 CONTINUE
      END
```

Exécution

L'utilisateur tape le titre lu par le sous-programme INITI, puis définit une fonction. Dans l'exemple ci-dessous, les lignes tapées sont précédées d'une flèche (->) :

```
   TITRE DU TRAVAIL ?
-> DEMO
   FONCTION ?
-> F(X) = 2 * ASIN X ;

   F(0.0) = 0.00000000
   F(0.1) = 0.20033484
   F(0.2) = 0.40271584
   F(0.3) = 0.60938531
   F(0.4) = 0.82303371
   F(0.5) = 1.04719757
   F(0.6) = 1.28700224
   F(0.7) = 1.55079502
   F(0.8) = 1.85459046
   F(0.9) = 2.23953909
   F(1.0) = 3.14159268
```

3.2 Présentation

Comme dans tout programme Modulef, il est nécessaire d'initialiser l'allocation dynamique des tableaux et la lecture en format libre. Un moyen simple est d'appeler le sous-programme INITI [Modulef 1].

Ensuite, le sous-programme FONINI initialise l'interpréteur de fonctions. Il peut être appelé plusieurs fois, effaçant alors les informations mémorisées auparavant.

Pour analyser une fonction définie par l'utilisateur, le sous-programme FONDEF, ainsi que ses variantes FONDES et FONDEN, sont proposés. A chaque appel, une nouvelle fonction est mémorisée, et le paramètre de sortie IFON repère cette fonction en mémoire.

Les sous-programmes suivants exécutent une fonction définie par l'utilisateur : FONORR, FON1RR, FON2RR, FON3RR (retour d'une valeur réelle à partir de 0, 1, 2, ou 3 arguments réels). D'autres sous-programmes pourront être développés à la demande.

L'utilitaire FONDRL a été écrit pour les programmes de visualisation. Il permet d'obtenir la dernière ligne lue, afin de l'afficher sur écran.

Enfin, FONTER détruit les tableaux dynamiques [Modulef 1] nécessaires à l'interpréteur.

Le paragraphe ci-dessous détaille le but et les paramètres de ces utilitaires, classés par ordre alphabétique.

3.3 Description des sous-programmes

FONORR

```
      SUBROUTINE FONORR(M,IFON,R,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     EXECUTER UNE FONCTION AYANT 0 PARAMETRE ET UN RESULTAT REEL
C
C PARAMETRES D ENTREE :
C     M       : SUPER-TABLEAU
C     IFON    : INDICE DE LA FONCTION (RETOURNE PAR FONDEF)
C
C PARAMETRES DE SORTIE :
C     R       : RESULTAT DE LA FONCTION
C     ICODE   : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FON1RR

```
      SUBROUTINE FON1RR(M,IFON,X,R,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     EXECUTER UNE FONCTION AYANT 1 PARAMETRE REEL ET UN RESULTAT REEL
C
C PARAMETRES D ENTREE :
C     M        : SUPER-TABLEAU
C     IFON     : INDICE DE LA FONCTION (RETOURNE PAR FONDEF)
C     X        : PARAMETRE DE LA FONCTION
C
C PARAMETRES DE SORTIE :
C     R        : RESULTAT DE LA FONCTION
C     ICODE    : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FON2RR

```
      SUBROUTINE FON2RR(M,IFON,X,Y,R,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     EXECUTER UNE FONCTION AYANT 2 PARAMETRES REELS ET UN RESULTAT REEL
C
C PARAMETRES D ENTREE :
C     M        : SUPER-TABLEAU
C     IFON     : INDICE DE LA FONCTION (RETOURNE PAR FONDEF)
C     X,Y      : PARAMETRES DE LA FONCTION
C
C PARAMETRES DE SORTIE :
C     R        : RESULTAT DE LA FONCTION
C     ICODE    : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FON3RR

```
      SUBROUTINE FON3RR(M,IFON,X,Y,Z,R,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     EXECUTER UNE FONCTION AYANT 3 PARAMETRES REELS ET UN RESULTAT REEL
C
C PARAMETRES D ENTREE :
C     M        : SUPER-TABLEAU
C     IFON     : INDICE DE LA FONCTION (RETOURNE PAR FONDEF)
C     X,Y,Z    : PARAMETRES DE LA FONCTION
C
C PARAMETRES DE SORTIE :
C     R        : RESULTAT DE LA FONCTION
C     ICODE    : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FONDEF

```
      SUBROUTINE FONDEF(M,CFON,IFON,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     ANALYSER LA DEFINITION D UNE FONCTION
C     <DEFINITION> ::= <SPECIFICATION> <CORPS>
C
C PARAMETRES D ENTREE :
C     M       : SUPER-TABLEAU
C
C PARAMETRES DE SORTIE :
C     CFON    : IDENTIFICATEUR DE LA FONCTION
C     IFON    : INDICE DE LA FONC. (NECESSAIRE POUR L EXECUTER ENSUITE)
C     ICODE   : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
      CHARACTER*(*) CFON
```

FONDEN

```
      SUBROUTINE FONDEN(M,MCOU,LCOU,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     ANALYSER LES DEFINITIONS DE FONCTIONS NUMEROTEES
C     <IDENTIFICATEUR DE FONCTION NUMEROTEE> ::= <PREFIXE> <NUMERO>
C     <PREFIXE> ::= SUITE NON VIDE DE LETTRES (A-Z) ET DE SOULIGNES (_)
C     <NUMERO>  ::= SUITE NON VIDE DE CHIFFRES
C     LES FONCTIONS SONT DEFINIES SUCCESSIVEMENT, JUSQU A RENCONTRE
C     DE L'ITEM < OU FIN
C
C PARAMETRES D ENTREE :
C     M       : SUPER-TABLEAU
C     MCOU    : TABLEAU MODIFIE PAR CE S.P. DE TELLE SORTE QUE,
C               EN SORTIE, MCOU(I) EST L'INDICE DE LA FONCTION I
C               (NECESSAIRE POUR L EXECUTER ENSUITE)
C     LCOU    : NUMERO MAXIMAL DES FONCTIONS A ANALYSER
C               L'ENTIER REPRESENTE PAR <NUMERO> DOIT ETRE COMPRIS
C               ENTRE 1 ET LCOU
C
C PARAMETRES DE SORTIE :
C     ICODE   : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*),MCOU(*)
```

FONDES

```
      SUBROUTINE FONDES(M,IFON,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     ANALYSER LA DEFINITION D UNE FONCTION SANS RETOURNER SON IDENT.
C
C PARAMETRES D ENTREE :
C     M       : SUPER-TABLEAU
C
C PARAMETRES DE SORTIE :
C     IFON    : INDICE DE LA FONC. (NECESSAIRE POUR L EXECUTER ENSUITE)
C     ICODE   : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FONDRL

```
      SUBROUTINE FONDRL(LIGNE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     RETOURNER LA DERNIERE LIGNE LUE
C
C PARAMETRES DE SORTIE :
C     LIGNE    : DERNIERE LIGNE LUE
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      CHARACTER*(*) LIGNE
```

FONINI

```
      SUBROUTINE FONINI(M,LFON,ICODE)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     INITIALISER L ANALYSEUR DE FONCTIONS
C
C PARAMETRES D ENTREE :
C     M        : SUPER-TABLEAU
C     LFON     : NOMBRE DE MOTS ACCORDES A L ANALYSEUR
C
C PARAMETRES DE SORTIE :
C     ICODE    : CODE DE RETOUR (VOIR LE S.P. FONERR)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

FONTER

```
      SUBROUTINE FONTER(M)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     LIBERER DE LA PLACE MEMOIRE LORSQUE L'UTILISATION
C     DES FONCTIONS INTERPRETEES EST TERMINEE
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
```

Codes d'erreur

Un code de retour (ICODE) est retourné par la plupart des sous-programmes. Si aucune erreur n'est apparue (exécution normale), la valeur retournée est 0. Sinon, un code non nul est retourné, et le message correspondant est éventuellement imprimé par le sous-programme FONERR. Le tableau ci-dessous donne les messages correspondant à différentes valeurs de ICODE (chaque message commence par le nom du sous-programme qui a détecté une erreur).

1	FONACS: CONSTANTE INCORRECTE
2	FONSPE: NOM DE FONCTION TROP LONG
3	FONNRR: LE 3E PARAMETRE NE DOIT PAS ETRE REEL
4	FONCID: IDENTIFICATEUR NON TROUVE
5	FONCOR: IDENTIFICATEUR NE DESIGNANT PAS UN SCALAIRE
6	FONSPE: TYPE DES CONSTANTES ILLEGAL
7	FONCOR: PAS ASSEZ DE PARENTHESES OUVRANTES
8	FONCOR: PAS ASSEZ DE PARENTHESES FERMANTES
9	FONCOR: DEBORDEMENT PILE DES OPERATEURS
10	FONCOR: DEBORDEMENT TABLE DES EXPRESSIONS
11	FONCOR: PAS ASSEZ D OPERANDES
12	FONCOR: PAS ASSEZ D OPERATEURS
13	FON2RR,FON3RR: EXECUTION FONCTION ERRONEE
14	FONINI: LONGUEUR FOURNIE TROP FAIBLE
15	FONINT: INSTRUCTION INCONNUE
16	FONINT: DEBORDEMENT DE LA PILE DES OPERANDES
17	FONOP1: INSTRUCTION INCORRECTE
18	FONOP2: INSTRUCTION INCORRECTE
19	FONSPE: NOM DE L EXPRESSION INCORRECT
20	FONSPE: NOM DE L EXPRESSION NON SUIVI DE (
21	FONSPE: NOM DE PARAMETRE FORMEL INCORRECT
22	FONAID: IDENTIFICATEUR DEJA EXISTANT
23	FONSPE: PARAMETRE FORMEL NON SUIVI DE ,
24	FONSPE: NOM DE PARAMETRE FORMEL INCORRECT
25	FONSPE: UN = DOIT SUIVRE LA LISTE DE PARAMETRES
26	FONTRO: IDENTIFICATEUR NE DESIGNANT PAS UNE FONCTION
27	FONNRR: LE 1E PARAMETRE NE DOIT PAS ETRE REEL
28	FONNRR: LE 2E PARAMETRE NE DOIT PAS ETRE REEL
29	FONNRR: NOMBRE DE PARAMETRES INCORRECT
30	FONNRR: RESULTAT NON REEL
31	FONAID: DEBORDEMENT TABLE DES IDENTIFICATEURS
32	FONSPE: TYPE DES CONSTANTES NON SUIVI DE ;

3.4 Application à Modulef

Avec le module APNOPO [Modulef 104], l'utilisateur peut donner les équations des courbes d'un maillage sans recourir à une fonction Fortran (mot-clé COURBES). Ce paragraphe montre comment est programmée cette partie de APNOPO.

Le module APNOPO mémorise les équations grâce à la série d'instructions :

```
      SUBROUTINE APNOPO(M,XM)
      DIMENSION M(*),XM(*)
      COMMON /COMCOU/ NCOU,IACOU,LCOU
         ......
   11 CALL LECTU1(1,LCOU)
      CALL FONINI(M,500+40*LCOU,ICODE)
      IF ( ICODE .GT. 0 ) GO TO 112
      NCOU  = ICHAR4(':COU')
      IACOU = 0
      CALL READRE(1,CHAR4(NCOU),IACOU,LCOU,M,IRET)
      DO 111 I = 1,LCOU                      -- Cette initialisation permet
         M(IACOU-1+I) = -1                   --- à FFRONT de détecter
  111 CONTINUE                               -- les courbes non définies.
      CALL FONDEN(M,M(IACOU),LCOU,ICODE)
      IF ( ICODE .EQ. 0 ) GO TO 2000
  112 STOP
```

La fonction FFRONT est écrite de la façon suivante :

```
      FUNCTION FFRONT(ICOU,X,Y)
      COMMON M(1)
      FFRONT = FFRON1(M,ICOU,X,Y)
      END

      FUNCTION FFRON1(M,ICOU,X,Y)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C BUT :
C     EXECUTER UNE FONCTION FFRONT (APPELEE PAR LES MODULES DE APNOPO)
C     FOURNIE PAR L'UTILISATEUR DANS LES DONNEES
C     DANS CE S.P., M EST UN PARAMETRE DE DIMENSION INCONNUE
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      DIMENSION M(*)
      COMMON /COMCOU/ NCOU,IACOU,LCOU
C
C IFON = INDICE DE LA FONCTION DECRIVANT LA COURBE NUMERO ICOU
      IF (ICOU.LE.0 .OR. ICOU.GT.LCOU) THEN
         WRITE(IINFO('I'),*) ' FFRON1: NUMERO DE COURBE = ',ICOU,
         ' <= 0 OU > ',LCOU
         STOP
      END IF
      IFON = M(IACOU-1+ICOU)
      IF (IFON.LT.0) THEN
         WRITE(IINFO('I'),*) ' FFRON1: COURBE NUMERO ',ICOU,
         ' NON DEFINIE'
         STOP
      END IF
C
C CALCUL DE LA FONCTION D'INDICE IFON
      CALL FON2RR(M,IFON,X,Y,R,ICODE)
      FFRON1 = R
      IF (ICODE.NE.0) STOP
      END
```

Remarque : le super-tableau M n'apparaît pas dans les paramètres de la fonction FFRONT (car il est inutile lorsque l'utilisateur écrit lui-même cette fonction). Pour pouvoir malgré tout utiliser le super-tableau, celui-ci doit être mis en commun blanc dans le programme principal.

Durant l'exécution de APNOPO, l'utilisateur tape par exemple :

```
COURBES
0 2      $ IMPRE NFONC $
COURBE1(X,Y) = (X-.125)**2 + (Y-1)**2 - (COS 0.7)**2 ;
COURBE2(X,Y) = X**2 + Y**2 - .0625 ,
FIN
```

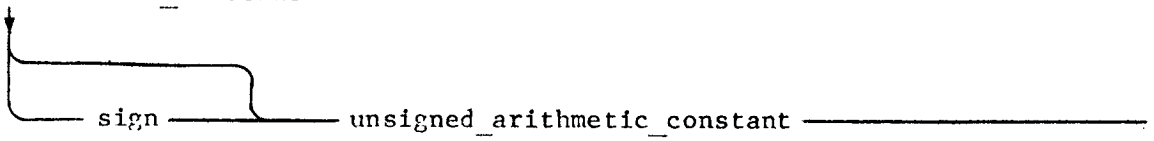
4. Manuel de référence

Pour des raisons de portabilité, l'analyseur et l'interpréteur sont entièrement écrits en Fortran 77. Le programme source, qui est fourni à tous les membres du Club Modulef [Modulef 85], contient de nombreux commentaires explicatifs. Dans ce paragraphe, nous nous bornons à présenter certains diagrammes qui illustrent le fonctionnement de l'analyseur lexicographique (scanner) et syntaxique.

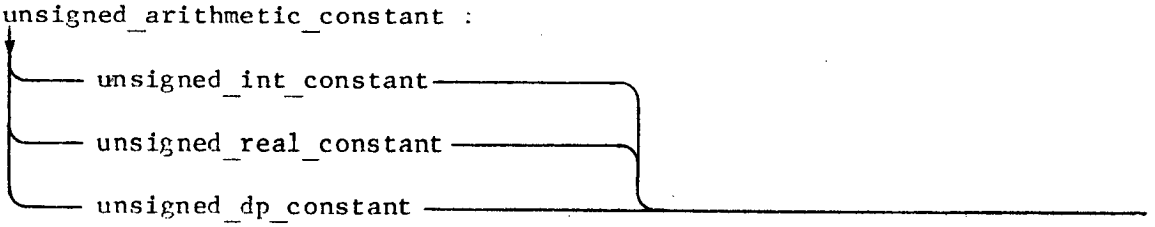
4.1 Constantes arithmétiques reconnues par le scanner

La structure des constantes arithmétiques est la même qu'en Fortran 77. Nous indiquons ci-dessous la partie de la norme qui décrit ces constantes [Fortran, Appendix F], en omettant les constantes de type COMPLEX.

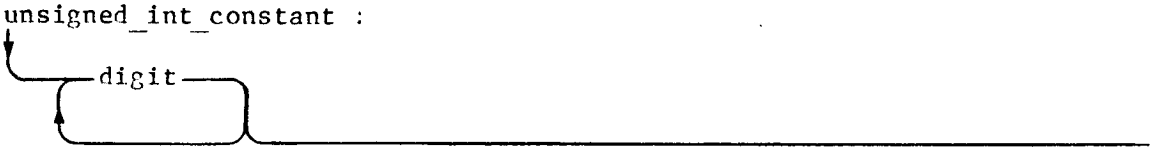
100 arithmetic_constant :



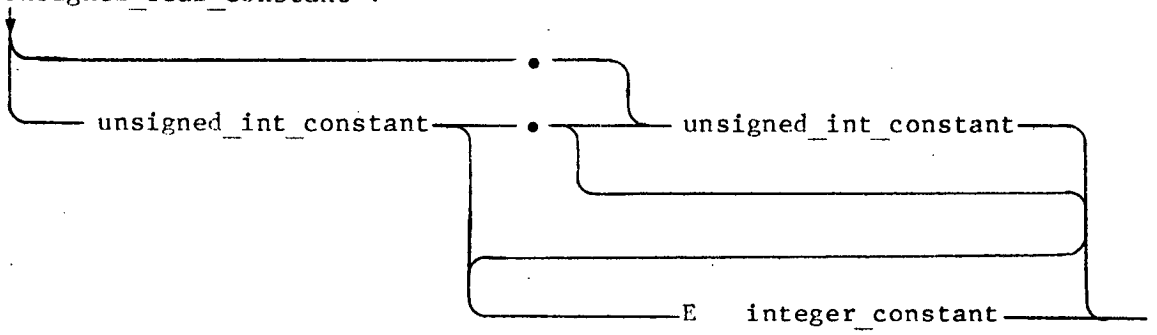
101 unsigned_arithmetic_constant :



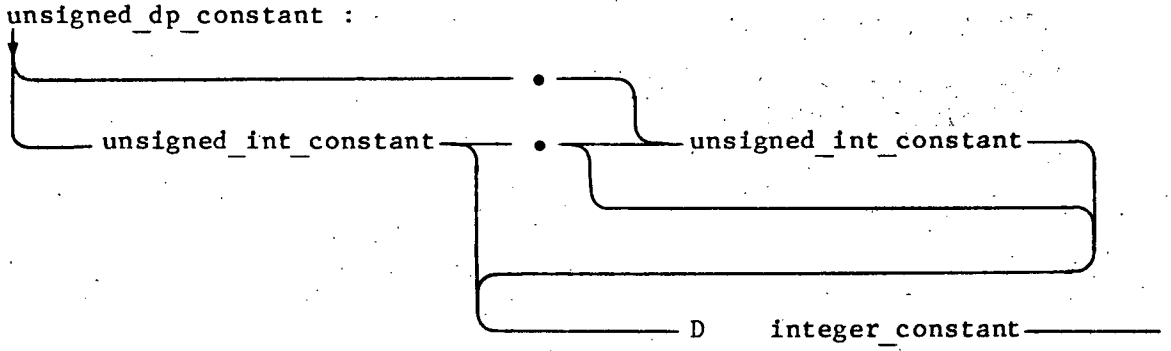
102 unsigned_int_constant :



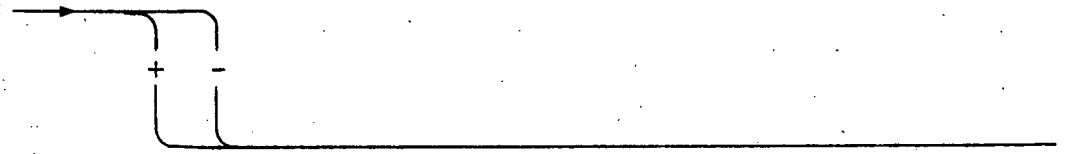
105 unsigned_real_constant :



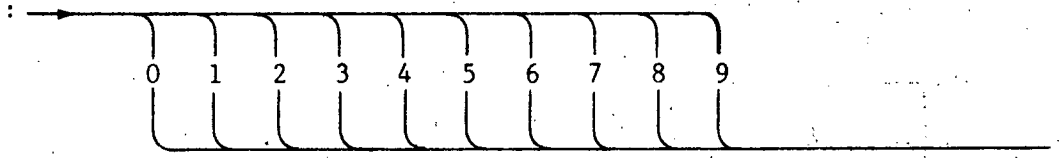
106 unsigned_dp_constant :



114 sign :

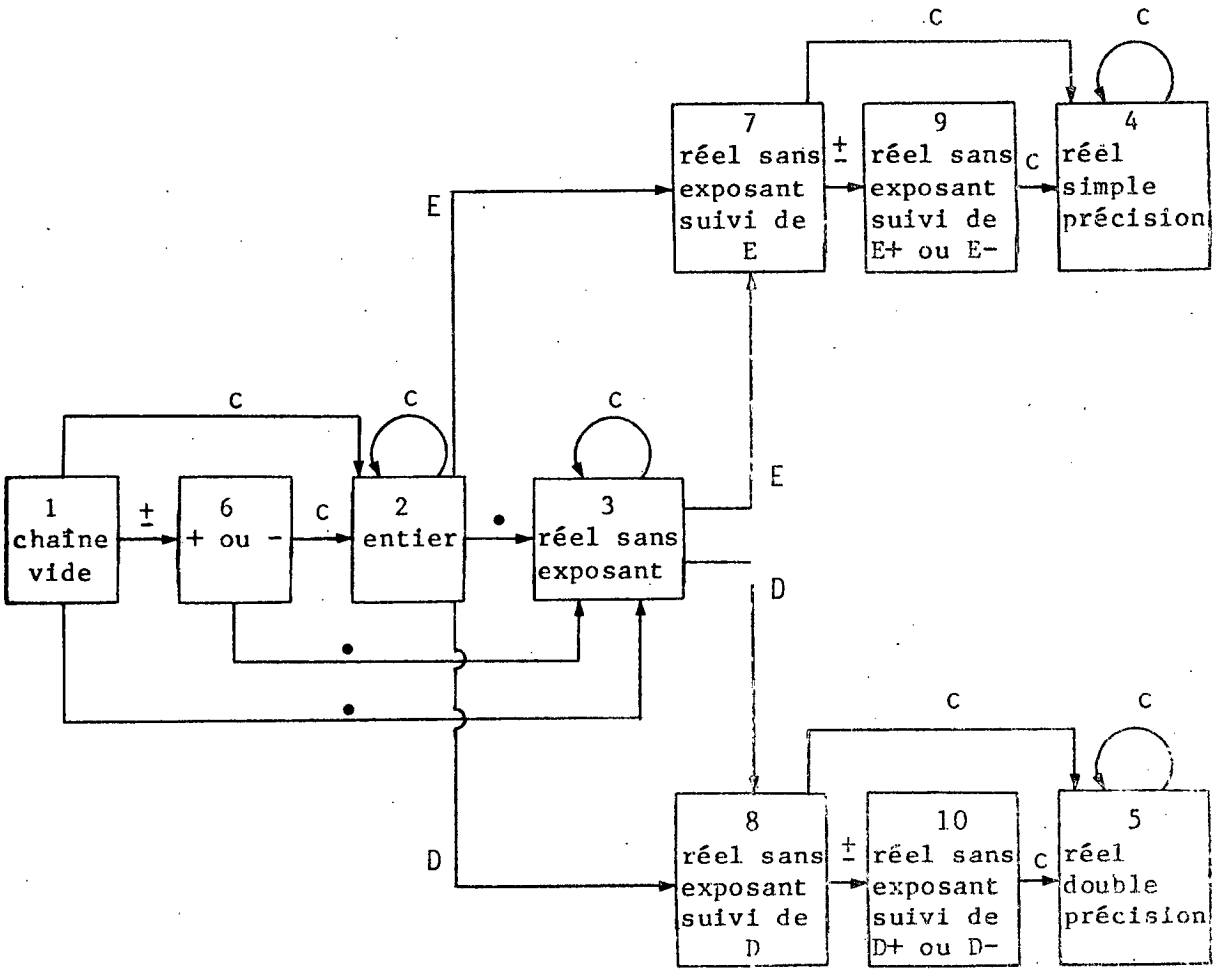


115 digit :



4.2 Algorithme de reconnaissance des constantes arithmétiques

Le sous-programme FONAUT, qui reconnaît les constantes arithmétiques décrites précédemment, est programmé comme un automate à 10 états :



c représente tout chiffre de 0 à 9

L'état initial est 1

Les états finals sont 1, 2, 3, 4, 5.

4.3 Analyse syntaxique des expressions

Les expressions sont mémorisées sous forme polonaise inverse, ce qui facilite la phase d'interprétation. Par exemple, l'expression notée sous forme habituelle :

A * (B + C)

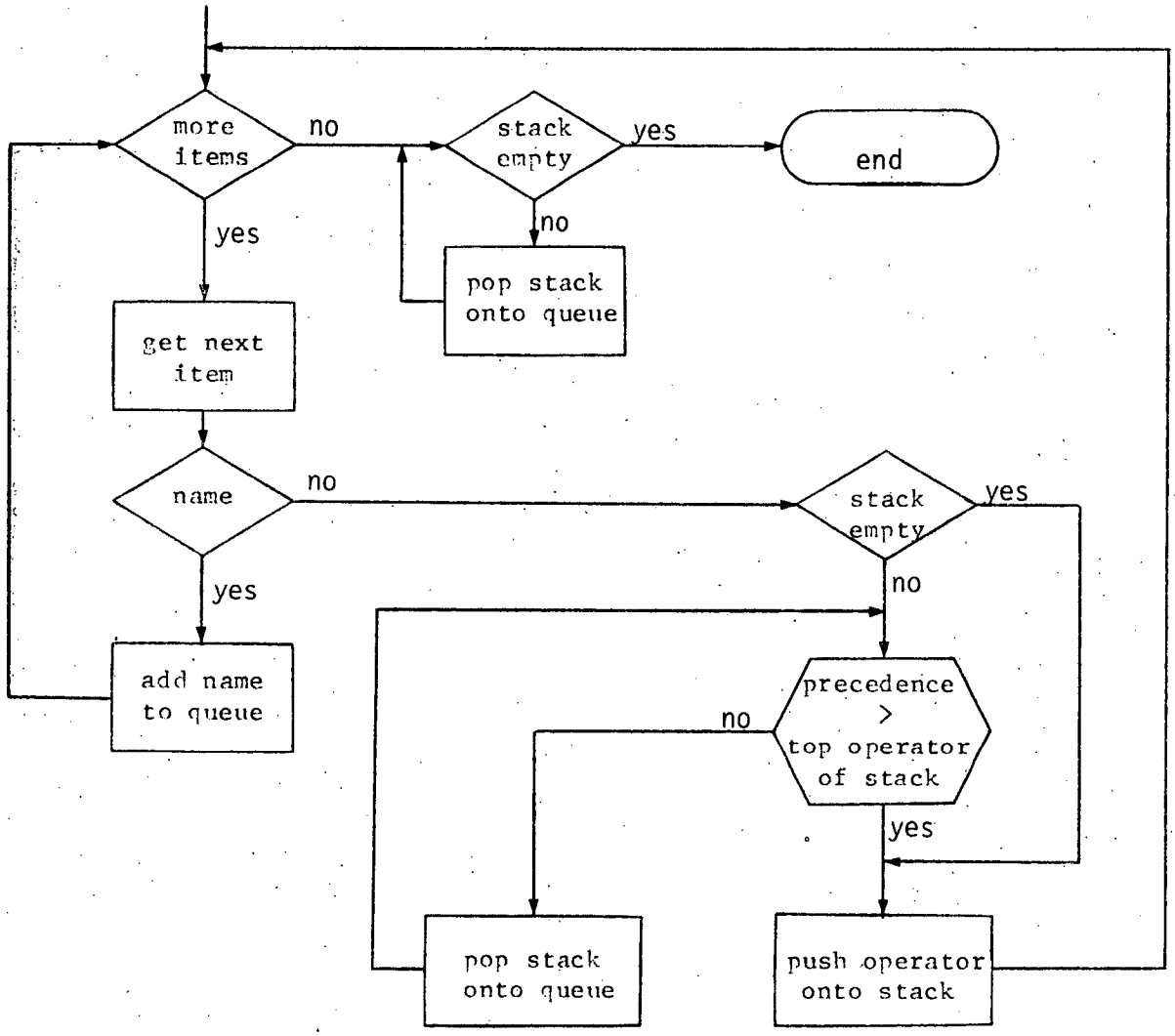
s'écrit en notation polonaise inverse :

A B C + *

A l'exécution,
- A, B, C sont empilés,
- C et B sont dépilés, et le résultat R1 = B + C est empilé,
- R1 et A sont dépilés, et le résultat R2 = A * R1 est empilé.

Finalement, la pile contient un seul élément, qui est le résultat final R2.

Un organigramme, qui représente l'algorithme de transformation d'une expression conventionnelle (mais non parenthésée) en expression polonaise inverse, est donné dans [Elson, Appendix 1.3]. Le schéma ci-dessous reproduit cet organigramme, mais en considérant tous les opérateurs comme associatifs de gauche à droite :



L'expression est lue élément par élément ("item"). Les opérateurs sont éventuellement stockés dans une pile intermédiaire ("stack"), et l'expression finale est déversée dans une "queue".

Le sous-programme FONCOR reprend cet algorithme en permettant de plus l'emploi de parenthèses et l'appel de fonctions intrinsèques SIN(X), MIN(X,Y), ...

5. Conclusion

Actuellement, l'interpréteur de fonctions est déjà utilisé avec succès dans de nombreuses applications. Cependant, deux principales améliorations sont maintenant envisagées :

Jusqu'ici, les expressions sont mémorisées, puis interprétées un certain nombre de fois. Il serait également possible d'interpréter directement les expressions, notamment lors de la lecture de chaque constante numérique. Ceci permettrait de définir plus facilement et plus clairement ces constantes (1/3, PI/2, SQRT(5), ...), ou encore de paramétrer les données en fonction de quelques variables (nombre de points sur une ligne par exemple).

Dans la version actuelle, le langage d'entrée est limité à la définition de fonctions arithmétiques. Il serait préférable d'admettre un langage quelconque, défini par une grammaire pourvue d'actions sémantiques. Cette solution présenterait plusieurs avantages :
- l'interpréteur serait plus général, et n'aurait plus à gérer les priorités des opérateurs, ou à effectuer des tests auxiliaires pour détecter les expressions incorrectes,
- le langage d'entrée serait facilement modifiable,
- la grammaire fournirait des spécifications précises, à la fois pour l'utilisateur et pour l'implémenteur du langage,
- l'implémenteur pourrait définir des "langages orientés problèmes" munis des structures de contrôle classiques IF THEN ELSE, ...

6. Bibliographie

Elson
M. Elson, "Concepts of Programming Languages", Science Research Associates, 1973.

Fortran
"Programming Language Fortran", Norme ANSI X3.9 - 1978.

Modulef 1
P.L. George, M. Vidrascu, "Guide d'Utilisation et Normes de Programmation", brochure Modulef no 1, INRIA, mars 1984.

Modulef 44
P. Laug, "Lecture de Données en Format Libre", brochure Modulef no 44, INRIA, janvier 1979.

Modulef 85
"Présentation du Club Modulef", brochure Modulef no 85, INRIA, mai 1984.

Modulef 104
P.L. George, "Le Module APNOPO", brochure Modulef no 104, INRIA, mai 1984.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique